Décomposition modulaire des graphes. Théorie, extension et algorithmes

Fabien de Montgolfier

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

— **S**CIENCES ET **T**ECHNIQUE DU **L**ANGUEDOC —

THÈSE

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de doctorat

SPÉCIALITÉ	: **INFORMATIQUE**
Formation Doctorale	: **Informatique**
École Doctorale	: **Information, Structures, Systèmes**

DÉCOMPOSITION MODULAIRE DES GRAPHES

THÉORIE, EXTENSIONS ET ALGORITHMES

par

Fabien de MONTGOLFIER

Soutenue le 5 décembre 2003 devant le jury composé de :

M. Maxime CROCHEMORE, professeur, IGM, Marne-la-Vallée . . . . . président
M. Jean-Luc FOUQUET, professeur, LIFO, Orléans . . . . . examinateur
M. Cyril GAVOILLE, professeur, LaBRI, Bordeaux . . . . . rapporteur
M. Michel HABIB, professeur, LIRMM, Montpellier . . . . . directeur de thèse
Mr Ross MCCONNELL, assistant professor, Colorado State University, USA . . . . . rapporteur
M. Christophe PAUL, chargé de recherches CNRS, LIRMM, Montpellier . . . . . co-encadrant
M. Jean-Marie VANHERPE, maître de conférences, LIFO, Orléans . . . . . rapporteur

Table des matières

Remerciements

Alors que je m'apprête à porter ce lourd mémoire à l'imprimeur, force m'est de constater qu'il n'aurait pû être sans le concours de nombreuses personnes, auxquelles je tiens à rendre un hommage mérité.

Tout d'abord à celui qui m'a amené à connaître les subtilités de l'algorithmique de graphes mais aussi les personnes qui la font vivre, qui a su m'orienter vers les problèmes les plus intéressants de ce domaine, à son sens comme au mien, qui a dirigé cette thèse en proposant plutôt qu'en imposant, et qui m'a offert sa confiance et son amitié, Michel Habib, merci.

Merci aussi à Christophe Paul, arrivé au cours de ma thèse pour me guider au milieu des arcanes de l'affinage de partitions.

Je remercie les rapporteurs de cette thèse de m'avoir accordé leur temps et leur attention : Cyril Gavoille, à qui mes analyses de complexité n'ont parfois pas résisté, Ross McConnell, qui a su surmonter la barrière du langage (mais l'algorithmique n'est-elle pas la langue adamique de l'informatique ?) et dont les idées brillantes ont rendu ces pages bien des fois meilleures, et Jean-Marie Vanherpe, à l'origine de mes travaux sur les bipartis, qui a apporté une attention méticuleuse à valider l'ensemble.

Merci aussi à Maxime Crochemore qui a porté sur mon travail un regard extérieur, et à Michel Chein dont le regard était plus intérieur mais tout aussi bienveillant.

Je remercie les membres du département Informatique de l'IUT qui ont eu la gentilesse de m'intégrer dans une équipe pédagogique formidable, et particulièrement Henri Bétaille, Jean-Claude Bajard et Olivier Cogis, tuteur pédagogique au vrai sens du terme sur lequel je me suis appuyé à maintes reprises, qui m'ont appris à faire apprendre.

Merci aussi aux chercheurs du LIRMM et d'ailleurs qui, à l'occasion d'une rencontre fortuite ou d'un séminaire, m'ont présenté des travaux passionnants et ont su répondre aux questions que je me posais. Il serait trop long de tous les nommer.

Le personnel du laboratoire, qui a tellement facilité mes travaux, mérite tous mes remerciements. Je pense en particulier à Josette, Marie, Julie et Nicole.

Je voudrais également remercier mes collègues stagiaires, doctorants et docteurs du LIRMM, qui ont créé une ambiance de travail unique (pour le moins) autour de moi : Stéphan, Guillaume, Olivier, Éric et l'autre Olivier pour m'avoir fait découvrir toutes les facettes de la vie lirmienne, Rémi et Arnold pour leur assistance technique, Jean-Marc, amateur de patates à la passion communicative, Jérôme qui sait choisibiliser les bipartis, Sèverine qui a mis quatre heures de plus que moi à faire sa thèse, Pierre qui m'a donné le goût des sandwiches, Emmanuelle qui l'avait déjà, Feryal qui sait parler et se taire, Jean parce que, Sylvain pour son Contact chaleureux, Fabien pour ses promenades en 750, Christophe pour son caractère complètement dynamique, Mohamed pour sa vision communautaire du Web, Toufik pour ses prix imbattables, et Alexis pour ses Pises-aller astronomiques. Sans oublier la base arrière Strasbourgeoise : Alexandre, Jean-Pascal, Loïc, Jean-Marc, Marie-Madeleine et Benoît.

Merci à tous ceux qui m'ont accompagné depuis toujours : mes grand-parents Malapert qui m'ont soutenu depuis mes premiers pas, Matthieu qui m'a montré qu'il fallait rentrer dans les mathématiques pour pouvoir en sortir, Diane qui a choisi les amalgames plutôt que les décompositions, et mes parents qui m'ont transmis leur curiosité et m'ont toujours encouragé à être tel que je suis devenu.

Enfin un merci spécial à Denise, qui m'a montré qu'il fallait sortir des mathématiques pour pouvoir y rentrer, et qui n'a pu que constater que la Recherche est une maîtresse exigeante.

Introduction

divide ut regnes

MACHIAVEL, *1532*

Décompositions et familles partitives

CONSIDÉRONS une *famille*[1] d'objets. Nous allons chercher à *décomposer* cette famille. Mais qu'est-ce que décomposer ?

Certaines décompositions consistent à *hiérarchiser* : on découpe la famille en des classes d'intersection vide, et on recommence récursivement. Une hiérarchie a ainsi la forme d'un arbre. Une des plus fournies des hiérarchies arborescentes connues est celle de Linné, où l'arbre des créatures vivantes n'a pas moins de neuf niveaux, allant des règnes et embranchements jusqu'aux espèces, races et individus.

D'autres décompositions sont constituées par un ensemble de *générateurs*, chaque élément de la famille étant une combinaison linéaire de ces générateurs. Par exemple, les couleurs visibles sont une combinaisons des trois couleurs primaires rouge, jaune et bleu. Nous nous intéressons ici seulement aux combinaisons *booléennes* : un générateur est présent ou absent dans l'élément.

Les familles **partitives** que nous étudions dans le premier chapitre mélangent ces deux décompositions. Ce sont des familles de parties d'un ensemble. Les parties génératrices sont ordonnées en un *arbre* ; un membre de la famille est soit un générateur, soit une combinaison booléenne (union) de générateurs *frères*, descendants d'un même nœud de l'arbre.

Dès lors qu'on a prouvé qu'une famille donnée est partitive, on a alors une puissante panoplie d'outils et de propriétés de cette famille, qui sont de simples corollaires des propriétés des familles partitives. Par exemple, nombreuses peuvent être les parties dans la famille, mais leur représentation en arbre est concise et unique. Ou encore savoir que, pour une classe d'objets don-

1. Ce mémoire considère trois niveaux d'abstraction : les éléments, les ensembles et les familles (ou classes) qui sont des ensembles d'ensembles.

née et pour une propriété donnée, la famille associée est partitive constitue une preuve théorique que des algorithmes efficaces sont possibles, pour des problèmes en relation avec cette propriété. Si de plus on majore l'arité[2] des nœuds de l'arbre de décomposition, alors la classe de complexité du problème change. Prenons l'exemple de la *clique maximale*, un problème de graphes NP-complet, donc dur. Le problème passe au quotient vis-à-vis de la décomposition modulaire. Pour la classe des *cographes* on prouve qu'il est résoluble en temps linéaire par un algorithme simple (max, +) se basant sur l'arbre structurant les modules, appelé le *coarbre* du cographe. Les familles partitives, ainsi que familles bipartitives, faiblement partitives et faiblement bipartitives qui sont étudiées ici sont donc un outil *générique* pour de nombreux problèmes. Une petite liste de telles familles est donnée dans ces pages, mais dans le cadre de ce mémoire ce sera surtout un formalisme unifié pour présenter diverses décompositions de graphes.

Ces concepts ont été développés par Michel Chein, Michel Habib et Marie-Catherine Vilarem [Hab81, CHM81], cherchant une généralisation de la décomposition modulaire , et parallèlement par William Cunningham et Jack Edmonds [CE80] qui généralisent la *split decomposition*. Le chapitre 1 présente un formalisme nouveau unifiant les deux visions.

Une famille partitive possède les mêmes propriétés que les modules d'un graphe, à savoir que si deux parties A et B de la famille se chevauchent, alors l'union $A \cup B$ de ces deux parties, leur intersection $A \cap B$ ainsi que les différence $A \setminus B$ et $B \setminus A$ sont des éléments de la famille. Cette propriété est l'*axiome* définissant une famille partitive, et toutes les autres propriétés en découlent. Le chapitre 1 explique tout cela.

Les graphes et les modules

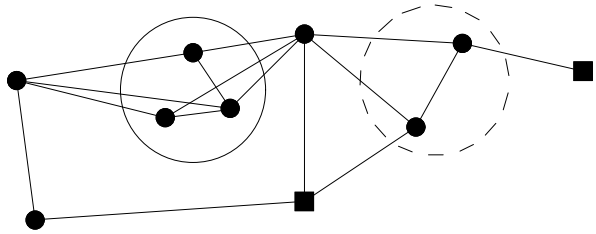
Ce mémoire présente ensuite diverses **décompositions** de graphes. Le mot « décomposition » possède deux sens distincts : une *opération* consistant à diviser un objet en sous-objets, mais aussi la *famille* des produits de la décomposition. C'est essentiellement ce deuxième aspect qui est considéré ici.

Pour bien faire comprendre la décomposition modulaire, présentons parallèlement une des décompositions les plus connue d'un nombre entier n. L'opération de *division euclidienne* de n définit des ensembles de décompositions, les *diviseurs* de n. L'écriture sous forme de *facteurs premiers* triés par ordre croissant est une écriture *canonique*, unique, des diviseurs *minimaux* dans un certains sens (ils n'ont pas, eux-mêmes, de diviseurs). Cette écriture permet, par combinaison de diviseurs, de *générer* tous les diviseurs d'un nombre : on peut dire qu'elle *représente* les diviseurs du nombre.

Dans cette thèse, il est question de graphes, et les ensembles de décomposition considérés seront des modules, ainsi que diverses généralisations de cette notion. De même que le concept de diviseur vient naturellement de la division euclidienne, de même le concept de module vient naturellement de la *distinction*, aussi appelée *séparation*. Un sommet x d'un graphe distingue un ensemble de sommets S de ce même graphe s'il est adjacent (relié) à certains sommets de S,

2. ou valence : nombre de nœuds fils

mais non à tous. x est alors un *séparateur* de S. Un module est un ensemble sans séparateurs. La *décomposition modulaire* d'un graphe est la famille de tous ses modules.



Un module (partie entourée d'un trait plein) dans un graphe. La partie entourée d'un trait pointillé n'est pas un module : les deux sommets carrés la distinguent.

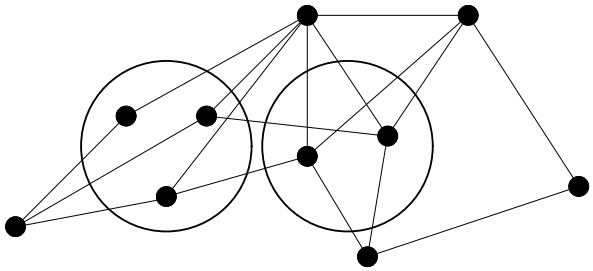
Le deuxième chapitre expliquera comment certains modules, les modules *forts*, jouent le rôle des facteurs premiers : ils permettent de générer tous les modules d'un graphe. L'écriture unique de la décomposition n'est pas un produit de nombre, mais un *arbre* d'inclusion de modules forts, portant en plus certaines marques. Il s'agit d'une famille *partitive*, au sens du paragraphe précédent.

Les graphes sont des objets très utilisés pour la modélisation, dans à-peu-près tous les domaines. Selon l'objet modélisé, le graphe peut avoir une « forme » (c'est-à-dire des propriétés spécifiques) variable, et en particulier de nombreux modules. Les problèmes à résoudre sont innombrables : la théorie des graphes est un outil de base dans beaucoup de disciplines. Or, de nombreux problèmes de graphes – tels la coloration, la clique maximale, l'orientation transitive ou l'appartenance à certaines classes – *passent au quotient* : si l'on sait résoudre le problème sur chacun des modules du graphes, et sur le graphe quotienté par tous ces modules, on sait résoudre le problème sur le graphe de départ. Or, le **temps** mis pour résoudre les problèmes est une fonction croissante de la **taille** des données, et en général la croissance est rapide[3]. Si les modules et le quotient du graphe sont petits, alors s'appuyer sur la décomposition modulaire du graphe permet de gagner un temps de calcul considérable.

Généralisation des modules

Durant cette thèse je me suis intéressé à des *généralisations* de la décomposition modulaire. Sont présentées diverses décompositions de graphes, assez différentes, mais tournant autour de la notion de *2-module*.

3. C'est le cas de l'immense classe de problèmes *NP-Complets*, pour lesquels les algorithmes trouvent, en pratique, la solution en un nombre d'étapes *exponentiel* en la taille de l'instance : la limite concrète de calculabilité est assez vite atteinte...

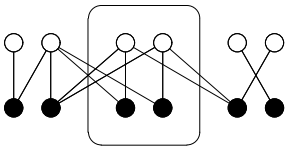


Exemple de 2-module. Il s'agit non plus d'un mais de deux *sous-ensembles de sommets, chacun entourés ici. Un sommet extérieur aux deux ensembles ne peut distinguer ni le premier, ni le second (mais peut distinguer l'union des deux : sinon, le 2-module est un module).*

Le chapitre 3 s'ouvre de façon pessimiste en constatant que les 2-modules violent deux propriétés fondamentales des modules : le fait de former une famille partitive, et le passage au quotient : si M est un module de G et N un module de $G[M]$, alors N est un module de G. Cela n'est plus vrai pour les 2-modules. Le théorème 14 tempère cependant ce résultat négatif, en établissant que la famille des 2-modules d'un graphe sans modules est *presque* partitive. On s'intéresse alors à des sous-familles qui, elles, sont partitives ou bipartitives. C'est le cas par exemple des *sesquimodules*. Un 2-module est *parfait* si son complémentaire est un 2-module. Les 2-modules parfaits appartiennent à quatre familles : modules, coupes, co-coupes et 2-joints. Celle des 2-joints étant nouvelle, elle est étudiée plus en détails. Elle ressemble à la décomposition en coupes [Cun82], avec deux types de graphes totalement décomposables

Il se trouve que coupes, co-coupes et 2-joints généralisent chacun la décomposition modulaire, et de façon *presque* mutuellement exclusive, donc on obtient pour un graphe donné non pas quatre arbres mais un seul, plus riche d'informations que la décomposition modulaire. Cette recherche répond à des questions posées par Jean-Marc Lanlignel dans ses travaux de thèse [Lan01] et s'inscrit dans leur continuité.

L'attention du lecteur est ensuite portée sur les graphes bipartis. L'adaptation des modules au graphes bipartis, les *bimodules*, est un cas particulier de 2-modules.



Un bimodule (sommets entourés). Le graphe est biparti *entre sommets noirs et blancs : les adjacences ne sont autorisées qu'entre deux sommets de couleurs différentes. Aucun sommet noir hors du bimodule ne distingue les sommets blancs du bimodule, et réciproquement.*

Les bimodules, contrairement aux 2-modules, *passent au quotient* : l'approche « arbre de décomposition construit récursivement », inadéquate dans le cas des 2-modules en général, leur est tout à fait adaptée. Nous définissons une sous-famille de bimodules, la **décomposition canonique**, construite récursivement en utilisant quatre opérations différentes de décomposition : `série`, `parallèle`, `K+S` et `premier`. Ils s'agit de cas semblables à ceux de la décomposition modulaire des graphes orientés, ce qui établit un parallèle fort entre ces deux décompositions.

Ces travaux s'inspirent de ceux de Fouquet, Giakoumakis et Vanherpe [GV97b, FGV99, Van99] qui connaissaient les trois premières décompositions. Tous les bimodules ne sont pas canoniques, mais je montre que les bimodules non-canoniques peuvent être lus sur l'arbre de décomposition canonique, qui est donc un arbre codant *toute* la famille des bimodules ! Un algorithme polynômial calculant cette décomposition est proposé, tournant en temps $O(n^4)$.

L'algorithmique. Nécessité de bons algorithmes.

L'*algorithmique* s'intéresse à la résolution de problèmes *répétitifs*, surgissant toujours sous la même forme, et résolubles toujours de la même façon. Une méthode de résolution de problème, que l'on appelle un **algorithme**, doit être **sûre**, **rapide**[4] et **simple**, sans quoi elle est inemployable.

La **sécurité** d'un algorithme est la garantie qu'il retournera un résultat correct dans tous les cas[5]. Elle repose sur des techniques mathématiques de **preuve** (invariants, etc.) bien connues et en général consensuelles. Bien qu'une preuve ne puisse tout dire, les informaticiens s'accordent en général sur le niveau de détails qu'ils jugent suffisant.

La **vitesse** d'exécution d'un algorithme est ce qui le rend utilisable, et permet une comparaison objective entre algorithmes concurrents. Elle est *a priori* très dépendante de la machine sur laquelle on programme et de l'instance du problème considérée. Pour pouvoir en parler de façon abstraite, on se donne un modèle de *machine standard*, sur lequel certaines opérations sont *atomiques* (élémentaires). Le modèle utilisé dans ce mémoire postule en particulier :

- le stockage d'un nombre entier prend une case mémoire ;
- on peut accéder en temps unité au contenu d'une case mémoire ;
- les opérations arithmétiques entre deux nombres entiers se font en temps unité.

Ces postulats sont choquants dans la mesure où la taille des nombres ou de la mémoire n'est pas bornée : stocker le nombre n devrait nécessiter $\log n$ bits de mémoire[6]. Je justifierai ce modèle (dans le cadre de ce mémoire) en remarquant qu'il me suffit, pour un N donné représentant les capacités de stockage de la machine *réelle* à laquelle on pense, que les opérations sur des nombres à $\log N$ bits se fassent en temps unité. Tel est le cas depuis l'aube de l'informatique, et à l'heure actuelle, N est de l'ordre de 2^{40} tandis que les processeurs réalisent des opérations atomiques sur des nombres à 64 bits, ce qui laisse de la marge. Le codage d'un graphe, sous la forme classique des *listes d'adjacence*, occupe donc *par définition* un espace de taille $n + m$, où n est le nombre de sommets et m le nombre d'arêtes.

Alors que la complexité temporelle et spatiale d'un algorithme est quantifiée (par des comparaisons asymptotiques seulement, voire [Knu73]), sa **simplicité** est un critère plus qualitatif. Entre deux algorithmes de même complexité temporelle, qu'est-ce donc qui rend l'un plus

4. Multiplier des nombres à n chiffres demande de l'ordre de 10^n opérations, si l'on compte sur ses doigts. Par la méthode « école primaire » on le fait avec n^2 opération environ. Il existe des algorithmes parallèles en $\log n$.

5. Elle est à distinguer de la *robustesse*, qui est la capacité à distinguer des données erronées.

6. le logarithme considéré dans ce mémoire est toujours le logarithme de la base 2.

simple ? Le *nombre de lignes* d'un programme qui l'implémente est une bonne mesure. Mais elle conduit à des excès, les programmes les plus compacts étant en général illisibles [7]. Le *temps* mis par un programmeur pour transcrire l'algorithme sur une machine donnée est une autre mesure. Mais là encore elle n'est pas bonne car les parties les plus longues à développer et les plus « buggées » des applications sont parfois celles nécessitant le moins d'algorithmique, comme l'affichage d'une fenêtre. Finalement, le critère semble éminemment *subjectif*. Je pense qu'un algorithme simple peut être compris facilement, prouvé limpidement et programmé sans trop de « bugs ». Cet impératif est souvent contradictoire avec celui de vitesse [8].

Algorithmes de décomposition modulaire

La deuxième partie de ce mémoire présente des algorithmes de décomposition modulaires, pour lesquels j'ai essayé de concilier sécurité, vitesse (tous sont linéaire en la taille de la donnée) et simplicité. Des algorithmes linéaires ont déjà été écrits pour la décomposition des graphes non-orientés [CH94, MS99, DGM97], aboutissement d'une longue quête de la complexité optimale. Mais ces algorithmes restent d'une grande complexité sémantique, ce qui rend leur compréhension malaisée, leurs preuves lourdes, et leur implémentation rebutante. La *permutation factorisante* est apparue à l'équipe montpelliérenne [Cap97b, CH97] comme l'outil qui allait permettre la simplification. Il s'agit d'une permutation des sommets du graphe où chaque module fort est un facteur [9]. La stratégie employée est la suivante : dans un premier temps, obtenir une permutation factorisante d'un graphe. Et dans un deuxième temps construire l'arbre de décomposition de ce graphe, en s'appuyant sur la connaissance de cette permutation. Ce concept permet de présenter les premiers algorithmes en temps linéaire de décomposition des graphes orientés, des graphes de permutation et des graphes d'intervalles (pour ces deux derniers cas, le codage considéré est le *modèle d'intersection*, de taille $O(n)$). Cela justifie pleinement l'intérêt que l'on peut porter aux permutations factorisantes.

Les chapitres 6 et 7 correspondent au premier temps, dans le cas des graphes non-orientés, des tournois et des graphes orientés. Pour les deux premières classes, des techniques d'*extension d'ordre* sont utilisées. Partant d'un ordre vide, l'algorithme ajoute des comparaisons jusqu'à aboutir à un ordre total. Ces algorithmes ont été écrit en collaboration avec Michel Habib et Christophe Paul, dans la continuité d'un algorithme en $O(m \log n)$ de leur part [HPV99]. Et pour les graphes orientés, l'algorithme est une réduction à trois autres algorithmes présentés dans ce mémoire, et a été écrit en collaboration avec Ross McConnell.

7. Certains programmeurs considèrent comme un art et une gageure de produire du code d'une concision extrême. Ainsi, il existe un jeu de Tétris$^{\mathrm{tm}}$ graphique qui tient sur le kilo-octet du secteur de démarrage d'une disquette ! On ne peut prétendre qu'un tel programme soit simple...

8. Exemple : on a prouvé que l'on peut multiplier deux matrices de taille $n \times n$ en $O(n^{2,38})$ opérations, mais grâce à des techniques d'une sophistication poussée – alors que l'algorithme « naïf » de trois lignes utilise n^3 opérations.

9. aussi appelé *intervalle* : c'est une suite d'éléments consécutifs selon la permutation.

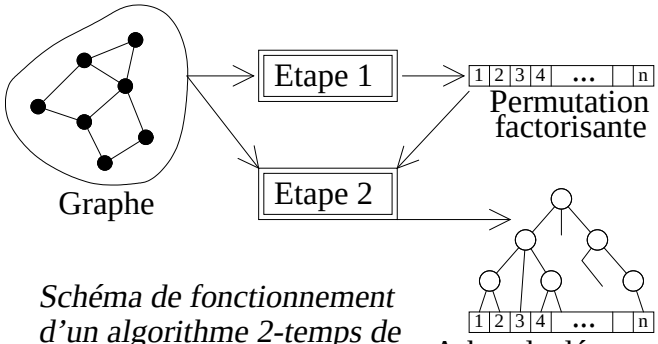


Schéma de fonctionnement d'un algorithme 2-temps de décomposition modulaire

Le deuxième temps est traité par le chapitre 8. Introduisant la notion de *fractures*, il surimprime des *parenthèses* à la permutation factorisante. Cela permet de construire un arbre, qui est transformé en l'arbre de décomposition modulaire. Par ailleurs, dans deux classes de graphes particuliers (graphes d'intervalles et graphes de permutation) si on possède un *modèle d'intersection* du graphe, non seulement on connaît une permutation factorisante, mais en plus on peut avoir les séparateurs d'une paire de sommets. Cela permet d'écrire un algorithme de décomposition modulaire en temps $O(n)$ pour chaque classe, linéaire en la taille du modèle d'intersection et sous-linéaire en la taille du graphe !

L'algorithme pour le cas des graphes orientés utilise un algorithme d'intersection de familles partitives. Cet outil possède sans doute bien d'autres applications.

Principaux algorithmes de cette thèse

Intersection de deux familles partitives	linéaire	§5.5
Calcul d'un k-joint dans un graphe	$O(k^2n^2m^k)$	§3.6.5
Décomposition en 2-joints d'un graphe	$O(n^3m^2)$	§3.6.5
Décomposition canonique des graphes bipartis	$O(n^4)$	§4.9
Calcul d'une P.F. d'un graphe non-orienté (par extension d'ordres)	$O(n+m)$	chap. 6
idem, version parallèle, $O(n)$ processeurs	$O(n)$	chap. 6
Calcul d'une P.F. d'un tournoi (par extension d'ordres)	$O(m)$	§7.1
Calcul d'une P.F. d'un graphe orienté (par réduction)	$O(n+m)$	§7.2
D.M. d'un graphe (orienté ou non), connaissant une P.F.	$O(n+m)$	chap. 8
idem, version parallèle, $O(n)$ processeurs	$O(n)$	chap. 8
D.M. d'un graphe d'intervalles, modèle d'intervalles connu	$O(n)$	§8.8
D.M. d'un graphe de permutation, modèle de permutation connu	$O(n)$	§8.9

(*) P.F. abrège Permutation Factorisante, et D.M. abrège Décomposition Modulaire

Publications

Voici la liste à ce jour des publications de mes travaux. [BBM03a, BBM03b] correspondent à des travaux sur le graphe du Web en collaborations avec Toufik Bennouas et Mohamed Bouklit, et ne rentrent pas dans le cadre de ce mémoire. Les travaux sur les décomposition 2-modulaire et bimodulaires, trop récents, n'ont pas encore fait l'objet de publication. Tous ces documents sont téléchargeables sur
`http://www.lirmm.fr/~montgolfier`

[Mon00] F. DE MONTGOLFIER. Trouver l'arbre de décomposition modulaire d'un graphe à partir d'une permutation factorisante. Mémoire du D.E.A. *Algorithmique*, E.N.S. de Cachan - LIRMM, 2000.

[CHM02] C. CAPELLE, M. HABIB, et F. DE MONTGOLFIER. Graph decomposition and factorizing permutations. *Discrete Mathematics and Theoretical Computer Sciences*, vol. 5, n^o 1, 2002. `http://dmtcs.loria.fr/volumes/abstracts/dm050104.abs.htm`.

[Mon01] F. DE MONTGOLFIER. Modular decomposition of tournaments, directed graphs and 2-structures : linear algorithms. Rapport Tech. RR01379, LIRMM, 2001.

[MM03] R. M. MCCONNELL et F. d. MONTGOLFIER. Linear-time modular decomposition of directed graphs. Accepté pour publication dans *Discrete Applied Mathematics*.

[BBM03a] T. BENNOUAS, M. BOUKLIT, et F. d. MONTGOLFIER. Un modèle gravitationnel du Web. In *Actes des 1ères Journées Francophones de la Toile*, 2003. http://www.antsearch.univ-tours.fr/jft2003/.

[BBM03b] T. BENNOUAS, M. BOUKLIT, et F. d. MONTGOLFIER. Un modèle gravitationnel du Web. In *Actes de* ALGOTEL03 *5ème Rencontres Francophones sur les aspects Algorithmiques des Télécommunications*, 2003. http://dept-info.labri.u-bordeaux.fr/algotel03/.

[Mon03a] F. DE MONTGOLFIER. Décompositions de graphes. Rapport tech., 11èmes Journées des Doctorants de l'école doctorale ISS - Montpellier.

[HMP03] M. HABIB, F. DE MONTGOLFIER, et C. PAUL. A simple linear-time modular decomposition algorithm for graphs, using order extending. Rapport de recherche duLIRMM n° RR03007, 2003. Soumis à SODA03 - Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms.

[Mon03b] F. DE MONTGOLFIER. A twomodular graph decomposition theory. Soumis à STACS 2004 (symposium on theoretical aspects of computer sciences).

Première partie

Autour de la décomposition modulaire

Chapitre 1

Ensembles, familles de parties et familles de bipartitions

If I have seen farther than others, it is because I was standing on the shoulders of giants.

Isaac NEWTON

Mathematicians stand on each other's shoulders.

GAUSS

In the sciences, we are now uniquely priviledged to sit side by side with the giants on whose shoulders we stand.

Gerald HOLTON

If I have not seen as far as others, it is because giants were standing on my shoulders.

Hal ABELSON

CE CHAPITRE présente quatre concepts ensemblistes : les familles partitives, faiblement partitives, bipartitives et faiblement bipartitives. Ils seront utilisés dans le cadre des décompositions de graphes, afin de présenter un éventail de décompositions variées sous un formalisme unifié.

Les sections 1.3 et 1.4 présentent des résultats de Michel Chein, Michel Habib et Marie-Catherine Maurer (Mme Villarem) sur les familles **partitives** [CHM81, Hab81]. Leurs travaux portent sur des familles de parties d'un ensemble donné possédant de bonnes propriétés de fermeture pour l'union, l'intersection et la différence. Cette approche est applicable à des décompositions de graphes où les ensembles de décomposition sont des parties, telles que la décomposition modulaire. Elle n'est en revanche pas adéquate pour les décompositions de graphes où les ensembles de décomposition sont des bipartitions, telles que la décomposition en 2-joints,

dont il sera question dans ce mémoire. C'est pourquoi est présenté §1.6 un autre outil, les familles **bipartitives**. Des résultats équivalents se trouvent déjà dans les travaux de Cunningham et Edmonds [Cun73, CE80], mais présentés de façon très différente [1].

Il se trouve que nombreux objets (au sens large) sont des familles partitives ou bipartitives. Un rapide catalogue est dressé §1.8. Celles qui sont en lien avec la *décomposition modulaire des graphes* sont présentées dans les trois chapitres suivants. L'approche que j'ai suivie dans ce mémoire part du constat que toutes ces décompositions, étant à base de familles partitives et bipartitives, ont des propriétés proches, qu'il convient de factoriser. C'est l'objet du présent chapitre. Il n'y aura plus, pour une décomposition donnée, que de prouver un théorème disant que ses ensembles de décomposition forment une famille partitive ou bipartitive, pour en déduire aussitôt en corollaire beaucoup de propriétés de cette décomposition. Mais présentons tout d'abord les outils les plus basiques.

1.1 Ensembles et partitions

Ce mémoire est destiné au lecteur familier de la théorie des graphes ; cependant étant donné que certains termes et notations utilisées sont personnels, et afin de conserver la possibilité d'être lu par un néophyte, la terminologie utilisée dans ce mémoire ainsi que les concepts fondamentaux dont il sera question seront définis au moment opportun. Une **table des notations** située page 240, et un **index** situé page 242, permettront au lecteur de retrouver rapidement la signification de tel signe ou mot étrange.

Peu se risquent à définir les **ensembles**, mais tous connaissent les opérations permettant de les manipuler : union $\cup$, intersection $\cap$, différence $\setminus$, complément $\overline{\ }$, différence symétrique [2] Δ, union disjointe [3] $\uplus$. Nous utilisons la relation (opérateur booléen) de **chevauchement** $\between$: $A \between B$ si $A \setminus B$, $A \cap B$ et $B \setminus A$ sont non vides.

Les notations de variables de ce mémoire utilisent, autant que possible, les minuscules pour des éléments (x), les majuscules pour des ensembles (E) et les majuscules calligraphiques pour des ensembles d'ensembles ou des objets d'ordre supérieur ($\mathcal{F}$). Sauf exceptions il sera question d'ensembles *finis*. Cette restriction est légitime d'un point de vue algorithmique, si l'on se préoccupe de la *taille* des données. Les structures que nous utilisons sont *dénombrables*, au sens qu'elles possèdent toutes une taille entière et que pour une taille donnée il existe un nombre fini de structures différentes possibles. Tout au long de ce mémoire V désigne un ensemble fini à n éléments, et la définition de V et n sera souvent omise.

Une **partition** d'un ensemble fini E est un ensemble $\mathcal{P} = \{C_1, \dots C_k\}$ de k sous-ensembles de E, tel que chaque **partie** (ou *classe*) C_i est non-vide, que l'union des partie est égale à

1. Finalement, les résultats de Chein, Habib et Maurer [CHM81] rejoignent ceux de Cunningham et Edmonds [Cun73, CE80] ainsi que ceux de Möhring et Radermacher [MR84], publiés indépendemment et dans des formalismes différents.

2. $A \Delta B = (A \setminus B) \cup (B \setminus A)$

3. $A \uplus B = A \cup B$, mais cette notation insiste sur le fait que $A \cap B = \emptyset$

E et que deux parties différentes sont d'intersection vide. La partition est **au sens large** si on s'autorise des parties vides. S'il existe $e \in E$ tel que $C_i = \{e\}$, alors C_i est un **singleton**[4].

Une propriété du chevauchement qui nous sera utile est :

Proposition 1
Si $A \circledcirc B$ et $C \circledcirc (A \cup B)$ alors $A \circledcirc C$ ou $B \circledcirc C$.
Si $A \circledcirc B$ et $C \circledcirc (A \cap B)$ alors $A \circledcirc C$ ou $B \circledcirc C$.
Si $A \circledcirc B$ et $C \circledcirc (A \smallsetminus B)$ alors $A \circledcirc C$ ou $B \circledcirc C$.

1.2 Familles arborescentes

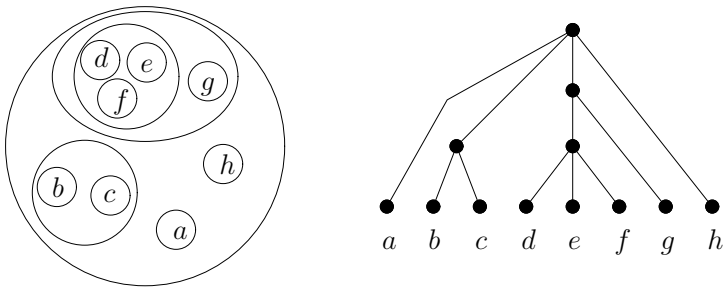
Présentons une première classe de familles de parties, la plus simple (à mon sens) à cause de la taille polynômiale et de la structure d'*arbre* de ces familles.

Définition 1
*Soit $\mathcal{F} \subset 2^V$ un ensemble de parties de V. $\mathcal{F}$ est une **famille arborescente** de V si*

1. *$V \in \mathcal{F}$, $\emptyset \notin \mathcal{F}$ et pour tout $v \in V$, $\{v\} \in \mathcal{F}$*
2. *$\forall E, F \in \mathcal{F}$ on a $E \subset F$ ou $F \subset E$ ou $E \cap F = \emptyset$*

L'inclusion définit un ordre partiel sur $\mathcal{F}$. Soient E, F et G trois parties différentes de $\mathcal{F}$ telles que $E \subset F$ et $E \subset G$. On ne peut avoir $F \cap G = \emptyset$, donc F et G sont comparables pour l'inclusion. On en déduit que chaque partie sauf V possède un unique élément minimal (pour l'inclusion) qui la contienne. La réduction transitive[5] de l'ordre d'inclusion sur $\mathcal{F}$ est donc un *arbre* dont la racine est V, et dont les feuilles sont les singletons. Les nœuds de cet arbres ne pouvant avoir un seul fils, on en déduit

Proposition 2
Une famille arborescente possède entre $n+1$ et $2n-1$ parties.



Exemple de famille arborescente (à gauche) et de son arbre (à droite) sur l'ensemble $\{a...h\}$.

4. le terme me semble un peu franglais... mais son usage répandu parmi les joueurs de cartes m'autorise à l'utiliser dans ce mémoire.

5. Le plus petit graphe orienté dont la fermeture transitive soit égale à cet ordre. Elle est unique.

L'arbre d'inclusion d'une famille arborescente sera noté $\mathcal{T}_\mathcal{F} = (\mathcal{F}, R)$ où R est la réduction transitive de l'ordre d'inclusion de $\mathcal{F}$. Un arc de E vers F signifie que F est la plus petite partie qui contienne E. On dit alors que F est le *père* de E, qui est son *fils*. Les fils d'un nœud N sont notés $F_1^N \dots F_k^N$. Deux parties E et F d'une famille arborescentes sont **sœurs** si leur père dans $\mathcal{T}_\mathcal{F}$ (la plus petite partie qui les contiennent tous deux) est identique[6]. La proposition suivante est un point important dans les preuves sur les familles partitives :

Proposition 3
Soit $\mathcal{F}$ une famille arborescente de V et $E \subset V$. Si E ne chevauche aucun élément de $\mathcal{F}$ alors E est l'union de parties sœurs de $\mathcal{F}$.

Si le lecteur (à qui est laissé le soin de la démonstration) veut la vérifier, il lui suffit de considérer la plus petite partie de $\mathcal{F}$ contenant E : soit elle est égale à E, soit ses fils sont inclus ou exclus de E.

1.3 Familles partitives

La définition et le théorème de représentation des familles partitives présentés ici sont une reformulation des résultats de [CHM81], originellement présentés avec le formalisme des hypergraphes et des partitions. À l'origine, ce travail est une généralisation des propriétés de l'arbre de décomposition modulaire d'un graphe non-orienté. Dans ce mémoire, nous le verrons appliqué dans le cadre de nombreuses décompositions de graphes, qui sont toutes elles-mêmes plus ou moins des généralisations de la décomposition modulaire. Mais c'est un outil générique que nous avons entre les mains. Étant donnée une décomposition particulière, il suffira de prouver que la famille des *ensembles de décomposition* est partitive, pour obtenir par corollaire du théorème 1 de nombreuses propriétés intéressantes de ladite décomposition.

Définition 2
*Soit $\mathcal{P} \subset 2^V$ une famille de parties de V. $\mathcal{P}$ est **partitive** si*

1. *$V \in \mathcal{P}$, $\emptyset \notin \mathcal{P}$ et pour tout $v \in V$, $\{v\} \in \mathcal{P}$*
2. *Pour toute paire de parties $A, B \in \mathcal{P}$ telles que $A \oo B$ on a :*
   - *(a) $A \cap B \in \mathcal{P}$*
   - *(b) $A \setminus B \in \mathcal{P}$ et $B \setminus A \in \mathcal{P}$*
   - *(c) $A \cup B \in \mathcal{P}$*
   - *(d) $A \Delta B \in \mathcal{P}$*

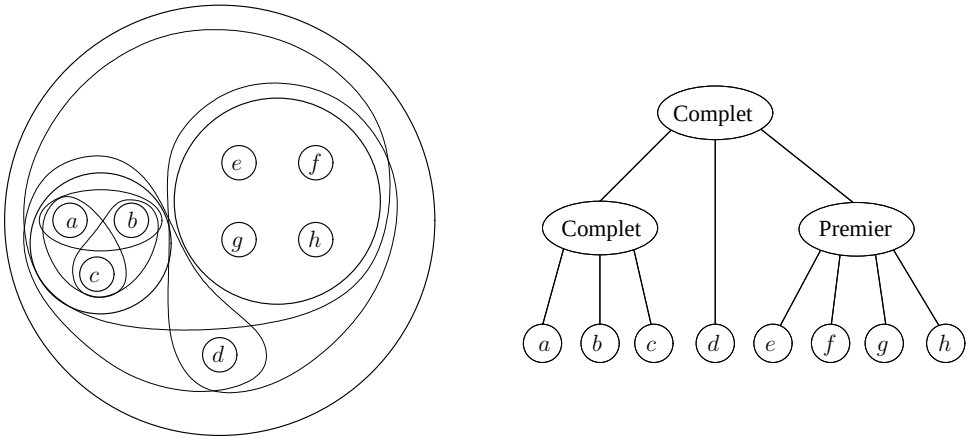
Une partie $F \in \mathcal{P}$ est **forte** si elle ne chevauche aucune autre partie. Les parties fortes de $\mathcal{P}$ sont une famille arborescente. Nous noterons $\mathcal{T}_\mathcal{P}$ l'arbre d'inclusion des parties fortes de $\mathcal{P}$. Les nœuds de cette arbre vont porter une *marque*, c'est-à-dire une étiquette[7].

6. On parlera aussi de sous-ensembles *frères*.
7. Un *marquage* est une application de l'ensemble des nœuds dans l'ensemble des marques.

Théorème 1 [CHM81]
Les nœuds de $\mathcal{T}_\mathcal{P}$ peuvent être marqués `complet` *ou* `premier` *de telle sorte que, pour tout nœud* `complet` N *à k fils et tout $I \subset [\![1, \dots k]\!]$ non vide on ait*

$$\bigcup_{i \in I} F_i^N \in \mathcal{P}$$

Réciproquement, toute partie $A \in \mathcal{P}$ est soit forte, soit l'union d'un ensemble de fils d'un nœud `complet`.



Exemple de famille partitive et de son arbre. L'ensemble de base est $\{a, ...h\}$, et les membres de la famille sont les singletons ainsi que $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, $\{a, b, c\}$, $\{a, b, c, d\}$, $\{a, b, c, e, f, g, h\}$, $\{d, e, f, g, h\}$, $\{e, f, g, h\}$ et $\{a, ...h\}$. Les parties fortes sont entourées en gras et se lisent directement sur l'arbre. Les parties faibles sont union de sœurs filles d'un même nœud `complet`.

Ce théorème peut se prouver en considérant les parties *faibles* (non-fortes) : la proposition 3 indique qu'elles sont union de sœurs. Si un nœud (partie forte) contient[8] une partie faible, on le marque `complet` et sinon `premier`. En considérant le *graphe de chevauchement* des parties faibles, on prouve que les composantes connexes non-triviales de ce graphe correspondent aux nœuds complets : l'union des membres d'une composante donne un nœud, et les intersections non-vides de membres d'une composante donnent les fils de ce nœud. Grâce aux axiomes on peut alors finir la démonstration.

Ce théorème exprime qu'il est possible de représenter une famille partitive sous forme d'un arbre, qui donne toute sa structure. D'après la proposition 2 page 25, cet arbre possède $\Theta(n)$ nœuds. Ce résultat est surprenant : en effet la famille de *tous* les sous-ensembles de V (sauf $\emptyset$) est partitive, mais possède $\Omega(2^n)$ éléments ! Les familles partitives, de taille potentiellement exponentielle, possèdent donc un *codage* compact, de taille polynômiale.

On remarque que tout élément $x \in V$ est présent dans toutes les parties fortes situées sur le chemin reliant la racine de l'arbre à la plus petite partie forte qui le contienne, et seulement dans

8. c'est-à-dire « est la plus petite partie forte contenant »

ces parties fortes-là.

Les seules ambiguïtés dans le marquage proviennent des nœuds à deux fils, qui peuvent indifféremment être marqués `premier` ou `complet`. Lever cette ambiguïté permet d'obtenir un arbre unique. Comme par ailleurs l'arbre d'inclusion des parties fortes est unique, on a :

Proposition 4
Pour peu que les nœuds à deux fils soient tous marqués `premier`*, l'arbre $\mathcal{T}_{\mathcal{P}}$ et son marquage, conformes au théorème 1 page 26, sont uniques.*

Au contraire de notre définition, [CHM81] impose l'axiome $\emptyset \in \mathcal{P}$. Cela simplifie les axiomes (les points (a) et (b) ne demandent plus que les parties se chevauchent) mais est gênant pour les remarques sur l'unicité : il n'y a même pas unicité de l'arbre d'inclusion ! En effet, tout nœud peut avoir la partie vide comme fils. Si au contraire on interdit la partie vide, l'arbre d'inclusion est unique.

1.4 Familles faiblement partitives

Dans la thèse d'État de Michel Habib [Hab81], on trouve la généralisation suivante des familles partitives. De même que les familles partitives ont été inventées pour abstraire la structure des modules d'un graphe non-orienté, de même les familles faiblement partitives correspondent aux modules des graphes orientés. Ce mémoire présentera d'autres familles, également faiblement partitives.

Définition 3
Soit $\mathcal{P} \subset 2^V$ une famille de parties de V. $\mathcal{P}$ est ***faiblement partitive*** *si*

1. *$V \in \mathcal{P}$, $\emptyset \notin \mathcal{P}$ et pour tout $v \in V$, $\{v\} \in \mathcal{P}$*
2. *Pour toutes parties $A, B \in \mathcal{P}$ telles que $A \oo B$ on a :*
   - *(a) $A \cap B \in \mathcal{P}$*
   - *(b) $A \setminus B \in \mathcal{P}$ et $B \setminus A \in \mathcal{P}$*
   - *(c) $A \cup B \in \mathcal{P}$*

Par-rapport aux familles partitives, il manque l'axiome sur la différence symétrique. Cela conduit à un théorème plus général (en gardant les mêmes notations) :

Théorème 2 [Hab81]
Les nœuds de $\mathcal{T}_{\mathcal{P}}$ peuvent être marqués `complet`*,* `linéaire` *ou* `premier`*, et les fils d'un nœud* `linéaire` *peuvent être ordonnés, de telle sorte que, pour tout nœud* `complet` *N à k fils et tout $I \subset [\![1, \dots k]\!]$ non vide on ait*

$$\bigcup_{i \in I} F_i^N \in \mathcal{P}$$

et pour tout nœud `linéaire` N *à* k *fils et tous* $1 \leqslant a \leqslant b \leqslant k$ *on ait*

$$\bigcup_{a \leqslant i \leqslant b} F_i^N \in \mathcal{P}$$

Réciproquement, toute partie $A \in \mathcal{P}$ *est soit forte, soit l'union d'un ensemble de fils d'un nœud* `complet`*, soit l'union d'un ensemble de fils d'un nœud* `linéaire` *qui sont consécutifs dans l'ordre des fils.*

Cet arbre possède les mêmes propriétés de représentation que celui d'une famille partitive, sauf que maintenant l'ordre dans lequel les fils apparaissent est important (pour les nœuds `linéaire`). Un tel arbre est appelé **PQ-tree**[9] [BL76] : les fils d'un nœud P peuvent être arbitrairement ordonnés, tandis que les fils d'un nœud Q doivent être ordonnés dans un certain ordre. Nous y reviendrons dans le chapitre 5 à propos de permutations factorisantes. La même ambiguïté de marquage des nœuds à deux fils existe que pour l'arbre d'une famille partitive : l'arbre est unique pour peu que tous les nœuds à deux fils soient marqués `premier`.

Nous verrons dans la section 2.3 page 53 une autre façon de lever l'ambiguïté du marquage, dans le cas de la décomposition modulaire : selon le graphe quotient, les nœuds à deux fils seront marqués `série`, `parallèle` ou `ordre`.

Les familles faiblement partitives sont une généralisation stricte des familles partitives : en effet, la famille suivante de parties de $\{1, 2, 3\}$ est faiblement partitive, mais non partitive.

$$\mathcal{P} = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$$

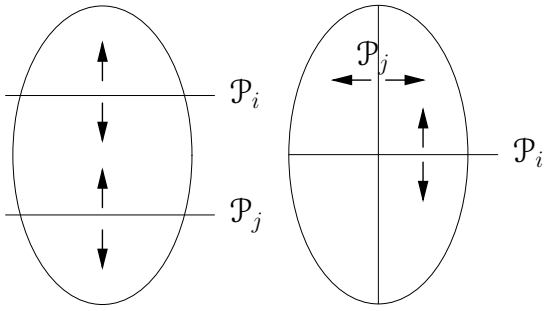
1.5 Familles arborées

Cette section et la suivante présentent des familles de **bipartitions** d'un ensemble : les familles *arborées* et *bipartitives*. Nous allons d'abord nous intéresser à une classe simple de famille de bipartitions, présentée ici.

$\{P^0, P^1\}$ est une **bipartition**[10] de V si $P^0 \cap P^1 = \emptyset$ et $P^0 \cup P^1 = V$, ce que l'on note $P^0 \uplus P^1 = V$. Une bipartition **élémentaire** est de la forme $\{\{v\}, V \setminus \{v\}\}$. Soient P_i et P_j deux bipartitions de V.

- Si $P_i^0 = P_j^0$ ou $P_i^0 = P_j^1$ il y a **égalité** entre P_i et P_j : elles représentent la même bipartition.
- Si $P_i^0 \cap P_j^0 = \emptyset$ ou $P_i^0 \cap P_j^1 = \emptyset$ ou $P_i^0 \cap P_j^0 = \emptyset$ ou $P_i^1 \cap P_j^1 = \emptyset$ alors il y a **non-chevauchement** entre P_i et P_j.
- Sinon il y a **chevauchement**. Noter que en cas de chevauchement entre P_i et P_j on a $P_i^\alpha \Cap P_j^\beta$ pour tous $\alpha, \beta \in \{0, 1\}$.

9. Voir §5.2.2. Traduire ce terme en « PQ-arbre » me semble pire que le laisser en anglais...
10. placer 0 et 1 en exposant permet de pouvoir indicier les bipartitions.



non-chevauchement *chevauchement*

Définition 4
Soit $\mathcal{B} = \{P_i\}_{i=1...m} = \{\{P_i^0, P_i^1\}\}_{i=1...m}$ une famille de bipartitions de V. $\mathcal{B}$ est une famille ***arborée*** *si*

- *deux bipartitions de cette famille ne se chevauchent jamais, et*
- *elle contient toutes les bipartitions élémentaires.*

Il est assez immédiat qu'une famille *arborescente* (*cf.* §1.2) possède une structure d'arbre enraciné (appelé *arborescence* par certains). Nous allons montrer qu'une famille *arborée* correspond elle aussi à un arbre, *non-enraciné*[11].

Prenons n'importe quel arbre non-enraciné dont les feuilles sont les éléments de V. Toute arête a de cet arbre définit une bipartition de V : supprimer a coupe l'arbre en deux composantes connexes. $\tilde{P}_a^0$ est formée des feuilles du premier sous-arbre, et $\tilde{P}_a^1$ des feuilles du second sous-arbre. On note $\tilde{P}_a = \{\tilde{P}_a^0, \tilde{P}_a^1\}$ la bipartition induite par a. Les arêtes d'un arbre $\mathcal{T}$ définissent donc une famille $\tilde{\mathcal{F}}_{\mathcal{T}}$ de bipartitions. On vérifie facilement que, étant données deux arêtes a et b d'un arbre $\mathcal{T}$, et les composantes C_a^0 et C_a^1 de $\mathcal{T} - a$ (resp. C_b^0 et C_b^1 de $\mathcal{T} - b$), il existe i et j tels que $C_a^i \cap C_b^j = \emptyset$. $\tilde{P}_a$ et $\tilde{P}_b$ ne se chevauchent pas : la famille $\tilde{\mathcal{F}}_{\mathcal{T}}$ est arborée, et contient toutes les bipartitions élémentaires. Nous nous intéressons à la réciproque : étant donnée une famille $\mathcal{F}$ de bipartitions fortes , nous allons construire un arbre $\mathcal{T}$ tel que

$$\tilde{\mathcal{F}}_{\mathcal{T}} = \mathcal{F} \tag{1.1}$$

Pour cela, on numérote de 1 à m les bipartitions non-élémentaires de $\mathcal{F}$. Nous construisons une suite $\mathcal{T}_0, \ldots \mathcal{T}_m$ d'arbres, vérifiant l'invariant suivant :

Invariant 1
Pour la famille arborée $\mathcal{F}_i$ contenant les bipartitions élémentaires et $P_1, \ldots P_i$, on a

$$\tilde{\mathcal{F}}_{\mathcal{T}_i} = \mathcal{F}_i$$

11. Les bio-informaticiens qui travaillent sur la *reconstruction phylogénétique* extraient de telles familles de l'ensemble de toutes les bipartitions de V, en essayant d'optimiser certains critères (voir [BB01]) et appellent *phylogénie* l'arbre obtenu, car ses feuilles sont les espèces animales et végétales. Beaucoup de problèmes de *clustering* tournent autour de telles familles...

$\mathcal{T}_0$ est l'arbre en étoile, possédant un nœud central et n feuilles adjacentes à ce nœud, une par éléments de V. Cet arbre représente[12] la famille bipartitive triviale contenant uniquement les bipartitions élémentaires.

Étant donné un nœud N de $\mathcal{T}_i$, les composantes connexes de $\mathcal{T}_i - N$ sont notées $\mathcal{C}(i, N) = C_1, \dots C_k$. Considérons la bipartition $P_{i+1} = \{P^0_{i+1}, P^1_{i+1}\}$.

Lemme 1 *Il existe un et un seul nœud N de $\mathcal{T}_i$ tel qu'aucun élément de $\mathcal{C}(i, N)$ ne chevauche P^0_{i+1} ni P^1_{i+1}.*

Démonstration: La famille $\mathcal{F}_{i+1} = \mathcal{F}_i \cup \{P_{i+1}\}$ étant arborée, aucune bipartition P_j de $\mathcal{F}_i$ ne chevauche P_{i+1} : il existe α et β tels que $P^\alpha_j \cap P^\beta_{i+1} = \emptyset$. L'invariant donne $\tilde{\mathcal{F}}_{\mathcal{T}_i} = \mathcal{F}_i$. On peut donc *orienter* les arêtes de $\mathcal{T}_i$ d'après P_{i+1} : la flèche va du côté α au côté $1-\alpha$, pour la bipartition P_j correspondant à l'arête a.

Le point crucial est la suivant : pour a orientée de C^α_a vers $C^{1-\alpha}_a$ (les deux composantes de $\mathcal{T}_i - a$), toutes les arêtes de C^α_a sont orientées vers a. Soit b une arête de C^α_a et $\{P^0_b, P^1_b\}$ la bipartition de $\mathcal{F}$ correspondante (d'après l'invariant). Il existe γ tel que $P^\gamma_b \subset P^\alpha_j$, donc $P^\gamma_b \cap P^\beta_i = \emptyset$, ce qui amène le résultat.

Les arbres vérifiant cette propriété d'orientation des arêtes sont les arbres enracinées : il n'existe aucun nœud d'où partent deux arêtes. La racine (unique nœud dont ne part aucune arête) est le nœud N recherché.

□

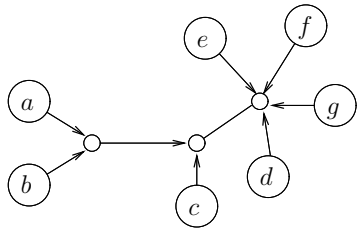
Ce lemme indique comment construire $\mathcal{T}_{i+1}$ depuis $\mathcal{T}_i$: le nœud N est coupé par une arête, représentant P_{i+1}. Les parties de $\mathcal{C}(i, N)$ incluses dans P^0_i sont données à l'une des extrémités de cette arêtes, et les autres à l'autre extrémité. Il est clair que cela vérifie l'invariant. Finalement $\mathcal{T}_m = \mathcal{T}$.

On constatera que $\mathcal{T}$ possède n feuilles, $1 \leqslant m < n$ nœuds internes et n'a pas de nœud interne de degré 2 (les bipartitions correspondant aux deux arêtes seraient égales).

Aspects algorithmiques

On déduit de cette preuve un *algorithme* en $O(nm)$ pour construire $\mathcal{T}$: chaque arête peut être ajoutée en $O(n)$ à l'arbre. À l'étape i, on *marque* tous les éléments de P^0_i puis on propage la marque de telle sorte que pour tout nœud dont tous les fils sont marqués, ce nœud soit aussi marqué et ses fils démarqués (voir algorithme 3 page 149). On trouve ainsi N qui est coupé entre fils marqués et non-marqués. Notons que cela est sous-linéaire en la taille de la famille (on ne considère pas les bipartitions élémentaires), où la *taille* d'une famille $\mathcal{F}$ est $\sum_{E\in\mathcal{F}} |E|$ (voir chapitre 5).

12. « *représente* » signifie « vérifie l'équation 1.1 »



Famille arborée. Ses bipartitions sont $a|bcdefg$ (représentation concise de $\{\{a\},\{b,c,d,e,f,g\}\}$), $b|acdefg$, $c|abdefg$, $abc|defg$, $d|abcefg$, $e|abcdfg$, $f|abcdeg$ et $g|abcdef$. L'orientation des arc est celle fournie par l'algorithme quand on essaye d'insérer la bipartition $abcde|fg$, ce qui amenera à couper en deux le nœud racine de cette orientation.

1.6 Familles bipartitives et faiblement bipartitives

Nous allons maintenant nous intéresser à des familles plus complexes de bipartitions. De même que familles partitives et faiblement partitives généralisent les modules des graphes non-orientés et orientés, de même familles **bipartitives** et **faiblement bipartitives** généralisent les coupes des graphes non-orientés et orientés. Les familles bipartitives sont un outil général pour présenter trois décompositions dont il sera question dans ce mémoire : la décomposition en coupes, la décomposition en co-coupes, et la décomposition en 2-joints.

Cunningham et Edmonds [Cun73, CE80, Cun82] définissent le concept de *cadre de décomposition* (*decomposition frame*), où l'on considère une opération divisant un membre d'une classe (comme un graphe) en deux membres partageant un même *marqueur*. C'est en fait bien d'une opération de bipartition qu'il s'agit. Le formalisme utilisé ici supprime les marqueurs et l'aspect récursif, mais on retrouve des résultats déjà présent dans [Cun73, CE80]. Une famille bipartitive est un cadre de décomposition possédant les propriétés d'intersection et de transitivité, et une famille faiblement bipartitive possède juste la propriété d'intersection. Le formalisme présenté ici est assez différent, c'est pourquoi j'ai jugé utile de présenter des preuves des théorèmes, qui sont nouvelles.

Un **PC-tree**[13] [HM03] est la structure représentant les familles faiblement bipartitive ; il a été introduit par Hsu et McConnell pour étudier la propriété des 1 circulaires[14].

Définition 5
*Soit $\mathcal{B} = \{P_i\}_{i=1\ldots m} = \{\{P_i^0, P_i^1\}\}_{i=1\ldots m}$ une famille de bipartitions de V. $\mathcal{B}$ est une **famille faiblement bipartitive** si*

- *il n'y a égalité entre aucune de ses bipartitions*
- *elle contient toutes les bipartitions élémentaires*

13. Voir §5.2.4. Traduire ce terme en « PC-arbre » me semble autant affreux que « PQ-arbre ».

14. Il s'agit de la propriété que possède une matrice booléenne quand ses colonnes peuvent être permutées de sorte, sur chaque ligne, soit les 1 sont consécutifs, soit les 0 sont consécutifs.

– *pour tout couple de bipartitions P_i, P_j* ***se chevauchant****, on a*

1. $\{P_i^0 \cap P_j^0, P_i^1 \cup P_j^1\} \in \mathcal{B}$
2. $\{P_i^0 \cap P_j^1, P_i^1 \cup P_j^0\} \in \mathcal{B}$
3. $\{P_i^1 \cap P_j^0, P_i^0 \cup P_j^1\} \in \mathcal{B}$
4. $\{P_i^1 \cap P_j^1, P_i^0 \cup P_j^0\} \in \mathcal{B}$

Une famille faiblement bipartitive est ***bipartitive*** *si, pour tout couple de bipartitions se chevauchant, on a de plus*

5. $\{P_i^0 \Delta P_j^0, P_i^0 \Delta P_j^1\} \in \mathcal{B}$

Une bipartition est **forte** si elle ne chevauche aucune autre bipartition de la famille. $\mathcal{F} \subset \mathcal{B}$ est la famille des bipartitions fortes. Une bipartition élémentaire est forte. Les bipartitions fortes forment donc une sous-famille arborée d'une famille bipartitive.

La section précédente a montré que l'arbre $\mathcal{T}_\mathcal{F}$ permet de lire, d'un seul coup d'œil, toutes les bipartitions fortes de $\mathcal{F}$. Nous allons maintenant étendre ce résultat aux familles bipartitives. En effet, $\mathcal{T}_\mathcal{B}$, muni d'un marquage adéquat, permet de lire **toutes** les bipartitions d'une famille $\mathcal{B}$! Notons que $\mathcal{T}_\mathcal{B}$ est de taille $O(|V|)$ tandis que la famille de *toutes* les bipartitions de V est bipartitive, et qu'elle possède $2^{n-1} - 1$ bipartitions...

Pour un nœud N de $\mathcal{T}_\mathcal{F}$ et une arête a adjacente à N, $E(N, a)$ est l'ensemble des feuilles du sous-arbre de $\mathcal{T}_\mathcal{F} - a$ ne contenant pas N. $\Gamma(N)$ est l'ensemble des arêtes adjacentes à N.

Lemme 2 *Soit $\mathcal{F}$ une famille arborée et $P = \{P^0, P^1\}$ une bipartition de V. Si P ne chevauche aucune bipartition de $\mathcal{F}$ alors il existe un unique nœud N de $\mathcal{T}_\mathcal{F}$ et un ensemble $A \subset \Gamma(N)$ d'arêtes tels que*

$$P = \{\bigcup_{a \in A} E(N, a), \bigcup_{a \in \Gamma(N) \setminus A} E(N, a)\}$$

Démonstration: Ce lemme est une reformulation du lemme 1.

□

Étant donnée une bipartition faible P d'une famille bipartitive $\mathcal{B}$ (qui donc ne chevauche aucune bipartition forte de la famille), le nœud de $\mathcal{T}_\mathcal{F}$ *correspondant* à E est le nœud défini par le lemme précédent.

Théorème 3
Soit $\mathcal{B}$ une famille faiblement bipartitive. Les nœuds de $\mathcal{T}_\mathcal{B}$ peuvent être marqués `complet`*,* `circulaire` *ou* `premier` *de telle sorte que*

– *pour tout nœud* `complet` *N et tout $A \subset \Gamma(N)$ différent de Ø et de $\Gamma(N)$ on ait :*

$$\{\bigcup_{a \in A} E(N, a), \bigcup_{a \in \Gamma(N) \setminus A} E(N, a)\} \in \mathcal{B}$$

– *les arêtes adjacentes à un nœud* `circulaire` *peuvent être ordonnés de* a_1 *à* a_k *de telle sorte que pour tous* $1 \leqslant b < c \leqslant k$ *on ait :*

$$\{ \bigcup_{b \leqslant i \leqslant c} E(N, a_i), \bigcup_{i<b \text{ ou } i>c} E(N, a_i)\} \in \mathcal{B}$$

Réciproquement, toute bipartition $P \in \mathcal{B}$ *est*

– *soit forte,*

– *soit une bipartition faible correspondant à un nœud* `complet`*,*

– *soit une bipartition faible correspondant à un nœud* `circulaire` *et respectant son ordre.*

Démonstration: Soit $\mathcal{N} = \{P_1, \dots P_k\} \subset \mathcal{B}$ l'ensemble des bipartitions faibles correspondant à un nœud N de $\mathcal{T}_\mathcal{B}$. Si $\mathcal{N} = \emptyset$ alors on le nœud est défini comme étant `premier`. Dans toute la suite de cette preuve on considérera le cas $\mathcal{N} \neq \emptyset$ (ce qui implique $|\mathcal{N}| \geqslant 2$) et on ne considérera, sauf mention contraire, que des bipartitions de $\mathcal{N}$.

Soient $a_1 \dots a_k$ les arêtes de $\mathcal{T}_\mathcal{B}$ adjacentes à N. On pose $F_i = E(N, a_i)$. $\{F_1, \dots F_k\}$ est une partition de V. Pour tout i, $\{F_i, V \setminus F_i\}$ est une bipartition forte de $\mathcal{B}$, par définition de l'arbre $\mathcal{T}_\mathcal{B}$.

Toute bipartition de $\mathcal{N}$ est de la forme $\{\bigcup_{i\in I} F_i, \bigcup_{j\notin I} F_j\}$. Prendre $\{1, \dots k\} \setminus I$ à la place de I définit bien sûr la même bipartition. Dorénavant, pour parler des bipartitions de $\mathcal{N}$ on parlera de parties de $\{1, \dots k\}$. Ce formalisme est plus concis. $\overline{I} = \{1, \dots k\} \setminus I$ définit la même bipartition que I.

Lemme 3 *Pour tout* $b \in \{1, \dots k\}$ *il existe* $c \neq b$ *tel que* $\{b, c\}$ *soit une bipartition faible.*

Démonstration: Supposons le contraire : toute bipartition faible de $\mathcal{N}$ est de la forme $I \subset \{1, \dots k\}$ avec $b \in I$ et $|I| \geqslant 3$. Soit I_0 une bipartition faible de $\mathcal{N}$ avec $b \in I_0$ qui minimise $|I|$. I_0 est faible donc chevauchée par une certaine bipartition I_1. On suppose $b \in I_1$ (quitte à complémenter I_1). D'après les axiomes, la bipartition $I_0 \cap I_1$ est une bipartition de $\mathcal{B}$. Or puisque I_0 est minimale pour les bipartitions faibles, c'est une bipartition forte avec $I_0 \cap I_1 = \{b\}$. On en déduit que toute bipartition qui chevauche I_0 est de la sorte : elle coupe I_0 en $\{b\}$ et $I_0 \setminus \{b\}$. $I_0 \setminus \{b\}$ est donc une bipartition forte. Mais $|I_0| \geqslant 3$ donc $|I_0 \setminus \{b\}| \geqslant 2$, contradiction : il existe une bipartition forte à laquelle ne correspond aucune arête de $\mathcal{T}_\mathcal{B}$.

□

Soit $G_\mathcal{N}$ le graphe ayant $\mathcal{N}$ comme sommets et une arête reliant deux bipartitions ssi [15] elles se chevauchent.

Lemme 4 *$G_\mathcal{N}$ est connexe.*

15. dans ce mémoire *ssi* abrège *si et seulement si*.

Démonstration: Supposons que le graphe de chevauchement ait deux composantes connexes. Soit $\mathcal{C} = I_1, \ldots I_l$ une composante dans ce formalisme. Si $I_i \circledcirc I_j$ alors par définition des familles faiblement bipartitives $I_i \cup I_j$ est une bipartition de $\mathcal{N}$. Soit $J_i = I_1 \cup \ldots \cup I_i$. Nulle bipartition ne chevauche J_l. Si J_i est une bipartition, elle est forte : on a donc $J_l = \{1, \ldots k\} \setminus \{b\}$. Mais d'après le lemme précédent une bipartitions $\{b, c\}$ chevauche alors J_1. donc $J_l = \{1, \ldots k\}$. Soit $\mathcal{C}' = I'_1, \ldots I'_m$ une autre composante de chevauchement et $J'_i = I'_1 \cup \ldots \cup I'_i$. On a aussi $J'_m = \{1, \ldots k\}$. Or $J_1 \cap J'_1 = \emptyset$ (car on peut, quitte a complémenter I_1, toujours représenter deux les bipartitions $I_1 = J_1$ et $I'_1 = J'_1$ par deux intervalles d'intersection vide). Donc il existe i et j tels que $J_i \circledcirc J'_j$. prenons i et j minimums. D'après la proposition 1 page 25 $I_i \circledcirc I'_j$, contradiction.

□

Une bipartition est un **duo** si elle est de la forme $\{a, b\}$ dans notre formalisme. Le lemme 3 a établi que les duos recouvrent $\{1, \ldots k\}$. On peut renforcer le lemme 4 :

Lemme 5 *Le graphe de chevauchement des duos est connexe.*

Démonstration: Supposons pour contradiction que le graphe de chevauchement des duos ait deux composantes différentes. Soit $\{a, b\}$ un duo de la première composante et $\{c, d\}$ un duo de la deuxième.

D'après le lemme 4 il existe un chemin $I_0 = \{a, b\}, I_1, \ldots I_l = \{c, d\}$ dans le graphe de chevauchement. On suppose sans perte de généralité que $I_0 \cap I_1 = \{b\}$ et $I_{l-1} \cap I_l = \{c\}$. On peut prendre $I_i \cap \{a, b, c, d\} = \emptyset$. On pose $I = I_1 \cup \ldots \cup I_{l-1}$. I est une bipartition de $\mathcal{N}$, à cause de l'axiome d'union de bipartitions fermant la famille. D'après ce même axiome $I \cup \{a, d\}$ (l'union de tout le chemin) est une bipartition. Elle chevauche $\overline{I}$, donc leur intersection $\{a, d\}$ est aussi une bipartition de $\mathcal{N}$. C'est un duo qui relie les deux composantes, contradiction.

□

Lemme 6 *Si les duos $\{a, b\}$, $\{a, c\}$ et $\{a, d\}$ appartiennent à $\mathcal{N}$ pour a, b c et d distincts, alors c'est aussi le cas de toutes les parties non-vides de $\{a, b, c, d\}$.*

Démonstration:

- Les singletons sont déjà dans la famille car forts.
- Par union, $\{a, b\}$ et $\{a, c\}$ donnent $\{a, b, c\}$.
- Par union, $\{a, b\}$ et $\{a, d\}$ donnent $\{a, b, d\}$.
- Par union, $\{a, c\}$ et $\{a, d\}$ donnent $\{a, c, d\}$.
- Par union, $\{a, b, c\}$ et $\{a, b, d\}$ donnent $\{a, b, c, d\}$.
- Par intersection, $\overline{\{a, d\}}$ et $\{a, b, c\}$ donnent $\{b, c\}$.
- Par intersection, $\overline{\{a, c\}}$ et $\{a, b, d\}$ donnent $\{b, d\}$.
- Par intersection, $\overline{\{a, b\}}$ et $\{a, c, d\}$ donnent $\{c, d\}$.
- Par union, $\{b, c\}$ et $\{c, d\}$ donnent $\{b, c, d\}$.

□

Nous avons maintenant tous les éléments pour distinguer les cas `complet` et `circulaire`, selon que la configuration du lemme 6 se présente ou non :

Lemme 7 *Si il existe un entier a appartenant à trois duos distincts, alors toutes les parties non-vides de $\{1 \dots k\}$ sont des bipartitions de$\{1, \dots k\}$. N est un nœud* `complet`.

Démonstration: Cela se démontre par récurrence sur une suite de parties $I_0 \dots I_{k-4}$. Le lemme 6 fournit le cas de base pour $I_0 = \{a, b, c, d\}$, où a est un entier appartenant à trois duos distincts. Ensuite, on utilise le lemme 5 : un duo $\{e, f\}$, $e \in I_i$ et $f \notin I_i$, chevauche la partie I_i. On trouve deux entiers g et h dans I_i pour être dans les conditions du lemme 6, puis les unions donnent toutes les parties suivantes.

□

Lemme 8 *Si tout tout entier appartient à au plus deux duos, alors tout entier appartient à exactement deux duos. Il existe une permutation σ de $\{1 \dots k\}$ telle que tout facteur de cette permutation est une bipartition de $\mathcal{N}$. N est un nœud* `circulaire`.

Démonstration: Puisque tout duo chevauche au maximum deux duos, et que le graphe de chevauchement des duos est connexe (lemme 5), alors le graphe de chevauchement est soit un chemin de longueur k soit un cycle. En fait si c'est un chemin, en prenant l'union de tous les duos sauf les deux extrémités de ce chemin, on obtient une bipartition à $k - 2$ sommets, qui est en fait un duo formé des deux extrémités de ce chemin : le chemin est en fait un cycle. Donc tout sommet appartient à exactement deux duos. σ est une permutation obtenue en lisant les entiers le long de ce cycle, en partant d'un point quelconque. Il est clair que tout facteur de cette permutation est une bipartition de $\mathcal{N}$: cela se prouve par récurrence en partant du duo extrémité gauche et en suivant le cycle de chevauchement, en faisant des unions.

□

Cela achève la preuve du théorème.

□

On remarque que les nœuds `circulaire` à plus de trois fils violent l'axiome (5) de la définition 5, qui distingue les familles faiblement bipartitive des familles bipartitives. Donc les arbres bipartitifs des familles bipartitives (non-faibles) n'ont pas de nœud `circulaire`.

Théorème 4
Soit $\mathcal{B}$ une famille bipartitive. Les nœuds de $\mathcal{T}_{\mathcal{B}}$ peuvent être marqués `complet` *ou* `premier` *de telle sorte que pour tout nœud* `complet` *N et tout $A \subset \Gamma(N)$ différent de $\emptyset$ et de $\Gamma(N)$ on ait :*

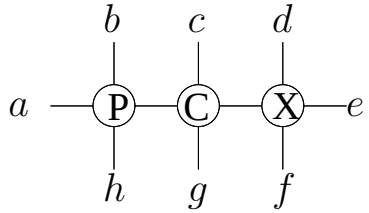
$$\{\bigcup_{a \in A} E(N, a), \bigcup_{a \in \Gamma(N) \setminus A} E(N, a)\} \in \mathcal{B}$$

Réciproquement, toute bipartition $P \in \mathcal{B}$ est forte, soit une bipartition faible correspondant à un nœud `complet`.

Exemple

Soit $V = \{a, b, c, d, e, f, g, h\}$. Pour plus de concision une bipartition de V sera noté par exemple $abc|defgh$.

Considérons l'arbre suivant :



Le nœud marqué P est premier, celui marqué X est complet et celui marqué C est circulaire. Les bipartitions de la famille correspondantes sont les huit bipartitions élémentaires comme $a|bcdefgh$; les bipartitions fortes $abh|cdefg$ et $abcgh|def$; les bipartitions faibles de la première classe de chevauchement (correspondant au nœud C) $abch|defg$ et $cdef|ghab$, et enfin bipartitions faibles de la deuxième classe de chevauchement (correspondant au nœud X) $abcdgh|ef$, $abcegh|df$ et $abcfgh|de$.

1.7 Lien entre familles partitives et bipartitives

Étant donné un ensemble V et une famille $\mathcal{P} \subset 2^V$, on construit la famille de bipartitions suivante :

$$\mathcal{B} = \{\{A, V \setminus A\} \mid A \in \mathcal{P} \wedge A \neq V\}$$

Proposition 5
Si $\mathcal{P}$ est partitive alors $\mathcal{B}$ est une famille bipartitive.
Si $\mathcal{P}$ est faiblement partitive alors $\mathcal{B}$ est une famille faiblement bipartitive.

Démonstration: Si $\{A, V \setminus A\}$ chevauche $\{A', V \setminus A'\}$, alors A chevauche A' donc :

1. $A \cap A' \in \mathcal{P}$ donc $\{A \cap A', V \setminus (A \cap A')\} \in \mathcal{B}$
2. $A \setminus A' \in \mathcal{P}$ donc $\{A \setminus A', V \setminus (A \setminus A')\} = \{A \cap (V \setminus A'), A' \cup (V \setminus A)\} \in \mathcal{B}$
3. $A' \setminus A \in \mathcal{P}$ donc $\{A' \setminus A, V \setminus (A' \setminus A)\} = \{A' \cap (V \setminus A), A \cup (V \setminus A')\} \in \mathcal{B}$
4. $A \cup A' \in \mathcal{P}$ donc $\{A \cup A', V \setminus (A \cup A')\} = \{(V \setminus A) \cap (V \setminus A'), A \cup A'\} \in \mathcal{B}$
5. enfin si $A \Delta A' \in \mathcal{P}$ ($\mathcal{P}$ est partitive non-faible) on a
   $\{A \Delta A', V \setminus (A \Delta A')\} = \{A \Delta A', A \Delta (V \setminus A')\} \in \mathcal{B}$

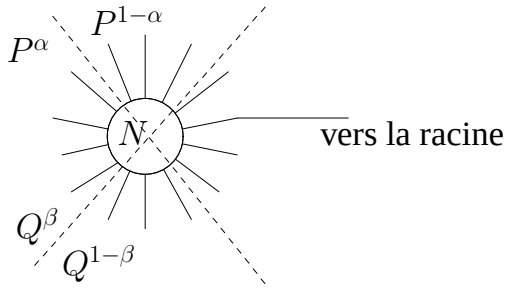
□

De façon réciproque, on peut transformer une famille bipartitive (resp. faiblement bipartitive) $\mathcal{B}$ en famille partitive (resp. faiblement partitive) en fixant une racine R à l'arbre bipartitif $\mathcal{T}_{\mathcal{B}}$. On obtient alors exactement l'arbre partitif d'une certaine famille $\mathcal{P}$. $\mathcal{P}$ contient comme parties fortes le côté de chaque bipartition opposé à R (en identifiant bipartitions fortes et arêtes de $\mathcal{T}_{\mathcal{B}}$).

Les nœuds `complet` et `premier` gardent leur sens. Notons bien que la famille contenant les *deux* côtés d'une bipartition n'est pas partitive.

Proposition 6
Si $\mathcal{B}$ est bipartitive alors $\mathcal{P}$ est une famille partitive.
Si $\mathcal{B}$ est faiblement bipartitive alors $\mathcal{P}$ est une famille faiblement partitive.

Démonstration: Soient deux bipartitions $P = \{P^0, P^1\}$ et $Q = \{Q^0, Q^1\}$ qui se chevauchent. Dans $\mathcal{T}_{\mathcal{B}}$, ces partitions correspondent (au sens du lemme 2) au même nœud N. Si N est choisi comme racine de $\mathcal{T}_{\mathcal{P}}$ alors P^0, P^1, Q^0 et Q^1 sont dans $\mathcal{P}$. Donc d'après les axiomes des familles faiblement bipartitives, leurs union et intersections sont dans $\mathcal{P}$, et si $\mathcal{B}$ est bipartitive leurs différences sont aussi dans $\mathcal{P}$. Si N n'est pas racine, il existe α et β tels que P^α et Q^β soient dans $\mathcal{P}$, mais pas $P^{1-\alpha}$ ni $Q^{1-\beta}$. Le chemin allant de N à la racine passe par une arête de $\Gamma(N)$ qui correspond à $P^{1-\alpha}$ et $Q^{1-\beta}$.



D'après les axiomes des familles faiblement bipartitives, $\{P^\alpha \cap Q^\beta, P^{1-\alpha} \cup Q^{1-\beta}\}$ est dans $\mathcal{B}$, et $P^\alpha \cap Q^\beta$ est dans $\mathcal{P}$ car c'est le côté qui ne contient pas la racine. De même $\{P^\alpha \cap Q^{1-\beta}, P^{1-\alpha} \cup Q^\beta\}$ est dans $\mathcal{B}$, et $P^\alpha \cap Q^{1-\beta} = P^\alpha \setminus Q^\beta$ est dans $\mathcal{P}$ car c'est le côté qui ne contient pas la racine. On termine la preuve avec les deux autres cas. Et si la famille est partitive, $\{P^\alpha \Delta Q^\beta, P^\alpha \Delta Q^{1-\beta}\}$ est dans $\mathcal{B}$, et $P^\alpha \Delta Q^\beta$ est dans $\mathcal{P}$ car, une fois encore, c'est le côté qui ne contient pas la racine.

□

1.8 Applications

Ce tableau présentes certains objets, connus ou introduits ici, qui correspondent à des familles partitives ou bipartitives. Il est malheureusement trop court : il doit exister bien plus de décompositions rentrant dans ce formalisme... En fait ces outils sont appliqués essentiellement aux *décompositions de graphes* : l'ensemble V considéré est alors l'ensemble des sommets d'un graphe donné. Les décompositions *modulaires* (au sens large, englobant des généralisations des modules) de graphes sont l'objet de cette thèse. Le chapitre suivant en présente certaines déjà connue, et les chapitres 3 et 4 de nouvelles.

Familles partitives	
– Décomposition modulaire des graphes non-orientés	§2.2 page 44
– Décomposition modulaire des k-structures, $k \geqslant 3$, et spécialisations	§2.3 page 53
– Décomposition en bloc des graphes d'héritage	§2.5 page 58
– Décomposition en comités des hypergraphes	§2.4 page 58
– Famille des q-sesquimodules, q donné	§3.4 page 69
– Famille des 2-modules boiteux $\{q, M\}$, q donné	§3.4 page 69
– Décomposition disjonctive des fonctions booléennes	[Ash59]

Familles faiblement partitives	
– Décomposition modulaire des graphes orientés, 2-structures, etc.	§2.3 page 53
– Famille des bimodules canoniques	chapitre 4 page 97
– *modules linéaires* d'une matrice aux 1 consécutifs	[HM03]

Familles bipartitives	
– Décomposition en 2-joints	§3.6 page 73
– Décomposition en coupes des graphes non-orientés	§3.5.2 page 72
– Décomposition en co-coupes des graphes non-orientés	§3.5.2 page 72

Familles faiblement bipartitives	
– Décomposition en coupes des graphes orientés	[Cun82]
– *modules circulaires* d'une matrice aux 1 circulaires	[HM03]
– Césures d'un multigraphe non-séparable	[Tut66, CE80]

Cunningham et Edmonds présentent [Cun73, CE80] trois applications de leur théorie qui diffèrent beaucoup de la décomposition modulaire. Les deux premières sont une décomposition des systèmes d'équations linéaires et une décomposition des matroïdes, qui correspondent peut-être à des familles faiblement partitives ou bipartitives. La troisième est en quelque sorte une décomposition en composantes 3-connexe des graphes, que j'expose brièvement ici.

Un graphe est **non-séparable** s'il est connexe et que, pour tout sommet x, $G - x$ est connexe. Une **césure**[16] est une bipartition $\{E_1, E_2\}$ de l'ensemble E des *arêtes* d'un graphe G telle que E_1 et E_2 contiennent chacune au moins deux éléments, et que il existe au plus[17] deux sommets adjacents à des arêtes de E_1 *et* de E_2. On peut reformuler un résultat de Cunningham et d'autres (Tutte [Tut66] en particulier) en :

16. anglais *split*, mais ne doit pas être confondu avec le terme *split* employé dans un sens *très* différent et traduit par *coupe* dans ce mémoire, voir §3.5.2.

17. dans un graphe non-séparable ce « au plus » devient « exactement ».

Théorème 5
La famille des césures d'un multigraphe non-séparable G est faiblement bipartitive.
Les arêtes de G qui sont feuilles d'un même nœud `premier` *(resp.* `complet`*,* `circulaire`*) de l'arbre bipartitif induisent dans G une composante 3-connexe (resp. un faisceau*[18] *, un cycle).*

18. anglais *bond* : deux sommets reliés par au moins trois arêtes dans le multigraphe.

Chapitre 2

Graphes, décomposition modulaire et variations sur ce thème

Je composerai jusqu'à la décomposition
Serge GAINSBOURG

LES GRAPHES sont l'objet de toute la suite de ce mémoire, passé ce prologue sur des outils ensemblistes. Nous introduisons d'abord ce concept, ainsi que l'opération sur les graphes qui est le cœur du sujet de ma thèse, la décomposition modulaire. Elle est présentée d'abord, de façon plus pédagogique, dans le cas particulier des graphes, puis de façon plus abstraite dans le cas général des k-structures. D'autres décompositions de graphes et de généralisations de graphes, voisines de celles-ci, seront également décrites.

2.1 Graphes. De leur intérêt

Venons-en au principal sujet de ce mémoire : les **graphes**. Les graphes sont des objets très élémentaires des mathématiques, car ils ne réclament pour leur définition que très peu de prérequis, en théorie des ensembles et en combinatoire. Le même mot est utilisé en analyse (graphe d'une fonction) ou en statistiques, mais dans un sens très différent.

Il s'agit d'un vieux concept : leur première utilisation est due semble-t-il à Euler, et a été popularisée par la chimie et la géométrie avant qu'une communauté dédiée ne se forme. Le concept nous est parvenu avec une terminologie manifestement géométrique. Les francophones utilisent l'ouvrage de référence de Claude Berge [Ber70] ; je me suis également servi avec plaisir de l'ouvrage de Douglas West [Wes96] qui contient de très jolies démonstrations.

Bien que les graphes soient définis comme relation, leur nom même indique la différence de taille qui rend leur manipulation aisée et leurs théorèmes facilement prouvable : l'aspect *graphique*, visuel, permettant d'appréhender l'objet d'un seul coup d'œil. Le *dessin de graphes*

constitue une branche entière de la discipline. Utilisée en dessin de graphes [SM95], la *décomposition modulaire* dont il sera beaucoup question devient un puisant outil pour la visualisation des graphes, et permet à l'œil de saisir encore davantage d'un dessin.

Presque toutes les communautés scientifiques font usage de graphe : biologistes (tant en génomique qu'en analyse du métabolisme), chimistes (pour représenter des molécules et des réactions), physiciens des matériaux, ingénieurs (pour quantité de modélisations) économistes, planificateurs, géographes, généalogistes, etc. Les graphes sont donc un des piliers de la Science.

2.1.1 Définition et notations

Un graphe n'est rien d'autre que la donnée d'un ensemble *fini* de **sommets**, noté usuellement[1] V, et d'une *relation d'adjacence* sur V, noté usuellement E. Il n'est question ici que de graphes *non-orientés* ; les différentes généralisations du concept seront vues en §2.3.2 page 54. La relation d'adjacence est binaire, antiréflexive et symétrique. On peut aussi voir E comme un ensemble de parties de V à deux éléments.

E est l'ensemble des **arêtes** ; on note $n = |V|$ et $m = |E|$. Une arête **relie** x et y, qui sont alors **voisins** ; on note une arête $(x, y) \in E$ et non $\{x, y\}$ bien qu'il s'agisse d'un ensemble non-ordonné. Deux parties disjointes A et B de V sont **adjacentes** si tous les sommets de A sont adjacents à tous ceux de B. En cas d'absence de relation entre x et y on parle de **non-arête** ; s'il n'y a aucune arête entre A et B ces deux ensembles sont **non-adjacents**. Notons que A et B sont en général ni adjacents ni non-adjacents.

Le **voisinage** ou **adjacence** d'un sommet est $\Gamma(x) = \{y|(x, y) \in E\}$ et son **non-voisinage** est $\overline{\Gamma}(x) = V \setminus (\Gamma(x) \cup \{x\})$. Le **degré** d'un sommet est la taille de son voisinage. Le **complémentaire** $\overline{G}$ d'un graphe permute voisinage et non-voisinage pour tous les sommets. Le graphe **induit** par un ensemble $S \subset V$ est $G[S] = (S, E \cap (S \times S))$. Deux sommets x et y sont **faux jumeaux** si $\Gamma(x) = \Gamma(y)$ et **vrais jumeaux** si $\Gamma(x) \cup \{x\} = \Gamma(y) \cup \{y\}$. Deux vrais jumeaux sont unis par une arête, ce qui n'est pas le cas des faux. Comme on verra plus loin, deux jumeaux forment un *module*.

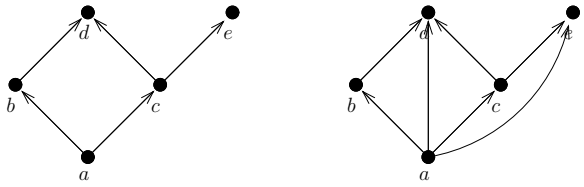
Dans un graphe *orienté*, la relation d'adjacence n'est plus symétrique. Un élément $(x, y) \in E$ est alors appelé un **arc** et diffère de (y, x). x et y sont sur une **arête** s'il y a deux arc entres, sur un **arc simple** s'il n'y a qu'un arc entre, et sur une **non-arête** sinon. Les concepts définis avant s'étendent aux graphes orientés.

Le *codage* usuel des graphes (orientés ou non) utilise soit une matrice de taille $n \times n$, soit la donnée, pour chaque sommet, de sa **liste d'adjacence**. Ce second codage est préférable car il prend une place $O(n + m)$, en général très inférieure[2].

1. Ces notations viennent bien sûr tout droit du monde anglo-saxon : V as Vertices et E as Edges. C'est un des rares cas où je n'ai pas utilisé l'usage français $G = (X, A)$. Un sommet sera toutefois souvent noté x au lieu de v.

2. Par exemple pour le *graphe du Web*, objet de nombreuses recherches, n est de l'ordre de grandeur du milliard et m de l'ordre de $4n$. Stocker un tel objet en listes d'adjacence est possible (à peine) avec les moyens actuels, mais complètement utopique sous forme de matrice.

Un **ordre** (strict, partiel) est une relation binaire, antiréflexive, antisymétrique et transitive. L'étude des ordres rentre dans le cadre de l'étude des graphes orientés ; d'ailleurs une façon usuelle de représenter les ordres est le *diagramme de Hasse*, graphe où les arcs de transitivité manquent.



*Graphes d'une relation d'ordre. Les arcs de transitivité, tels (a, d), manquent sur le dessin de gauche, qui est la **réduction transitive**, ou diagramme de Hasse. Toutes les comparaisons de l'ordre sont représentées sur le dessin de droite.*

Je n'irai pas plus loin dans la description de la théorie des graphes basique, que le lecteur est supposé connaître. Ce mémoire fait usage essentiellement des notions de *chemin* (suite de sommets reliés par des arêtes), *connexité* (propriété d'un graphe qu'il existe un chemin entre toutes paires de sommets) ainsi qu'à la *coloration*, sujets qui sont développés dans les ouvrages de référence [Ber70, Wes96].

2.1.2 Classes de graphes

Une **classe** de graphe est l'ensemble des graphes vérifiant une propriété donnée ; une classe est en général un ensemble infini (dénombrable). De nombreuses classes de graphes existent. [BLS99] en donne une liste assez impressionnante, beaucoup étant en relation avec la décomposition modulaire (qui sert à leur définition ou à leur reconnaissance). Je recommande chaudement cet ouvrage, d'une lecture captivante...

Nous rencontrerons les classes de graphes suivantes :

- Les **arbres** sont les graphes connexes à n sommets et $n-1$ arêtes.
- Les **cycles**. C_n est le graphe connexe à n sommets ou chaque sommet est de degré 2.
- Les **chemins**. P_n est le graphe C_n moins une arête.
- Les **cliques**. K_n est le graphe **complet**, à n sommets et $\frac{n(n-1)}{2}$ arêtes.
- Les **stables**. $\overline{K_n}$ est un graphe sans arête.
- Les **bicliques** ou **bipartis complets**. $K_{n,m}$ possède un ensemble de n sommets adjacent à un ensemble de m sommets, sans autres arêtes.

D'autres classes plus complexes apparaissent dans ce mémoire : cographes, graphes de permutation, d'intervalles, de comparabilité, etc. (voir l'index).

2.2 Décomposition modulaire des graphes

Cette partie présente la décomposition modulaire. Les éventuelles redites par-rapport au reste du mémoire sont volontaires, afin que la lecture puisse en être indépendante, et accessible au non-spécialiste.

La décomposition modulaire se présente comme l'un des outils majeurs de la théorie des graphes. Historiquement, elle vient d'études sur les ordres : c'est Tibor Gallaï [Gal67] qui a donné la première caractérisation des graphes premiers en regardant les ordres partageant le même graphe de comparabilité [3], et qui établit la grande proximité de cette décomposition avec le problème de l'*orientation transitive* [4] : il énonce le théorème 7 en caractérisant toutes les *classes de forçage* d'un graphe.

L'idée de module est, paraît-il, déjà présente dans [Cha47] pour les relations de congruence. Concernant les graphes, le concept apparaît sous le nom de *closed sets* [5] dans [Gal67], *stable sets* [6] dans [SF70], *clumps* dans [Bla78], *partitive sets* dans [Sum73, Gol80], *intervals* [7], dans [Sch81], *clans* dans [ER90b, ER90c], *externally related set* dans [HM79, CHM81], *autonomous sets* dans [Möh85b], *homogeneous sets* dans [BLS99], avant que le terme de *module* ne s'impose.

Elle a été étendue aux hypergraphes par [CHM81] (sous une forme incompatible avec celle-ci, présentée §2.4 page 58), aux relations k-aires par [MR84, Möh85b], puis aux 2-structures par [ER90b, ER90c]. [EM94] lui donne sa forme la plus générale en l'adaptant aux k-structures, qui forment l'étage supérieur de la fusée de la décomposition modulaire : c'est en effet —à ma connaissance— la généralisation suprême des graphes pour laquelle la notion de module ait un sens.

2.2.1 Présentation des modules

Un **module** est un ensemble de sommets d'un graphe qui se comporte comme *un seul* sommet vis-à-vis de l'extérieur. Un sommet extérieur au module est soit adjacent à tous les sommets du module, soit adjacent à aucun. Ainsi un module peut se dessiner comme un « gros sommet », entourant tous les sommets du module. Une adjacence de l'extérieur vers l'intérieur se dessine une seule fois, simplifiant ainsi le dessin et augmentant sa lisibilité.

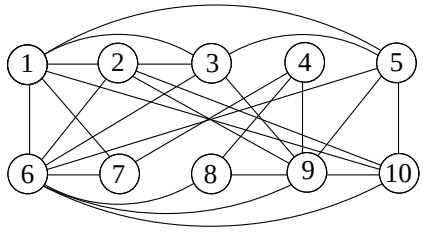
3. Ce qui nous amène à définir une nouvelle classe de graphes : les graphes de **comparabilité**, qui sont les graphes non-orientés qui sont la projection d'un ordre.

4. à tel point que les algorithmes de décomposition modulaire sont souvent des algorithmes d'orientation transitive après quelques changements mineurs [MS94b, MS99, Pau98]

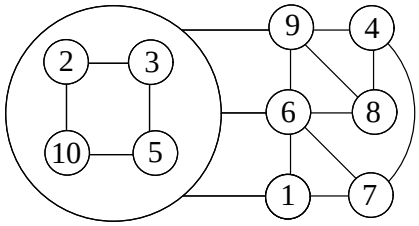
5. traduction anglaise de l'Allemand...

6. traduction anglaise du Russe...

7. car les propriétés de fermeture par union, intersection et différence sont partagées par les intervalles de la droite réelle, famille infinie correspondant en quelque sorte à un « gros » nœud `linéaire`...



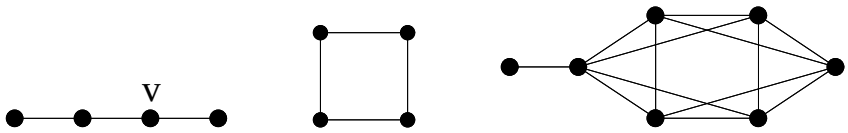
Un graphe dessiné de façon quelconque.



Le même en faisant apparaître un module. La clarté du dessin y gagne !

L'opération de **substitution**[8] remplace, dans un graphe donné G, un sommet v par un graphe H. Elle est notée G_v^H. Tous les sommets de G précédemment adjacents à v sont maintenant adjacents à tous les sommets de H. H est alors un module du nouveau graphe. Les modules apparaissent donc naturellement dans des graphes d'origine « humaine », comme conséquence des différentes *compositions* à notre disposition pour construire des graphes. La structure des modules retrace l'historique des substituions effectuées. Les opérations de substitutions ont été d'abord étudiées par [Jol73, Sab59].

Réciproquement, si M est un module de G, on peut **compacter**[9] M : tous ses sommets disparaissent de G, remplacés par un seul nouveau sommet m. Tous les sommets précédemment adjacent à M deviennent adjacent à m. Cette opération se note G_M^m.



$G = (G_v^H)_H^v$ $\qquad$ H $\qquad$ G_v^H

Substitution et compactage entre les deux graphes de gauche et celui de droite.

La **décomposition modulaire** d'un graphe est l'ensemble de tous ses modules. Un module est **trivial** s'il est égal à $\emptyset$, V ou bien $\{v\}$, $v \in V$ (clairement de tels ensembles sont toujours des

8. Formellement, la **substitution** d'un graphe H à un sommet v dans G produit le graphe

$$G_v^H = (\ (V_G \setminus \{v\}) \uplus H,\ \{\ (x,y) \mid (x,y) \in E_H \vee ((x,y) \in E_G \wedge v \notin \{x,y\})$$
$$\vee (v \neq x \in V_G \wedge y \in V_H \wedge (x,v) \in E_G)\ \}\ )$$

La substitution de plusieurs graphes à plusieurs sommets différents de G est notée $G_{v_1\ v_2}^{H_1\ H_2} \ldots^{H_k}_{v_k}$

9. Formellement, le **compactage** d'un module M en un sommet m dans G, $m \notin V(G)$, produit le graphe

$$G_M^m = (\ (V_G \setminus M) \uplus \{m\},\ \{\ (x,y) \mid ((x,y) \in E_G\ \wedge\ M \cap \{x,y\} = \emptyset)$$
$$\vee\ (x \in V_G \setminus M\ \wedge\ y = m\ \wedge\ \exists z \in M\ tq\ (x,z) \in E)\}\ )$$

Substitution et compactage sont associatives à gauche.

modules). Un graphe est **premier** s'il ne comporte que des modules triviaux. Ce sont les graphes *indécomposables* par la décomposition modulaire ; exactement comme les nombres premiers sont les nombres indécomposables par la division, car n'ayant que des diviseurs triviaux.

Il est bien connu que la quantité de nombres premiers inférieurs à n est $\Theta(\frac{n}{\log(n)})$: il y a donc très peu de nombres premiers. Encouragé par cette bonne nouvelle, le lecteur sera sans doute amené à conjecturer que la quantité de graphes premiers est elle aussi faible. Malheureusement, il n'en est rien. En effet, en notant $\mathcal{G}_n$ la classe des graphes à n sommets, la probabilité qu'un graphe tiré aléatoirement dans $\mathcal{G}_n$ soit premier tend vers 1 quand n tend vers l'infini... Ce résultat décevant doit être tempéré par le fait que l'activité humaine produit en général des graphes largement décomposables. De plus, il motive la recherche d'extensions de la décomposition modulaire, permettant de décomposer des graphes premiers en utilisant des ensembles de décompositions plus « subtils » que les modules. Le chapitre 3 présente diverses possibilités pour le faire, dans une extension des modules qui généralise toutes ces possibilités : les *2-modules*, et le chapitre 4 dans une extension des modules spécifique aux graphes bipartis.

2.2.2 Le théorème de décomposition modulaire

Cette section est un petit cours présentant la décomposition modulaire. Les propositions énoncées sont faciles à prouver (et, selon la formule consacrée, laissée en exercice au lecteur) [10] [MR84] reste une référence incontournable sur le sujet, à laquelle le lecteur est renvoyé.

Un module est **fort** s'il ne chevauche aucun autre module. On remarquera que les modules triviaux sont forts. Deux modules forts sont soit inclus l'un dans l'autre, soit d'intersection vide. Cela définit un *ordre d'inclusion*, qui est un ordre *arborescent*. Les modules forts peuvent en effet être disposés en un arbre, dont la racine est le module trivial V et les feuilles les modules triviaux $\{v\}$. Si le graphe est premier, l'arbre n'est pas plus complexe. Sinon, l'arbre possède au moins un nœud de plus. Les premières propriétés de la décomposition modulaire sont :

Proposition 7
Un module de G est un module de $\overline{G}$.

Proposition 8
Si M est un module de G et $S \subset V(G)$ alors $S \cap M$ est un module de $G[S]$.
Réciproquement, si S est un module alors tout module de $G[S]$ est module de G.

Proposition 9
Si M et N sont deux modules alors $M \cap N$ est un module.
De plus si $M \cap N \neq \emptyset$ alors $M \cup N$ est un module.
De plus si $M \oslash N$ alors $M \Delta N$ est un module. [11]

10. Je pense d'ailleurs qu'un TD de second cycle sur la décomposition modulaire est tout à fait possible, en utilisant ces propositions comme exercices afin d'inférer le théorème final.

11. À l'inverse, le chapitre 1 s'intéresse aux familles de parties qui vérifient cette propriété, qui devient un axiome d'où découle un certain nombre de théorèmes.

Un module (resp. module fort) M est **maximal** si le seul module (resp. module fort) fort qui le contienne est $V(G)$.

Proposition 10
Si un graphe est non-connexe, alors ses modules forts maximaux sont ses composantes connexes.

On peut aller plus loin :

Proposition 11
Tout module d'un graphe non-connexe est soit une union de composantes connexes, soit inclus dans une seule composante connexe.

Proposition 12
Si G est connexe et $\overline{G}$ est connexe, alors les modules maximaux de G sont des modules forts. [12]

Le **quotient** d'un graphe G, dont les modules forts maximaux sont $M_1, \dots M_k$, est le graphe $(G^{m_1}_{M_1}) \cdots^{m_k}_{M_k}$.

Proposition 13
Le quotient d'un graphe est soit une clique, soit un stable, soit un graphe premier.

On peut préciser les cas : si G n'est pas connexe, ses modules forts maximaux étant ses composantes connexes, le quotient de G est un stable. D'après la proposition 7, si $\overline{G}$ n'est pas connexe (on dit que G est *co-connexe*) alors le quotient de G est une clique. Et si G et $\overline{G}$ sont tous deux connexes, les propositions 8 et 12 assurent que le graphe quotient n'a plus de module non-trivial.

Revenons à l'arbre d'inclusion des modules forts. Cet arbre est noté $\mathcal{T}_G$. Les propositions 10, 7 et 12 permettent de caractériser les modules forts maximaux de G :

- Si G est non-connexe, ce sont ses composantes connexes
- Si $\overline{G}$ est non-connexe, ce sont les composantes co-connexes de G (composantes connexes de $\overline{G}$)
- Sinon, ce sont ses modules maximaux.

Or les modules forts maximaux de G sont, dans $\mathcal{T}_G$, les fils $M_1 \dots M_k$ de la racine $V(G)$. Tout autre module fort M sera inclus dans un M_i donné. D'après la proposition 8, M est un module de $G[M_i]$, qui de plus est fort. Donc en recherchant **récursivement** les modules forts maximaux de $G[M_i]$ pour chaque module fort maximal M_i, on construit tout l'arbre de décomposition.

Introduisons une terminologie importante. Un module M est `parallèle` si le quotient de $G[M]$ est un stable, `série` si le quotient de $G[M]$ est une clique, et `premier` sinon ($G[M]$ étant alors connexe et co-connexe, son quotient est un graphe premier).

Les nœuds de $\mathcal{T}_G$, qui sont des modules forts, peuvent eux aussi être marqués `parallèle`, `série` ou `premier` selon le type du module. Les fils d'un nœud, dans cet arbre, sont des modules.

12. Pour prouver cette proposition, on peut supposer le contraire et, grâce à la proposition 9, faire apparaître des composantes.

Dans les deux premiers cas, ces modules sont des composantes connexes ou co-connexes du graphe induit, et *toute* union de composantes est un module. Dans le troisième cas, comme ces modules sont maximaux, l'union de plusieurs fils n'est un module que dans l'*unique* cas où l'on prend tous les fils. Cette différence est du comportement des fils vis-à-vis de l'union est fondamentale. Les nœuds `parallèle` et `série` seront appelés `complet` à cause de cette propriété.

Tous ces résultats sont résumés par le théorème de décomposition modulaire suivant. Deux modules forts sont frères s'ils ont le même père dans $\mathcal{T}_G$.

Théorème 6 (version non-récursive)
Soit M un module fort et $M_1, \dots M_k$ ses fils dans $\mathcal{T}_G$, et soit $I \subset [\![1, \dots k]\!]$ un ensemble tel que $2 \leqslant |I| < k$.

- *Si M est* `parallèle` *(resp.* `série`*) alors $\bigcup_{i\in I} M_i$ est un module faible* `parallèle` *(resp.* `série`*)*
- *Si M est* `premier` *alors $\bigcup_{i\in I} M_i$ n'est pas un module*

Réciproquement, tout module de G est

- *soit un module* `premier` *fort,*
- *soit un module* `parallèle` *ou* `série` *qui est l'union de modules forts frères.*

Théorème 7 (version récursive)
Soit G un graphe. Un et un seul des cas suivants est vrai

- *G n'a qu'un sommet.*
- *G est non-connexe. Ses modules forts maximaux sont ses composantes connexes.*
- *$\overline{G}$ est non-connexe. Les modules forts maximaux de G sont les composantes connexes de $\overline{G}$.*
- *G et $\overline{G}$ sont connexes. Les modules maximaux de G sont forts.*

Nous venons d'établir que certains d'entre eux, les modules **forts**, jouent un rôle distingué, étant en quelque sorte les *générateurs* de toute la famille.

Nous avons donc un moyen de représenter *tous* les modules d'un graphe : l'arbre d'inclusion des modules forts $\mathcal{T}_G$, que l'on appellera dorénavant **arbre de décomposition modulaire** de G. Or G peut avoir une quantité exponentielle de modules (par exemple dans la clique K_n il y a 2^n modules) mais possède moins de $2n$ modules forts (puisque $\mathcal{T}_G$ est un arbre à n feuilles sans nœud à un fils) ; et le stockage en mémoire de $\mathcal{T}_G$ demande un espace $O(n)$. L'arbre de décomposition modulaire est donc une *bonne* représentation de la famille des modules de G ! À cause de la maximalité des composantes, on a :

Proposition 14
Dans $\mathcal{T}_G$ les fils d'un nœud `parallèle` *(resp.* `série`*) ne peuvent être des nœuds* `parallèle` *(resp.* `série`*).*

Le théorème 7 donne une procédure récursive pour calculer $\mathcal{T}_G$: les quatre cas sont aisément distingables et permettent d'étiqueter la racine de $\mathcal{T}_G$ comme feuille, `parallèle`, `série` ou `premier` (respectivement). On poursuit récursivement sur les modules forts maximaux. Dans le cas 4, on trouve le plus gros module contenant un sommet v donné par *essais-erreurs* : à une étape donnée on possède un module M contenant v (initialisé à $\{v\}$). On essaye d'ajouter un autre sommet u à M, puis on ajoute itérativement tous les sommets x *distinguant* le nouvel ensemble : ceux qui sont adjacent à certains sommets, mais non à tous. Le plus petit module contenant M contient aussi x, car un module est *sans séparateurs* : nul sommet ne le distingue.

Si l'on obtient V, on repart avec l'ancien module, et sinon avec le nouveau. Il n'est pas nécessaire de revenir plus en arrière, car tout module contenant v, sauf V, est inclus dans le résultat recherché. On obtient ainsi un algorithme en $O(n^4)$ de décomposition modulaire implémentable par des étudiants de licence d'informatique, et on prouve

Proposition 15
La décomposition modulaire est un problème résoluble en temps polynômial.

Plusieurs équipes [CH94, MS99, DGM97] ont démontré que le problème est optimalement résoluble en $O(n+m)$. Les chapitres 5 et 8 donnent une nouvelle preuve de ce résultat avec un algorithme espéré être plus simple, sans battre la borne de $O(n+m)$ en séquentiel (et $O(n)$ en parallèle).

2.2.3 Décomposition modulaire et classes de graphes

La première classe de graphes que la décomposition modulaire permet de définir est :

Définition 6
Un cographe[13] *est un graphe sans module* `premier`.

Dans l'arbre de décomposition modulaire des cographes on ne trouve donc que des nœuds `parallèle` et `série`. D'après la proposition 14 ils sont alternés. Sumner a prouvé

Théorème 8 [Sum73]
G est un cographe si et seulement si

$$\forall H \subset V(G),\ G[H] \neq P_4$$

Une telle caractérisation d'une classe, par *sous-graphes exclus*, est extrêmement fréquente (il doit exister une centaine de théorèmes de ce genre, [BLS99] dresse un catalogue assez fourni !) La classe $\mathcal{C}$ est alors *héréditaire* : si $G \in \mathcal{C}$ et $H \subset V(G)$ alors $G[H] \in \mathcal{C}$. La décomposition modulaire est souvent un outil précieux pour les algorithmes de reconnaissance de telles classes.

13. Ce nom vient du fait qu'il y a bijection entre le *co*graphe et son arbre de décomposition modulaire, noté alors le *co*arbre.

On supprime un jumeau quand on remplace G par $G[V \setminus \{x\}]$, où x possède un jumeau. L'*élagage* des jumeaux consiste en la suppression itérée de tous les jumeaux possibles.

Théorème 9 [CLS81]
Les cographes sont les graphes dont l'élagage des jumeaux mène au graphe à un sommet.

[BLS99] donne un catalogue de pas moins de 8 conditions nécéssaires et suffisantes différentes pour qu'un graphe soit un cographe ! Les cographes trouvent des applications en logique : ils sont le *minterm graph* des formules logiques où chaque littéral apparaît exactement une fois [Gur77]. [HK89] s'en sert pour des problème d'apprentissage. [She86] les applique à des problèmes d'ordonnancement d'examens (sic).

2.2.4 Graphes orientés

Les *graphes orientés* sont également concernés par la décomposition modulaire. Puisque les graphes non-orientés peuvent être vus comme un cas particulier de graphes orientés (symétriques), les trois décompositions précédentes restent valables.

Il n'apparaît qu'un quatrième type. Un graphe orienté G est un k-**ordre** s'il existe une partition ordonnée $V_1 \uplus \dots V_k$ de V telle que pour tout $x \in V_i$ et tout $y \in V_j$, si $i < j$ alors $(x, y) \in E$ et $(y, x) \notin E$. Les n-ordres sont exactement les ordres totaux à n sommets.

Proposition 16
Il existe un unique k maximal tel que G soit un k-ordre. La partition ordonnée de V est alors unique.

On dira que G est un k-ordre *strict* si k est maximal. Toute classe de l'unique partition en k-ordre strict est un module. On l'appellera une **composante ordre**. Pour $i < j < k$ on constate que V_i ne peut distinguer $V_j \cup V_k$, V_k ne peut distinguer $V_i \cup V_j$, mais V_j distingue $V_i \cup V_k$. On en déduit que une union de composante ordre est un module si et seulement si les numéros de ces composantes se suivent. Par ailleurs, à cause de la maximalité des composantes on peut montrer qu'une composante ordre est un module fort maximal de G.

M est un module `ordre` si c'est un module fort de G et que $G[M]$ est au moins un 2-ordre. Ce cas exclut que M soit un module `série`, `parallèle` ou `premier` ; par contre c'est le seul cas possible en plus pour un graphe orienté. On rajoute alors au théorème 6 l'alinéa suivant :

– *Si M est* `ordre` *alors $\bigcup_{a \leqslant i \leqslant b} M_i$ est un module faible* `ordre` .

Et on reformule le théorème 7 en :

Théorème 10
Soit G un graphe orienté. Un et un seul des cas suivants est vrai

– *G n'a qu'un sommet.*
– *G est non-connexe. Ses modules forts maximaux sont ses composantes connexes.*
– *G est un k-ordre strict, $k \geqslant 2$. Ses modules forts maximaux sont ses composantes ordre.*

– *G et $\overline{G}$ sont connexes et G est un 1-ordre strict. Les modules maximaux de G sont forts.*

Tout ce qui a été dit par ailleurs reste vrai, sauf une chose : pour deux modules M et N qui se chevauchent, on n'a plus que $M \Delta N$ est un module, mais seulement $M \setminus N$ (et $N \setminus M$).

Les *ordres série-parallèle* [VLT82] sont les ordres dont la décomposition modulaire ne comporte pas de graphe `premier` : leur arbre de décomposition n'a que des nœuds `ordre` et `parallèle`, et leur graphe de comparabilité est un cographe. De nombreuses classes de graphes généralisent les cographes en n'autorisant que les nœuds `premier` d'une certaine forme [Hoà85, JO89, GRT97].

Le *dernier ancêtre commun* (ou *plus petit ancêtre*) de deux nœuds d'un arbre $\mathcal{T}$ est noté[14] $\mathrm{LCA}_{\mathcal{T}}(x, y)$.

Proposition 17
Soient x et y deux sommets d'un graphe $\mathrm{LCA}_{\mathcal{T}_G}(x, y)$
– *Si* $\mathrm{LCA}_{\mathcal{T}_G}(x, y)$ *est* `série` *alors* $\{x, y\}$ *est une arête*
– *Si* $\mathrm{LCA}_{\mathcal{T}_G}(x, y)$ *est* `parallèle` *alors* $\{x, y\}$ *est une non-arête*
– *Si* $\mathrm{LCA}_{\mathcal{T}_G}(x, y)$ *est* `ordre` *alors* $\{x, y\}$ *est un simple arc.*

De nombreuses autres propriétés de la décomposition modulaire, ainsi qu'une preuve complète des théorèmes et des exemples d'application, se trouvent dans l'étude de Möhring et Radermacher [MR84].

2.2.5 Utilité de la décomposition modulaire

Les graphes *P_4-sparse* [Hoà85, JO92], *P_4-reducible* [JO89], *P_4-extendible* [JO91], *P_4-tidy* [GRT97], *partner-limited* [RRT99], etc... sont des variantes des cographes, chez lesquelles la donnée de $\mathcal{T}_G$ est également *suffisante* pour coder le graphe G, car G est défini de façon univoque par son arbre de décomposition modulaire. Dans les autres cas, si on stocke pour chaque nœud `premier` M le quotient de $G[M]$, on a assez d'information pour retrouver tout le graphe. Un tel codage de G, si G est « suffisamment » décomposable, occupe « beaucoup » moins de place que le codage par liste d'adjacence, car en quelque sorte il factorise ces listes d'adjacence.

Un grand nombre de problèmes de décision et d'optimisation « durs » peuvent être résolus si une solution est connue pour le quotient de chaque nœud de l'arbre de décomposition, selon le procédé de *réduction*. Parmi les applications potentielles citons l'orientation transitive, la recherche d'une clique pondérée maximale, la coloration, etc. Par exemple,

– [Bol78] donne des algorithmes polynômiaux pour les graphes de comparabilités des problèmes (NP-complets en général) que sont la clique pondérée maximal, le couplage pondéré maximal, des approximation pondérées du problème de la coloration, et des techniques de construction de graphes parfaits ;

14. *Least Common Ancestor*, encore un anglicisme.

- [Möh85a] l'applique aux graphes d'intervalles et de comparabilité pour des problèmes de clique pondérée maximale, de stable maximal, etc ;
- [GR97] donne des algorithmes en $O(n(n+m))$ pour la clique maximale pondérée, le stable maximal et la coloration pondérée approchée, sur la classe des graphes sans P_5 ni $\overline{P_5}$;
- [GV97a] pour de nombreux problèmes pondérés d'optimisation sur les classes *extended P_4-reducible* et *extended P_4-sparse*, qui généralisent beaucoup de classes caractérisées par des restrictions sur la décomposition modulaire ;
- [Hir] fait le lien avec la dimension des ordres ;
- [Spi85] l'utilise pour reconnaître les graphes de comparabilité ;
- [Sid86] pour des problèmes d'ordonnancement ;
- [HPV01, Riz01] pour la reconnaissance en $O(n+m)$ des graphes de P_4-indifférence ;
- [SM95] en dessin de graphes ;
- [FG97] pour reconnaître les graphes *semi-P_4-sparse*.

De nombreuses classes de graphes, tels les graphes d'intervalles ou ceux de permutations, voient leurs membres premiers avoir de fortes contraintes, ce qui facilite la reconnaissance de telles classes, en utilisant un algorithme simple qui teste la propriété sur tous les nœuds premiers de $\mathcal{T}_G$. Dans le cas des graphes de comparabilité par exemple, la classe de forçage d'un graphe premier est unique. [Gol80, BLS99] donnent une étude assez exhaustive de ces techniques. À noter que peu de classes de graphes non-orientés ont été caractérisées, bien qu'assurément la décomposition modulaire soit un outil puissant pour elles aussi (voir [Mül97] par exemple).

La décomposition *homogène* [JO95] étend la décomposition modulaire. [Lan01] discute de la portée de cette généralisation [15]. Des paramètres de *largeur* de graphes ont été définis, et donnent des décompositions *en largeur* plus ou moins apparentées à la décomposition modulaire ou à ses extensions. Par exemple, la *cliquewidth* [16] d'un graphe est le maximum de la cliquewidth des graphes quotients apparaissant dans son arbre de décomposition modulaire. Ce qui entraîne que toutes les classes où l'arbre de décomposition modulaire a un nombre fini de quotients possibles ont une cliquewidth bornée (valant 2 pour les cographes et 3 pour les P_4-sparse, P_4-reducibles et P_4-tidy). Il y aurait encore sans doutes beaucoup d'autres lien à voir entre ces deux grandes familles de décompositions de graphes que sont les décompositions partitives et les décompositions en largeur...

15. l'arbre de décomposition possède un nouveau type de nœud, d'arité 2, qui coupe le quotient d'un graphe premier entre son *œil*, c'est-à-dire l'unique sommet par lequel ne passe aucun P_4, et les autres sommets par chacun desquels passe un P_4. [Bau96] étend les algorithmes linéaires de décomposition modulaire pour la calculer, en temps linéaire aussi : peu de manipulations d'arbre suffisent.

16. ou largeur de clique. Voir [Lan01] pour une explication francophone du concept, inventé par [CER93].

2.3 Décomposition modulaire des k-structures

Cette section traite des modules des k-structures, qui généralisent les graphes. La décomposition modulaire des k-structures est une application directe du théorème 1, puisque les modules forment une famille partitive. De plus c'est l'étude des modules des graphes qui a servi de point de départ pour inférer les propriétés plus générales des familles partitives. [EHPR96] fournit une liste de problèmes en relation avec les 2-structures et leur décomposition, et discute comment l'arbre de décomposition aide à résoudre des problèmes formulables en logique monadique du second ordre (MSOL), ce qui est le cas de beaucoup de problèmes NP-complets de graphes. Les 2-structures ont en fait été bien plus étudiées que les k-structures car elles ressemblent plus aux graphes et se représentent facilement [ER90b, ER90c, ER90a, EHR94, EHPR96]. [EGMS94, McC95] donnent des algorithmes en $O(n^2)$ pour leur décomposition modulaire[17].

2.3.1 k-structures

La terminologie présentée ici suit partiellement [EM94]. Notons V_*^k l'ensemble $\underbrace{V \times \ldots \times V}_{k fois} \setminus \{(\underbrace{x, \ldots, x}_{k fois}) | x \in V\}$. Pour $e \in V_*^k$, $e(i)$ est l'élément de e en i^{e} position.

Définition 7
*Une k-**structure** est un triplet $G = (V, E, k)$ où V est un ensemble fini de **sommets**, $k \geqslant 2$ est l'**arité** de G, et $E : V_*^k \to \mathbb{N}$ est la **fonction arête**.*

Soit $e = (x_1, \ldots x_k) \in V_*^k$. Le **support** de e est l'ensemble $sup(e) = \{x_1, \ldots x_k\} \subset 2^V$. On notera que $|sup(e)| \geqslant 2$. E est **réflexive** si

$$\forall e, e' \in V_*^k \;\; sup(e) = sup(e') \Rightarrow E(e) = E(e')$$

Deux k-structures $G = (V, E, k)$ et $G' = (V, E', k)$ sont **fortement isomorphes** si

$$\forall e, e' \in V_*^k \;\; E(e) = E(e') \iff E'(e) = E'(e')$$

$G = (V, E, k)$ et $G' = (V', E', k)$ sont **isomorphes** s'il existe une bijection $\phi : V' \to V$ telle que $G(V, E, k)$ et $G(\phi(V'), \phi(E'), k)$ soient fortement isomorphes. Tous les théorèmes énoncés sont *à isomorphisme près*, même si ce n'est pas explicitement écrit dans leur énoncé : on n'a ainsi pas à se préoccuper des valeurs effectivement prises par E, qui devient donc une simple *relation d'équivalence* entre k-uplets de V_*^k.

17. Le « 2 » qui préfixe l'expression « 2-structure » ne doit pas abuser le lecteur : les 2-structures n'ont rien à voir avec les 2-modules dont il sera question au chapitre 3 de cette thèse... bien qu'on puisse définir des 2-modules dans une 2-structure ! Ce 2 est en fait l'arité des « arêtes ».

2.3.2 La hiérarchie de généralisation des graphes

Les gens connaissent en général trois types de généralisation des graphes. En terme de *relations*, un **hypergraphe** est une relation quelconque, un **graphe orienté** une relation binaire, un graphe non-orienté une relation binaire antiréflexive symétrique, et un **multigraphe** une fonction binaire antiréflexive symétrique. Nous présentons ici tous ces objets comme cas particuliers des k*-structures*. La décomposition modulaire des k-structures s'applique à tous les cas particuliers, ce qui motive leur présentation ici.

- En se limitant à $k = 2$, on retrouve la notion usuelle d'**arête**, tandis que pour k quelconque, on trouve les k**-uplets hétérogènes** (*hétérogène* signifiant que la suite $(x, \ldots x)$ est interdite), dont une restriction est les **hyperarêtes** (k-uplets sans répétitions).
- En prenant E réflexive, on supprime l'*orientation* (dans le cas des graphes) car maintenant sont considérées des *parties* au lieu de *suites*. Une fonction réflexive $E : V_*^k \to \mathbb{N}$ est exactement une fonction $2^V \to \mathbb{N}$ (une fonction partielle en fait, car seules ont des images les parties à deux éléments au moins).
- En considérant la restriction $E : V_*^k \to \{0, 1\}$, la fonction arête indique une *présence* (1) ou absence (0) du k-uplet dans la structure, tandis que ne pas majorer E en fait une fonction de *coloration* des k-uplets, puisque les k-structures sont toujours considérées *à isomorphisme près*.

k	E réflexive?	max(E)	Nom de la structure
2	Oui	1	graphe (non-orienté, sans boucle)
2	Oui	c	multigraphe (non-orienté, sans boucle)
2	Non	1	graphe orienté (sans boucle)
2	Non	c	2-structure
k	Oui	1	k-hypergraphe (sans boucle)
k	Oui	c	k-multihypergraphe (sans boucle)
k	Non	1	relation k-aire (antiréflexive)
k	Non	c	k-structure

2.3.3 Décomposition modulaire

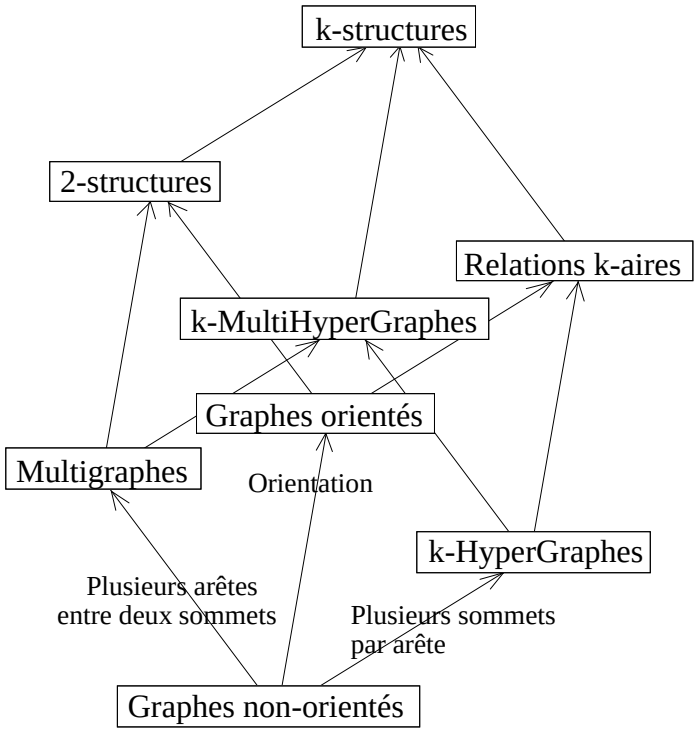
Soit $X \subset V$. $e \in V_*^k$ **transcende** X si $sup(e) \cap X \neq \emptyset$ et $\sup(e) \setminus X \neq \emptyset$. Deux k-uplets e et e' sont ***X*-équivalents** s'ils diffèrent seulement en deux éléments membre de X :

$$\exists i_0 \ (\forall i \neq i_0 \ e(i) = e'(i)) \ \wedge \ e(i_0) \in X \ \wedge \ e'(i_0) \in X$$

Définition 8 ([EM94])
*Un **module** est une partie $M \subset V$ telle que si e et e' sont M-équivalents et transcendent M alors $E(e) = E(e')$.*

Théorème et définition 9
*V, $\{v\}$, $v \in V$ et $\emptyset$ sont des modules de $G = (V, E)$ appelés modules **triviaux**.*

Cependant on ne considérera pas $\emptyset$ comme étant un module, pour ne pas créer d'exception au théorème et rester cohérent avec la définition des familles partitives. Un module est **fort** s'il ne chevauche aucun autre module. Les modules triviaux, en particulier, sont forts.



Hiérarchie des généralisations des graphes.

La définition des modules forts et triviaux donne immédiatement :

Proposition 18
Une k-structure possède $n+1$ modules forts triviaux et moins de n modules forts non-triviaux.

Définition 10
*Deux sommets x et y sont **jumeaux** si $\{x, y\}$ est un module.*

Le résultat majeur de la théorie de la décomposition modulaire est le suivant :

Théorème 11 [EM94]
La famille des modules d'une k-structure , $k \geqslant 3$, est partitive.
La famille des modules d'une 2-structure est faiblement partitive.

C'est un résultat classique et apparaissant sous différentes formes, pour ce qui concerne les graphes. On le trouve formalisé dans [CHM81] pour les graphes et hypergraphes (mais déjà informellement avant), dans [MR84] pour les relations k-aires, et dans [ER90b, ER90c] pour les 2-structures.

D'après la discussion page 28 sur l'unicité du marquage des nœuds, les nœuds à trois fils ou plus seront marqués sans ambiguïté, et l'arbre n'a pas de nœud à un fils. Nous allons voir comment marquer les nœuds à deux fils.

Le théorème 1 page 26 s'applique à la famille des modules d'un graphe. Il exprime l'existence d'un **arbre de décomposition modulaire** $\mathcal{T}_G$, l'arbre d'inclusion des module forts, qui permet de représenter toute la famille des modules. Élucidons ce à quoi peuvent bien correspondre les étiquettes `complet`, `linéaire` ou `premier` de $\mathcal{T}_G$.

Soit M un module fort non-singleton et $F_1^M, \dots F_p^M$ ses fils (dans $\mathcal{T}_G$, c'est-à-dire les modules forts maximaux inclus dans M). On choisit un **sommet représentatif** , $f_i^M \in F_i^M$. L'ensemble des sommets représentatifs de M est $V_M = \{f_1^M, \dots f_p^M\}$. Le **quotient** (ou *$k$-structure caractéristique*) de M est $G_M = (V_M, E_M, k)$ où $E_M : {V_M}_*^k \to \mathbb{N}$ est la restriction de E aux k-uplets de V_M. Si G est un graphe, on retrouve la notion de sous-graphe induit : $G_M = G[V_M]$.

Proposition 19
Le quotient d'un module fort `premier` *est une k-structure première (indécomposable).*

Démonstration: Si M correspond à un nœud `premier` de $\mathcal{T}_G$, alors il n'y a aucun module de G compris entre (au sens de l'inclusion) M et F_i^M. Un module de G_M comprenant f_i^M et f_j^M correspondrait dans G à un module contenant F_i^M et F_j^M (sans les chevaucher, car ils sont forts) : c'est alors M. G_M est donc une k-**structure première** : elle n'a pas d'autre module que les triviaux.

□

Cela justifie a posteriori l'identité de nom entre les étiquettes `premier` et les graphes ou k-structures premiers. Des auteurs tels que [ER90b] préfèrent parler de graphes *primitifs*.

Proposition 20
Le quotient d'un module fort `complet` *est une k-structure où E est la fonction constante.*

Démonstration: Si M correspond à un nœud `complet` de $\mathcal{T}_G$, alors toute union $\bigcup_{i \in I} F_i^M$ est un module de G. Rapporté au quotient, toute partie $X \subset V_M$ est un module. Cela est vrai en particulier des parties à deux éléments. Soient e et e' deux k-uplets de G_M. On construit une suite $e_0 = e, \dots e_k = e'$ de k-uplets tels que pour $j > i$ $e_i(j) = e(j)$ et pour $j \leqslant i$ $e_i(j) = e'(j)$. Considérons e_i et e_{i+1}. Si $e(i) = e'(i)$ alors $e_i(i) = e_{i+1}(i)$ donc $E(e_i) = E(e_{i+1})$. Sinon, e_i et e_{i+1} ne diffèrent qu'en leur i$^{\text{e}}$ position : ils sont donc $\{e_i(i), e_{i+1}(i)\}$-équivalents. Or $\{e_i(i), e_{i+1}(i)\}$ est un module. e_i et e_{i+1} le transcendent, sans quoi l'un d'eux serait le k-uplet interdit $(e, \dots e)$. Donc $E(e_i) = E(e_{i+1})$. Cela démontre que $E(e) = E(e')$. La propriété étant vraie pour tout couple, finalement $\exists c \, \forall e \; E(e) = c$.

□

Dans le cas des *graphes* (orientés ou non), le quotient est soit une clique, soit un stable. Cela amène à distinguer *deux* types de nœuds `complet` dans leur arbre de décomposition modulaire : les nœuds `série` sont ceux donc le quotient est une clique, et les nœuds `parallèle` sont ceux

dont le quotient est un stable. Dans le cas des k-structures, on étiquette le nœud par c, où c est la valeur prise par la fonction arête entre membres de ce nœud.

Pour les 2-structures apparaissent les nœuds `linéaire`. Étant données deux couleurs différentes c et c', une 2-structure est un (c, c')-**ordre** si ses sommets peuvent être numérotés de 1 à n tels que $E(x_i, x_j) = c$ si $i < j$ et $E(x_i, x_j) = c'$ si $i > j$.

Proposition 21
Le quotient d'un module fort `linéaire` *est un* (c, c')*-ordre.*

Un nœud `linéaire` sera étiqueté (c, c')- `ordre` et, dans le cas des graphes orientés, juste `ordre`.

2.3.4 Hérédité de la primalité

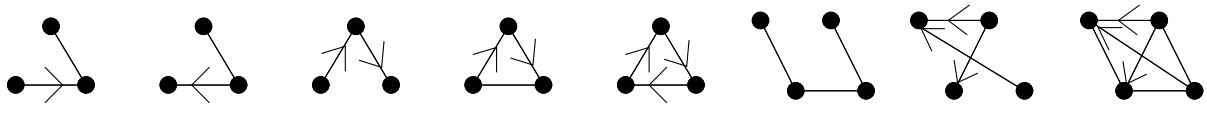
[EM94, Bon94, BM01] étudient les k-structures *premières* (indécomposables). Le résultat suivant est déjà présent dans [ER90a] pour les 2-structures :

Théorème 12 [EM94]
Soit $G = (V, E, k)$ une k-structure première. Il existe $V' \subsetneq V$ tel que $|V \setminus V'| \leqslant k$ et $G[V'] = (V', E \cap V'^k_*, k)$ est première.

Dans le cas des graphes (orientés), à tout graphe premier on peut enlever un ou deux sommets et obtenir un graphe premier.

Une k-structure première est **minimale** si elle ne contient pas de k-structure induite première. Le théorème précédent énonce que les k-structures premières minimales ont au plus $k + 2$ sommets. Elles sont donc en *nombre fini* : $|V^k_*| = n^k - n$. Toute k-structure est donc isomorphe à une k-structure où E est bornée par $n^k - n$. Il y a donc au plus $(n^k - n)^{n^k - n}$ k-structures à n sommets différentes à isomorphisme près, donc moins de $((k + 2)^k - (k + 2))^{(k+2)^k - (k+2)}$ k-structures minimales. Pour $k = 2$ cela fait tout de même de l'ordre de 10^{13}, de quoi décourager une énumération manuelle. Mais en se restreignant au cas des graphes, ce n'est finalement pas si terrible puisque l'on retrouve le résultat classique :

Proposition 22
le seul *graphe premier minimal est P_4. . Les graphes orientés premiers minimaux sont :*



2.4 Décomposition en comités des hypergraphes

[Bil71, CE80, CHM81] utilisent une définition différente des modules, appelés alors *comités*[18], pour le cas des hypergraphes :

Définition 11
*$M \subset V$ est un **comité** si pour toutes hyperarêtes $A, B \in E$ intersectant M on a $(A \cap M) \cup (B \setminus M) \in E$.*

Un hypergraphe est **simple** si pour deux hyperarêtes $A, B \in E$ $A \not\subset B$. Prenons l'hypergraphe simple $G = (\{1,2,3,4\}, \{\{1,2\},\{2,3,4\}\})$. $\{3,4\}$ est un comité, puisqu'une seule hyperarête l'intersecte. Voyons G comme une 3-structure où E vaut 1 pour tous les couples de support $\{1,2\}$ ou $\{2,3,4\}$ et 0 partout ailleurs. $(2,3,3)$ et $(2,3,4)$ sont $\{3,4\}$-équivalents et transcendent $\{3,4\}$, mais ils ne sont pas colorés de la même couleurs par E, donc $\{3,4\}$ n'est pas un module de G.

Réciproquement, considérons la 3-structure complète $G = (\{1,2,3,4\}, \{1,2,3,4\}^3_*)$. $\{3,4\}$ est un module de G. Si G est vu comme un hypergraphe simple, les deux hyperarêtes $\{1,2,3\}$ et $\{2,3,4\}$ appartiennent à E et intersectent $\{3,4\}$, mais $\{1,2,3,4\} \notin E$: $\{3,4\}$ n'est donc pas un comité de G. Il s'agit donc d'une décomposition différente.

Théorème 13 [CHM81, CE80, Bil71]
La famille des comités d'un hypergraphe simple est partitive.

[BDV95, BDV99] proposent un algorithme pour la décomposition en comités (et non la décomposition modulaire comme définie dans ce mémoire). Leur algorithme est exponentiel en la taille maximale k des hyperarêtes, et en $O(n^{3k-5})$ si $k \geqslant 3$ est la dimension de l'hypergraphe. La décomposition des hypergraphes en comités, en temps polynômial, semble donc un problème ouvert dans le cas général.

2.5 Décomposition en bloc des graphes d'héritage

Définition 12
*Un **graphe d'héritage** est un graphe orienté, sans circuit, possédant une unique source*[19] *notée 0_G et un unique puits noté 1_G.*

Cette notion apparaît dans [Huc92, HHS95, Cap97b, Cap97a] dans le cadre des langages à objets : le graphe est la hiérarchie d'héritage *explicite* (les seuls arcs de transitivité sont volontaires) d'un langage, 1_G la classe usuellement appelée `Object` dont héritent toutes les autres et O_G est la classe *absurde*, ajoutée par commodité, qui spécialise toute classe.

18. pauvre traduction de *committee*.
19. source : sommet dont le voisinage entrant est vide.
puits : sommet dont le voisinage sortant est vide.

Les notions de **h-module** et **bloc** sont voisines : dans les deux cas il s'agit d'une partie $M \subset V$ telle que $G[M]$ soit un graphe d'héritage. Cela permet de définir la source O_M et le puits 1_M du bloc. M est un h-module si pour tout $x \in M$ et tout $y \notin M$ tout chemin entre x et y passe par 0_M ou 1_M. M est un bloc s'il vérifie cet axiome pour tout $x \neq 0_M$ et $x \neq 1_M$. Un h-module est donc un bloc. On peut aussi dire que M est un bloc si $G[M]$ possède une unique source et un unique puits et que seuls ces deux sommets peuvent avoir des adjacences dans l'extérieur du bloc.

Proposition 23
[Huc92, HHS95] Si M est un h-module de G alors c'est un module de la fermeture transitive de G. La réciproque est vraie si la fermeture transitive de G est un treillis.

On en déduit :

Proposition 24
La famille des h-modules d'un graphe couvrant un treillis est faiblement partitive.

Proposition 25
[Cap97b] M est un bloc de G si et seulement si, dans la fermeture transitive du line-graph[20] *de G, le sous-graphe induit par les arêtes internes de M est un module.*

On en déduit :

Proposition 26
La famille $\mathcal{F}$ des parties de $E(G)$, où G est un graphe d'héritage et où toute partie A est l'ensemble des arêtes internes d'un bloc, est faiblement partitive.

Capelle caractérise les quotient des nœuds `complet` (antichaînes) et `linéaire` (chaînes). Les sommets d'un h-module donné sont un facteur de toute extension linéaire obtenue par un parcours en largeur du graphe d'héritage [DH89]. [Cap97b] en déduit un algorithme de décomposition, pour les h-modules et pour les blocs, en temps $O(n + m)$.

20. graphe sont les sommets sont les arêtes de G. Il y a adjacence entre deux sommets du line-graph si les arêtes correspondantes de G ont une extrémité commune.

Chapitre 3

Décompositions 2-modulaires

Duos habet et bene pendentes

extrait du rituel latin d'intronisation pontificale

LA DÉCOMPOSITION modulaire est certes un puissant outil, qui permet de mieux se représenter la structure d'un graphe, et de résoudre récursivement des problèmes. Mais peu de graphes ont des modules. Il convient donc de se pencher sur des décompositions « plus puissantes »[1]. Ce chapitre présente une généralisation des modules : les *2-modules*.

La notion apparaît dans [CS87] sous le nom de « *homogeneous pair* » ; son étude y est motivée par la conjecture de Berge sur les graphes parfaits. Le seul article à ma connaissance qui y soit consacré est [EKR97], qui fournit un algorithme en $O(mn^3)$ exhibant un 2-module non-trivial s'il en existe un. Ce même article décrit comment construire une certaine formule 2-SAT pour chaque triplet de sommet. Elles permettent d'obtenir, en examinant toutes les assignations valides, tous les 2-modules du graphe, en temps polynômial en la taille de la sortie.

La démarche suivie dans cette étude est la suivante. §3.1 présente les 2-modules, et §3.2 les premiers résultats : quelles propriétés des modules sont conservées, et lesquelles manquent, donc les faiblesses des 2-modules. §3.3 donne le théorème central, établissant que la famille des 2-modules n'est pas partitive, mais *presque*.

On s'intéresse alors à des sous-familles qui, elles, sont partitives ou bipartitives. D'abord (§3.4) les *sesquimodules*[2], intermédiaires en quelque sorte entre modules et 2-modules. Ensuite, les 2-modules parfaits. Un 2-module est *parfait* si son complémentaire est un 2-module. En §3.5 ils sont caractérisés et découpés en quatre familles : modules, coupes, co-coupes et 2-joints.

1. Une façon simple de définir qu'une décomposition B est plus puissante qu'une décomposition A est que tout ensemble de décomposition de A est un ensemble de décomposition de B. Pour éviter les cas triviaux, on se restreint aux ensembles *forts*. B permet d'aller "plus loin" que A car des graphes *premiers*, donc indécomposables par A, peuvent avoir une décomposition non-triviale par B.

2. du latin *sesquis*, un-et-demi.

Celle des 2-joints étant nouvelle, elle est étudiée plus en détails §3.6. La section suivante étudie les incompatibilités entre ces familles : coupes, co-coupes et 2-joints étendent la décomposition modulaire de façon *presque* mutuellement exclusive, donc on obtient pour un graphe donné non pas quatre arbres mais un seul, plus riche d'informations que la décomposition modulaire.

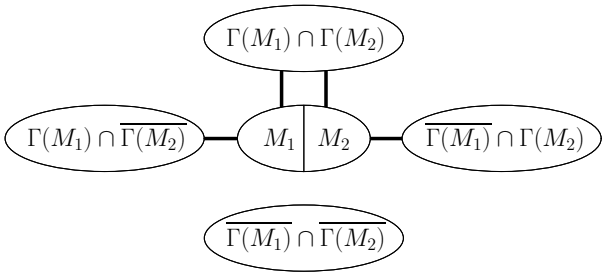
Malheureusement, tout 2-module d'un graphe n'est pas dans une de ces familles. La conclusion contient deux constatations pessimistes : tout d'abord, puisque *les 2-modules ne passent pas au quotient* (voir §3.2) l'approche « arbre de décomposition » semble inadéquate pour le cas général. Et aussi que le théorème central n'a pas lieu d'être pour les décompositions k-modulaires. Mais présentons tout d'abord l'objet de ce chapitre, le 2-module.

3.1 Les 2-modules

3.1.1 Définition

Définition 13 ([CS87])
*Un **2-module** ou **paire homogène** de $G = (V, E)$ est une paire de parties disjointes de V, notée $M = \{M_1, M_2\}$, telle que*

- $\forall x, y \in M_1 \; \forall z \notin (M_1 \cup M_2) \;\; (x, z) \in E \iff (y, z) \in E$
- $\forall x, y \in M_2 \; \forall z \notin (M_1 \cup M_2) \;\; (x, z) \in E \iff (y, z) \in E$



2-module.

Comme beaucoup d'objets, les 2-modules ont étés introduits pour l'étude des graphes parfaits : [CS87] prouve qu'un graphe minimal imparfait ne contient pas de 2-module non-trivial [3].

Une première remarque est qu'un graphe comporte beaucoup de 2-modules *triviaux*, c'est-à-dire des 2-modules qui existent quelle que soit la structure du graphe.

Théorème et définition 14
Soit $G = (V, E)$ un graphe, $u, v \in V$, et $\{V_1, V_2\}$ une bipartition de V.

1. $\{\emptyset, \emptyset\}$
2. $\{\emptyset, \{u\}\}$
3. $\{\{u\}, \{v\}\}$

3. Depuis que Chudnovsky, Robertson, Seymour et Thomas [CRST] ont prouvé la conjecture de Berge, on sait que les graphes minimaux imparfaits sont les trous C_{2n+5} et les anti-trous $\overline{C_{2n+5}}$.

4. $\{\Gamma(u), \overline{\Gamma(u)}\}$
5. $\{V_1, V_2\}$
6. $\{\emptyset, V\}$

sont des 2-modules, que nous appellerons ***2-modules triviaux****.*

Notons que tous les graphes n'ont pas les mêmes 2-modules triviaux, à cause de la définition de $\{\Gamma(u), \overline{\Gamma(u)}\}$ comme trivial.

Proposition 27
Soit $M = \{M_1, M_2\}$ un 2-module d'un graphe 1-premier[4]*. Si $M_1 \cup M_2 \neq V$, il n'y a qu'une façon de partitionner $M_1 \cup M_2$ en $\{M_1', M_2'\}$ telle que $\{M_1', M_2'\}$ soit un 2-module.*

Démonstration: Considérons les deux bipartitions $M = M_1 \uplus M_2$ et $M = M_1' \uplus M_2'$, et supposons $M_1 \neq M_1'$ et $M_1 \neq M_2'$. Il existe alors $i = 1$ ou 2 tel que $M_i \cap M_1' \neq \emptyset$ et $M_i \cap M_2' \neq \emptyset$. Soit $x \notin M$. x ne distingue pas M_1'. $M_i \cap M_1' \neq \emptyset$ entraîne que x ne distingue pas $M_i \cup M_1'$. $M_i \cap M_2' \neq \emptyset$ entraîne que x ne distingue pas M, qui est donc un 1-module, donc $M = V$.

□

Cette proposition autorise donc à complètement *confondre* le 2-module $M = \{M_1, M_2\}$ et la partie $M = M_1 \cup M_2$ de V, puisqu'il n'y a pas d'ambiguïté. Ce lemme est la vraie raison pour laquelle, dans toute la suite, nous travaillerons dans le cas G 1-premier : tout 2-module, sauf V lui-même, peut alors être vu comme *une* partie de V et non *deux* parties de V, ce qui est bien plus simple à manipuler. Dès que le besoin s'en fait sentir, cette partie est canoniquement coupée en $\{M_1, M_2\}$.

Puisque la décomposition 2-modulaire se veut une généralisation de la décomposition 1-modulaire, se restreindre aux graphes indécomposables par cette dernière ne me semble pas choquant. Et les preuves, déjà compliquées dans le cas 1-premier, risqueraient d'être bien pires dans le cas général !

Deux modules $M = \{M_1, M_2\}$ et $M' = \{M_1', M_2'\}$ se chevauchent si $M_1 \cup M_2$ chevauche $M_1' \cup M_2'$. Étant donné un 2-module $M = \{M_1, M_2\}$, $m \in M$ est un sommet M-**homogène** si m ne distingue ni $M_1 \setminus \{m\}$ ni $M_2 \setminus \{m\}$.

3.2 Premières propriétés

Certaines propriétés des 2-modules ressemblent à celles des modules :

Proposition 28
$\{M_1, M_2\}$ est un 2-module de G si et seulement si c'est un 2-module de $\overline{G}$.

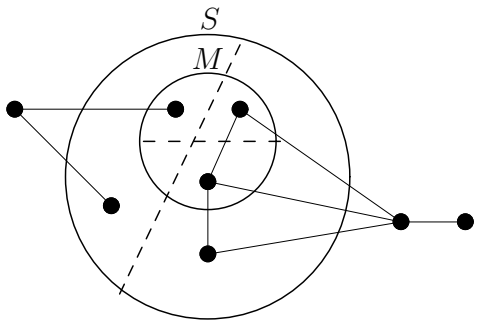
4. Pour éviter les ambiguïtés, on parlera dans ce chapitre et le suivant de « 1-module » au lieu de « module », et un graphe 1-premier est sans 1-module.

Démonstration: trivial. □

Proposition 29
Étant donné un graphe $G = (V, E)$, un 2-module $\{M_1, M_2\}$ de G et une partie $S \subset V$, $\{M_1 \cap S, M_2 \cap S\}$ est un 2-module de $G[S]$.

Démonstration: trivial. □

En revanche, d'autres propriétés des 1-modules ne sont plus vraies. La première est que pour $M \subset S \subset V$, si S est un 2-module de G et M un 2-module de $G[S]$ alors M peut *ne pas être* un 2-module de G.



Contre-exemple : M est un 2-module de $G[S]$ mais pas de G, alors que S est un 2-module.

Comme nous en discuterons §3.9.1, ce fait que les 2-modules ne *passent pas au quotient* est une vraie catastrophe. On a juste :

Proposition 30
Soit $G = (V, E)$ un graphe, $\{S_1, S_2\}$ un 2-module de G et $\{M_1, M_2\}$ un 2-module de $G[S_1 \cup S_2]$. Si ni M_1 ni M_2 ne chevauchent S_1 ou S_2, alors $\{M_1, M_2\}$ est un 2-module de G.

Démonstration: Supposons que $\{M_1, M_2\}$ ne soit pas un 2-module de G. Il existe alors $x \in V$ tel que $\Gamma(x)$ chevauche M_1 ou M_2 : x *distingue* $\{M_1, M_2\}$. Or $\{M_1, M_2\}$ est un 2-module de $G[S_1 \cup S_2]$. Donc $x \notin (S_1 \cup S_2)$. Puisque $\{S_1, S_2\}$ est un 2-module de G, ni S_1 ni S_2 ne chevauchent $\Gamma(x)$. Comme $M_1 \subset (S_1 \cup S_2)$, si $\Gamma(x)$ chevauche M_1, M_1 chevauche S_1 et S_2. Même preuve pour M_2.

□

Pour les familles de 2-modules où cette propriété de non-chevauchement de la bipartition par un 2-module inclus est respectée, une approche *récursive*, en recherchant les 2-modules maximaux $S_1 \dots S_k$, et en itérant, permet de construire un arbre qui énumère toute la famille. C'est le cas des 1-modules, et des bimodules qui seront étudiés au chapitre suivant. Mais dans le cas général cela ne marche pas : une telle approche crée des *artefacts*, des 2-modules de $G[S]$ mais non de G, vides de sens.

Une autre propriété non-conservée est l'inclusion des graphes premiers (hérédité de la primalité ; voir §2.3.4). Prenons un contre-exemple : le graphe C_n, $n \geqslant 7$.

– Il n'a aucun 2-module non-trivial. Car, en numérotant de 1 à n (modulo n) ses sommets,

$\{i, i+1\}$ est séparé par $i-1$ et $i+1$, $\{i, i+2\}$ est séparé par $i-1$ et $i+3$, et $\{i, i+k\}$, $k \geqslant 3$ est séparé par $i+1$ et $i+k-1$.

- Tout sous-graphe induit de C_n est une union de P_m disjoints.
- Toute composante connexe à plus de 2 et moins de $n-1$ sommets est un 2-module non-trivial.
- P_m, $m \geqslant 5$, possède un 2-module non-trivial $\{\{1,2\},\{3\}\}$.
- $P_4 \uplus P_1$ possède un 2-module non-trivial.

Donc *aucun sous-graphe induit à plus de quatre sommets de C_n n'est indécomposable, alors que C_n est indécomposable*.

Enfin, une troisième différence importante avec les 1-modules est que si deux 2-modules se chevauchent alors leur union, leur intersection on leur différence n'est pas nécessairement un 2-module. Ce point est étudié dans la section qui vient.

3.3 Le théorème de décomposition 2-modulaire

Le théorème suivant aurait pu prouver que la famille des 2-modules d'un graphe est partitive. Il n'en est rien. Cependant, de façon surprenante, cette famille est *presque* faiblement partitive : des quatre points requis par la définition d'une famille faiblement partitive, deux 2-modules se chevauchant en vérifient au moins deux, et au moins trois s'ils ne sont pas des 2-modules triviaux. Deux 2-modules qui se chevauchent sans que les quatre points requis soient valides sont dits en **conflit**

Si les conflits n'existaient pas, la famille des 2-modules non-triviaux d'un graphe serait faiblement partitive. Pourquoi n'est-ce pas le cas ? Une première réponse est donnée quand on regarde le comportement des modules triviaux de la forme $\{\{u\},\{v\}\}$. Pour toute partie non-triviale $S \subset V$, on peut trouver un tel 2-module qui la chevauche. Demander qu'union et intersection de 2-modules soient des 2-modules reviendrait à dire que à tout 2-module on peut ajouter ou supprimer un sommet *arbitrairement choisi* et obtenir un 2-module, ce qui revient à dire que, si la famille des 2-modules d'un graphe est non-vide, elle serait égale à 2^V. Il faut donc s'abstenir de considérer les 2-modules triviaux. On pourrait considérer la famille des 2-modules privée des triviaux. Mais cette famille n'est pas partitive : on risquerait de tomber sur des cas où l'intersection ou bien la différence de deux membres serait un 2-modules trivial ; et la famille n'est plus *close* par ces opérations. Le théorème suivant caractérise complètement les cas de conflits des graphes 1-premiers, et montre à quel point ils ont une forme particulière.

Théorème 14
*Soit G un graphe 1-premier et M, M' deux 2-modules de G qui se chevauchent. Si M et M' ne sont pas triviaux, au au moins **trois sur quatre** des propositions suivantes sont vraies. Et si l'un est trivial, au moins **deux sur quatre** des propositions suivantes sont vraies :*

(1) $M \cup M'$ est un 2-module.

(2) $M \cap M'$ est un 2-module.

(3) $M \setminus M'$ est un 2-module.

(4) $M' \setminus M$ est un 2-module.

De plus, si

- *$M \cup M'$ n'est pas un 2-module, alors $M \cap M' = \{m\}$.*
- *$M \cap M'$ n'est pas un 2-module, alors $M \cup M' = V$. M et M' sont parfaits ou triviaux.*
- *$M \setminus M'$ n'est pas un 2-module, alors $M' \setminus M = \{m\}$.*

La preuve de ce théorème est l'objet de la section 3.8. Il s'agit d'une longue étude par cas : un 2-module est la donnée de deux ensembles, et deux ensembles donnés peuvent entretenir cinq types de relations différents (intersection vide, égalité, deux inclusions strictes, chevauchement), que les symétries factorisent quand même beaucoup.

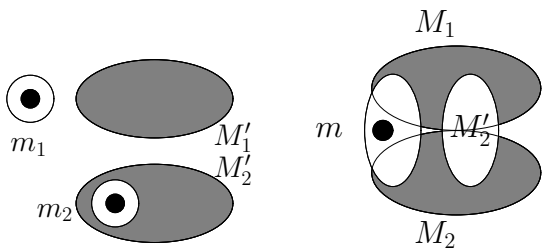
La liste des configurations où ces quatre points ne sont pas simultanément vérifiés est donnée ci-après (aux configurations symétriques près). Dans tous ces dessins, M est en blanc, M' en grisé, chaque partie dessinée est non-vide, et celles où un sommet est dessiné ne contiennent que ce sommet. Selon le cas, on parlera de

- Conflit **mineur** si un des 2-modules impliqué est trivial
- Conflit d'**union** si la propriété (1) est violée.
- Conflit d'**intersection** si la propriété (2) est violée.
- Conflit de **différence** si la propriété (3) ou (4) est violée.

3.3.1 Conflits mineurs

L'un des 2-modules impliqué est trivial.

- **Conflit mineur I**
  $M = \{\{m_1\}, \{m_2\}\}$, $m_1 \notin M'$, $m_2 \in M'$. $M' \setminus M$ et $M \cup M'$ ne sont pas nécessairement des 2-modules, mais $M \setminus M' = \{m_1\}$ et $M \cap M' = \{m_2\}$ si.
- **Conflit mineur II**
  $M = \{\Gamma(m), \overline{\Gamma(m)}\}$, $m \in M'_1$. $M \setminus M'$ et $M \cap M'$ ne sont pas nécessairement des 2-modules, mais $M' \setminus M = \{m\}$ et $M \cup M' = V$ si.



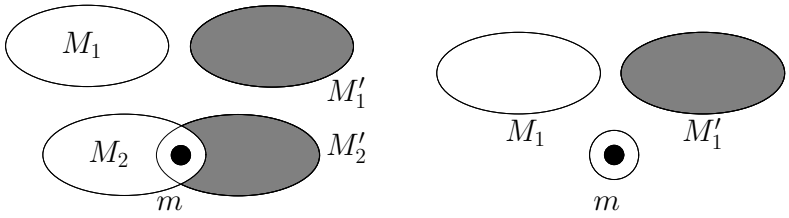
mineur I *mineur II*

3.3.2 Conflits d'union

$M \cup M'$ n'est pas un 2-module. On a $M \cap M' = \{m\}$.

– **Conflit d'union I**
$M_2 \cap M'_2 = \{m\}$; $M_2 \circledcirc M'_2$. m est M-homogène et M'-homogène.

– **Conflit d'union II**
$M_2 = M'_2 = \{m\}$. M et M' sont boiteux[5].



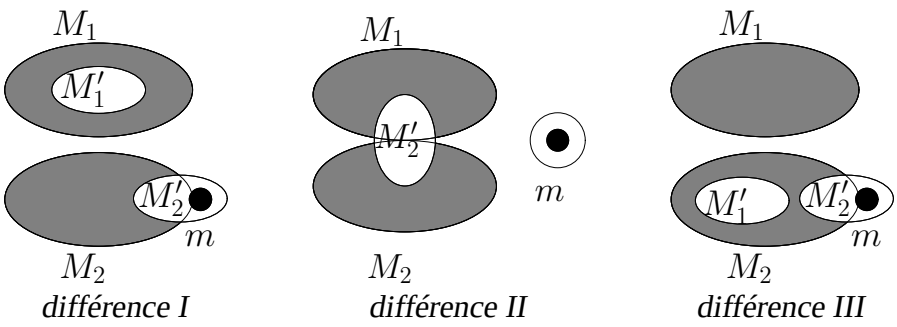
union I *union II*

3.3.3 Conflits de différence

$M \smallsetminus M'$ n'est pas un 2-module. On a $M' \smallsetminus M = \{m\}$.

– **Conflit de différence I**
$M'_1 \subset M_1$ et $M'_2 \smallsetminus \{m\} \subset M_2$. m est M-homogène et M'-homogène.
– **Conflit de différence II**
$M'_1 = \{m\}$, $m \notin M$. $M'_2 \subset M$. M' est boiteux.
– **Conflit de différence III**
$M' \smallsetminus \{m\} \subset M_2$. m est M-homogène et M'-homogène.

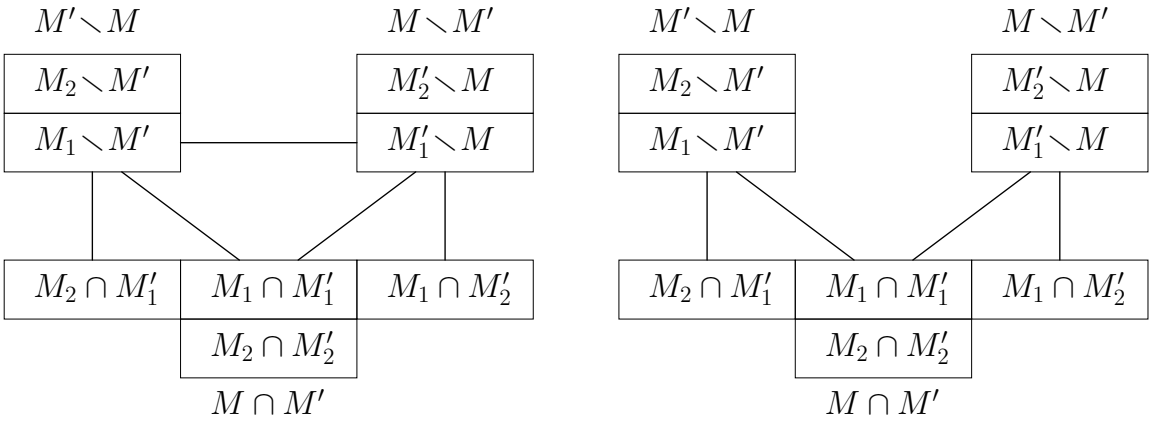
5. Un 2-module $\{M_1, M_2\}$ est boiteux si M_1 ou M_2 est un singleton, voir §3.4



3.3.4 Conflits d'intersection

$M \cap M'$ n'est pas un 2-module. $M \cup M' = V$, et M et M' sont parfaits. Ils correspondent à des **coupes**, des **co-coupes** ou des **2-joints**[6].

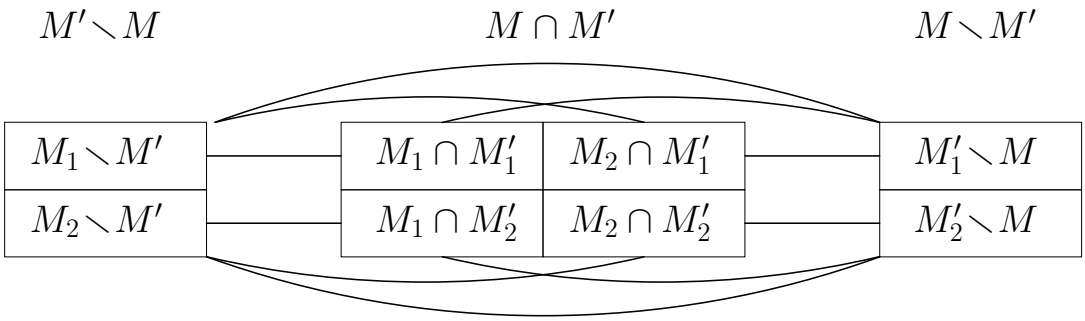
– **Cas où M et M' sont des coupes**
Au plus un des quatre ensembles $M_i \cap M'_j$ peut être vide.



– **Cas où M et M' sont des co-coupes**
Ce sont exactement les mêmes pris dans $\overline{G}$.

– **Cas où M et M' sont des 2-joints**
Au plus un des quatre ensembles $M_i \cap M'_j$ peut être vide.

6. voir les définitions 17 page 72 et 18 page 73.



3.4 Décomposition en sesquimodules. 2-modules boiteux.

Puisque la famille des 2-modules n'est pas partitive, nous allons nous intéresser dans le reste de ce chapitre à des sous-familles possédant cette propriété (ou une variante).

Définition 15
Soit $G = (V, E)$ un graphe 1-premier et $q \in V$. M est un q-sesquimodule de G si M est un module de $G[V \setminus \{q\}]$.

On notera $\mathcal{S}_q$ la famille de tous les q-sesquimodules, et $\mathcal{S}_q^*$ la famille des q-sesquimodules non-triviaux (ceux de cardinal strictement compris entre 1 et n).

Il est immédiat que $M \in \mathcal{S}_q$ est le 2-module $\{M \cap \Gamma(q), M \cap \overline{\Gamma}(q)\}$. Puisque on a établi au chapitre 1 que la décomposition modulaire d'un graphe est partitive, on en déduit immédiatement

Proposition 31
$\mathcal{S}_q$ est partitive.

Nous venons donc d'extraire une petite famille partitive de la grande famille des 2-modules. Ces 2-modules ressemblent beaucoup à des 1-modules, puisqu'ils sont effectivement 1-modules de $G - q$, ce qui justifie son nom de sesquimodule : 1.5-module.

On vérifie facilement que si M non-trivial est un p-sesquimodule et un q-sesquimodule, alors $p = q$. Donc

Proposition 32
Si $p \neq q$ alors $\mathcal{S}_p^* \cap \mathcal{S}_q^* = \emptyset$.

Nous voici donc avec n sous-familles différentes de 2-modules, toutes partitives ! Notons que l'arbre de décomposition q-sesquimodulaire peut être assez fourni : en effet, en prenant un graphe G avec un arbre de décomposition modulaire aussi compliquée que l'on veut, mais tel que les nœuds `complets` aient au plus deux fils, on peut faire un graphe premier en lui rajoutant un sommet q adjacent à exactement une feuille par nœud de l'arbre. L'arbre de décomposition

modulaire de G devient bien sûr l'arbre de décomposition q-sesquimodulaire de ce nouveau graphe.

Définition 16
Soit $G = (V, E)$ un graphe 1-premier et $M = \{M_1, M_2\}$ un 2-module de G. G est boiteux si $|M_1| = 1$ ou $|M_2| = 1$, et si $|M_1 \cup M_2| \geqslant 3$.

Pour q fixé, on notera $\mathcal{B}_q$ la famille de tous les 2-modules boiteux $\{\{q\}, M\}$.

Soit $\{\{q\}, M\}$ un 2-module boiteux. Tous les séparateurs de M sont dans $\{q\}$, donc M est un q-sesquimodule. Réciproquement, il est clair que si M est un q-sesquimodule alors $\{\{q\}, M\}$ est boiteux. Donc

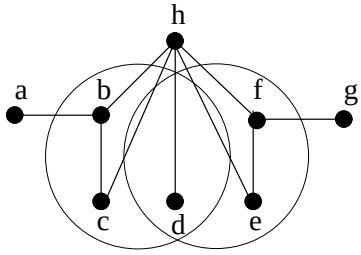
Proposition 33
$\mathcal{S}_q^*$ est en correspondance bijective avec $\mathcal{B}_q$.

Ce résultat est intéressant. $\mathcal{B}_q$ n'est pas du tout partitive : ses membres contiennent tous q, leur intersection n'est donc jamais vide et cela crée quantité de « faux chevauchements ». Cependant, puisqu'elle est en bijection avec les q-sesquimodules, qui eux sont une famille partitive, $\mathcal{B}_q$ possède une *structure partitive*. En particulier, l'arbre partitif de $\mathcal{S}_q$ fournit un moyen simple et de taille linéaire de représenter $\mathcal{B}_q$, famille de taille potentiellement exponentielle.

Donc, en plus des n familles partitives de sesquimodules, nous extrayons de la famille des 2-modules n nouvelles familles à structure partitive !

Notons qu'un 2-module peut être à la fois boiteux et sesquimodule : ainsi dans le graphe P_n, en numérotant les sommets de 1 à n, $\{\{i\}, \{i+1, \ldots n\}\}$ est boiteux, mais est aussi un $(i-1)$-sesquimodule.

Il n'est par contre pas plus intéressant de s'occuper de la famille des sesquimodules en général, $\mathcal{S} = \bigcup_q \mathcal{S}_q$. En effet, l'union de deux sesquimodules qui se chevauchent peut ne pas être un sesquimodule, comme le montre l'exemple suivant. Il y a donc des *conflits*, comme dans le cas des 2-modules en général.



Le a-sesquimodule $\{b, c, d\}$ chevauche le g-sesquimodule $\{d, e, f\}$ mais leur union n'est pas un 2-module.

3.5 Les décompositions 2-modulaires parfaites

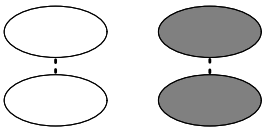
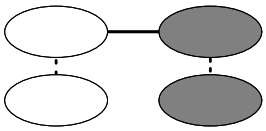
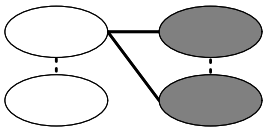
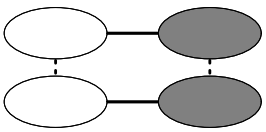
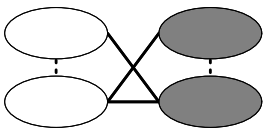
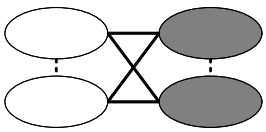
Un 2-module M est **parfait** si son complémentaire $V \setminus M$ est un 2-module (lui aussi parfait). Cette section caractérise les 2-modules parfaits des graphes (pas seulement 1-premiers). Définis-

sons rapidement quelques objets, qui seront vus plus en détail dans les sections suivantes. Un 2-module $M = \{M_1, M_2\}$ est le *côté d'une coupe* si pour tout $x \in M_1$ $\Gamma(x) \subset M$. Il est le *côté d'une co-coupe* s'il est le côté d'une coupe dans $\overline{G}$, et il est le *côté d'un 2-joint* si tout $m \notin M$ est adjacent soit à M_1, soit à M_2 (mais non aux deux). On vérifie facilement que ces « côtés » sont des 2-modules parfaits (voir le tableau dans le théorème suivant). Réciproquement on a :

Théorème 15
Soit M un 2-module parfait et $M' = V \setminus M$.

- *Soit M ou M' est un 1-module ;*
- *soit M est le côté d'une coupe ;*
- *soit M est le côté d'une co-coupe ;*
- *soit M est le côté d'un 2-joint.*

Démonstration: Le tableau suivant énumère les cas possibles. Les deux ovales blancs sont M_1 et M_2, les grisés sont M'_1 et M'_2. Il n'y a que 16 cas à considérer, selon les adjacences possibles entre ovales blancs et grisés. Les cas symétriques par échange de M et de M', de M_1 et de M_2, ou de M'_1 et de M'_2 sont regroupés.

1  M et M' sont des **1-modules**	2  G possède une **coupe**	3  M' est un **1-module**
4  G possède un **2-joint**	5  G possède une **co-coupe**	6  M et M' sont des **1-modules**

□

3.5.1 Les 1-modules

Parmi les quatre types de décompositions 2-modulaires parfaites, un est déjà connu du lecteur : c'est le cas où un *1-module* apparaît. Son complémentaire est alors soit un autre 1-module (cas 1 et 6) ; soit un 2-module (cas 3). Nous savons comment traiter ce cas : la section 2.3 page 53 décrit comment l'*arbre de décomposition modulaire* de G peut décrire tous les modules de G.

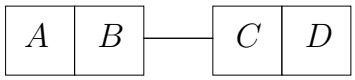
Il donne donc, par la même occasion, tous 2-modules qui sont complémentaires d'un module, « gratuitement ».

3.5.2 Les coupes et co-coupes

Les cas 2, 4 et 5 correspondent à des objets que nous n'avons pas encore vus : les *2-joints*, les *coupes*, et leurs *alter ego* les *co-coupes*. Les 2-joints sont des objets nouveaux, qui seront étudiés en §3.6. Les deux autres sont déjà connus. Une co-coupe de G est tout simplement une coupe de $\overline{G}$. La décomposition en coupe a été étudiée par [Cun82].

Définition 17
*Soit $G = (V, E)$ un graphe non-orienté sans boucles. G admet une **coupe** (anglais : split) si V peut être partitionné (au sens large : une partie peut être vide) en quatre ensembles A, B, C, D tels que*

- *$A \cup B$ est non vide*
- *$C \cup D$ est non vide*
- *A est non-adjacent à C et à D*
- *D est non-adjacent à A et à B*
- *C est adjacent à D*



La coupe (A, B, C, D)

$\{A, B\}$ et $\{C, D\}$ sont deux 2-modules parfaits. La famille des coupes d'un graphe est $\{\{A \cup B, C \cup D\} \mid (A, B, C, D)$ est une coupe $\}$.

Théorème 16 ***[Cun82]***
La famille des coupes d'un graphe est bipartitive.

McConnell et Spinrad [MS94a] donnent un algorithme en $O(n^2)$ permettant de la calculer. Dahlhaus [Dah00] a écrit un algorithme parallèle, et séquentiel en $O(n + m)$. Lanlignel [Lan01] est une excellente étude francophone sur le sujet.

La décomposition en coupes est un point fondamental pour la définition ou la reconnaissance de plusieurs classes de graphes :

- graphes d'intersection des cordes d'un cercle [Spi94],
- graphes paritaires[7], dont [CS99b] établit que les quotients sont des cliques ou des graphes bipartis,
- diverses classes entre bipartis et paritaires [CS99a].

[Wag90] l'étend aux relations k-aires (famille de k-uplets d'un ensemble fini).

7. *parity graphs* : pour tout couple de sommets, les longueurs de tous les chemins sans cordes les reliant sont congrues modulo 2. Les graphes distance-héréditaires en sont un cas particulier.

3.6 Décomposition en 2-joints

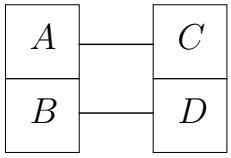
La notion de *2-joint* est utilisée par la communauté des gens travaillant sur les graphes parfaits et la preuve de la conjecture forte de Berge, en tant qu'opération de composition préservant la perfection. Nous l'utiliserons ici avec une définition différente, plus restrictive. La raison en est qu'il fallait que les deux côtés d'un 2-joint soient des 2-modules, alors qu'au sens habituel [CC85] ce sont des 3-modules particuliers.

Les 2-joints où l'un des côtés est réduit à deux sommets sont appelés *antijumeaux* [Ola88] et ont aussi été étudiés dans le cas de la preuve du théorème de Berge.

3.6.1 Définition d'un 2-joint

Définition 18
*Soit $G = (V, E)$ un graphe non-orienté sans boucles. G admet un **2-joint** si V peut être partitionné en quatre ensembles A, B, C, D tels que*

- *$A \cup B$ est non-vide*
- *$C \cup D$ est non-vide*
- *A est adjacent à C*
- *A est non-adjacent à D*
- *B est non-adjacent à C*
- *B est adjacent à D*



Le 2-joint (A, B, C, D)

Si G possède une telle décomposition, nous écrirons que G possède un 2-joint, noté $J = (A, B, C, D)$. (A, B, C, D) et (C, D, A, B) représentent le même 2-joint.

Le graphe[8] $2K_n$ possède $\Theta(2^{2n})$ 2-joints : toute bipartition de la première clique entre A et C, toute bipartition de la deuxième entre B et D, produit un 2-joint. Ce n'est pas étonnant, car les 2-joints généralisent les modules, qui existent déjà en quantité exponentielle. Un module en effet est un 2-joint trivial $(M, \emptyset, \Gamma(M), \overline{\Gamma(M)})$. Réciproquement, tout 2-joint où l'un des quatre ensembles est vide correspond à un module.

Une autre remarque à faire est que, tout comme la décomposition modulaire, cette décomposition est invariante par passage au complémentaire. Plus exactement, si (A, B, C, D) est un 2-joint de G, alors (A, B, D, C) est un 2-joint de $\overline{G}$.

8. Un préfixe numérique devant un nom de graphe connu, comme kG, est un graphe composé de k copies côte-à-côte de G, sans arêtes entre elles. $2K_n$ est donc une composition parallèle de deux graphes complets à n sommets chacun.

Or, on peut noter que connaître $A \cup B$ et $C \cup D$ permet, sans ambiguïté, de retrouver A, B, C et D. C'est pourquoi il est légitime de noter $J = \{J^0, J^1\}$ un 2-joint, avec $J^0 = A \cup B$ et $J^1 = C \cup D$. J^0 et J^1 sont des 2-modules de G, formant une bipartition de G.

Vu sous cet angle, un 2-joint de G est aussi un 2-joint de $\overline{G}$. De plus, puisqu'ils forment une bipartition de G, tout l'arsenal théorique développé au chapitre 1 page 23 va servir : il n'y aura plus qu'à prouver que la famille de 2-joints d'un graphe est bipartitive (ce qui est l'objet de la section suivante) pour en tirer des quantités d'informations sur la structure des 2-joints d'un graphe.

3.6.2 La famille des 2-joints est bipartitive

Soit $\mathcal{J}$ la famille des 2-joints d'un graphe G. Le but de cette section est de prouver :

Théorème 17
$\mathcal{J}$ est une famille bipartitive.

Démonstration: Soient $J_1 = \{J_1^0, J_1^1\} = (A_1, B_1, C_1, D_1)$ et $J_2 = \{J_2^0, J_2^1\} = (A_2, B_2, C_2, D_2)$ deux 2-joints de G. Suivant la définition des familles bipartitives, il faut démontrer que, si J_1 chevauche J_2, donc si J_1^0 chevauche J_2^0 et J_2^1, alors les cinq bipartitions suivantes sont des 2-joints :

- $J_3 = \{J_1^0 \cap J_2^0, J_1^1 \cup J_2^1\}$
- $J_4 = \{J_1^0 \cap J_2^1, J_1^1 \cup J_2^0\}$
- $J_5 = \{J_1^1 \cap J_2^0, J_1^0 \cup J_2^1\}$
- $J_6 = \{J_1^1 \cap J_2^1, J_1^0 \cup J_2^0\}$
- et $J_7 = \{J_1^0 \Delta J_2^0, J_1^0 \Delta J_2^1\}$

Supposons, sans perte de généralité, que A_1 intersecte A_2 et B_2.

- Soit $v \in C_2$. Le fait que C_2 est adjacent à A_2 et non-adjacent à B_2, et le fait que A_1 rencontre A_2 et B_2, implique que v n'est ni adjacent ni non-adjacent à A_1, et appartient donc à $A_1 \cup B_1$.
- Soit $v \in D_2$. Le fait que D_2 est non-adjacent à A_2 et adjacent à B_2, et le fait que A_1 rencontre A_2 et B_2, fait que v n'est ni adjacent ni non-adjacent à A_1, et appartient donc à $A_1 \cup B_1$.

Donc $(C_2 \cup D_2) \subset (A_1 \cup B_1)$. Il n'y a donc pas chevauchement entre J_1 et J_2.

De la même façon, on démontre que A_1 ne peut intersecter C_2 et D_2 simultanément. Donc l'un de ces quatre cas seulement est possible :

1. $A_1 \subset (A_2 \cup C_2)$
2. $A_1 \subset (A_2 \cup D_2)$
3. $A_1 \subset (B_2 \cup C_2)$
4. $A_1 \subset (B_2 \cup D_2)$

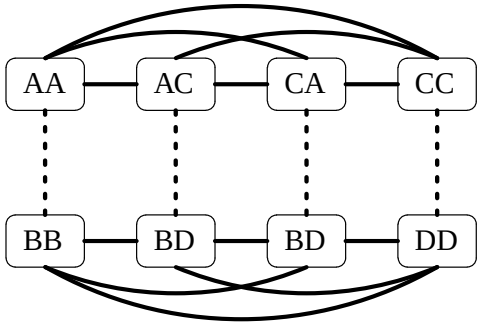
Prenons le cas 1.

- Soit $v \in (A_2 \uplus C_2)$. v est adjacent soit à A_2 (s'il est dans C_2) soit à C_2 (dans le cas contraire). Il touche donc forcément des sommets de A_1, et n'appartient donc pas à D_1. Donc $D_1 \subset (B_2 \cup D_2)$.
- Soit $v \in (B_2 \uplus D_2)$. v est non-adjacent soit à A_2 (s'il est dans D_2) soit à C_2 (dans le cas contraire). Il ne peut pas être adjacent à tous les sommets de A_1, et n'appartient donc pas à C_1. Donc $C_1 \subset (A_2 \cup C_2)$.
- Enfin, soit $v \in (A_2 \cup C_2)$. v est adjacent soit à A_2 soit à C_2, il touche donc forcément des sommets de C_1, et n'appartient donc pas à B_1. Donc $B_1 \subset (B_2 \cup D_2)$.

En résumé : $A_1 \cup C_1 = A_2 \cup C_2$ et $B_1 \cup D_1 = B_2 \cup D_2$. Prendre le cas 4 nous aurait conduit à trouver le même résultat, tandis que les cas 2 et 3 amènent à $A_1 \cup D_1 = A_2 \cup D_2$ et $B_1 \cup C_1 = B_2 \cup C_2$. Mais nous avons vu que complémenter G inverse C et D dans tous les 2-joints. Les cas 1 et 4 de G sont donc les cas 2 et 3 de $\overline{G}$. Nous allons donc continuer dans le cas 1 (et 4), puis nous passerons facilement aux deux autres cas par complémentation.

Considérons les ensembles (éventuellement vides) $AA = A_1 \cap A_2$, $AC = A_1 \cap C_2$, $CA = C_1 \cap A_2$, $CC = C_1 \cap C_2$, $BB = B_1 \cap B_2$, $BD = B_1 \cap D_2$, $DB = D_1 \cap B_2$ et $DD = D_1 \cap D_2$.

Puisque J_1 est un 2-joint, AA est adjacent à CA et CC, et non-adjacent à DB et DD. Puisque J_2 est un 2-joint, AA est adjacent à AC et CC, et non-adjacent à BD et DD. En faisant ce raisonnement pour tous les autres ensembles, on obtient le graphe suivant, où un trait plein entre deux ensembles signifie une adjacence, un trait pointillé qu'il n'y a pas nécessairement adjacence ni non-adjacence, et pas de trait signifie non-adjacence.

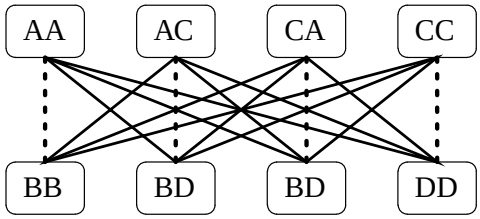


Décomposition K

En reprenant les notations du début de la preuve, on a

- $J_3 = \{AA \cup BB, (AC \cup CA \cup CC) \cup (BD \cup DB \cup DD)\} \in \mathcal{J}$
- $J_4 = \{AC \cup BD, (AA \cup CA \cup CC) \cup (BB \cup DB \cup DD)\} \in \mathcal{J}$
- $J_5 = \{CA \cup DB, (AC \cup AA \cup CC) \cup (BD \cup BB \cup DD)\} \in \mathcal{J}$
- $J_6 = \{CC \cup DD, (AC \cup CA \cup AA) \cup (BD \cup DB \cup BB)\} \in \mathcal{J}$
- $J_7 = \{(AA \cup CC) \cup (BB \cup DD), (AC \cup CA) \cup (BD \cup DB)\} \in \mathcal{J}$

Pour parfaire la preuve, il manque les cas 3 et 4. Dans ce cas, on obtient le graphe suivant, et le théorème est vrai aussi (avec les mêmes définitions des J_i) :



Décomposition S

□

Cette démonstration nous renseigne sur la forme des décompositions *complètes* en 2-joints. Le premier cas est appelé **Décomposition K** (K pour Clique) car il correspond à la décomposition du graphe $2K_n$, qui possède, rappelons-le, $\Theta(2^{2n})$ 2-joints. Le deuxième est appelé **Décomposition S** (S pour Stable) et correspond à la décomposition de $\overline{2K_n}$.

Il y a donc, exactement comme dans la décomposition modulaire, deux types de nœuds complets qui s'échangent par passage au complémentaire, et un type de nœud premier [9].

3.6.3 Analyse

On peut caractériser précisément les graphes dégénérés (dont l'arbre de décomposition n'a qu'un nœud `complet`, c'est-à-dire que toute bipartition des sommets définit un 2-joint) grâce à la preuve du théorème précédent :

Proposition 34
Les graphes dégénérés pour la décomposition en 2-joints sont $2K_n$ *et* $\overline{2K_n}$.

Un graphe G est **cobiparti** si ses sommets peuvent être partitionne en $A \uplus B$ de sorte que $G[A]$ et $G[B]$ soient des cliques. Ses **composantes** sont les composantes connexes du sous-graphe obtenu en ne laissant que les arêtes joignant un sommet de A à un sommet de B. Par exemple, le P_4 $e - f - g - h$ est cobiparti avec $A = \{e, f\}$ et $B = \{g, h\}$ et ses composantes sont $\{e\}$, $\{f, g\}$ et $\{h\}$. Le $2K_2$ $a - b$ $c - d$ est aussi cobiparti ; ses composantes sont les singletons.

Proposition 35
Un graphe cobiparti à plusieurs composantes possède une décomposition K*. Les fils du nœud* `complet` *sont ses composantes.*

Un graphe est **biparti** si ses sommets peuvent être partitionnés en $A \uplus B$ de sorte que $G[A]$ et $G[B]$ soient des stables. Ses **composantes** sont les composantes de $\overline{G}$ (qui est cobiparti). Par exemple, le P_4 $e - f - g - h$ est biparti avec $A = \{e, g\}$ et $B = \{f, h\}$ et ses composantes sont

9. C'est une coïncidence notable. Dans la décomposition en coupe, il y a aussi deux types de nœuds complets... Par contre, cette décomposition ne passe pas au complémentaire.

$\{e\}$, $\{f, g\}$ et $\{h\}$. Le $2K_2$ $a - b$ $c - d$ est aussi biparti ; ses composantes sont triviales.

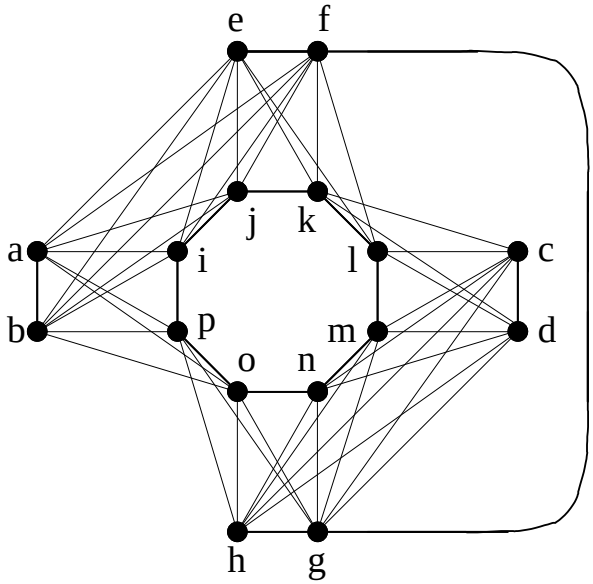
Proposition 36
Un graphe biparti à plusieurs composantes possède une décomposition S. Les fils du nœud `complet` *sont ses composantes.*

Il est beaucoup plus dur de caractériser les graphes premiers. Cette classe n'est pas héréditaire (P_4, décomposable, est un sous-graphe induit de C_8, premier) ni close par minoration (C_4 est un mineur de C_8). La classe des graphes décomposables non plus d'ailleurs.

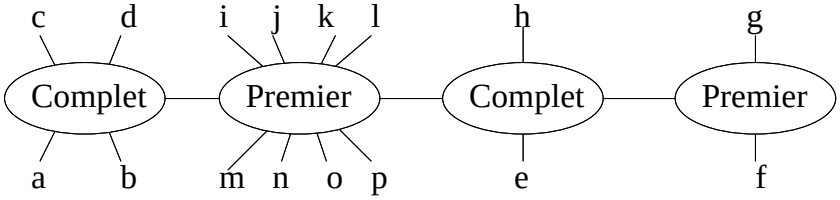
Question ouverte 1
Caractériser les graphes premiers pour la décomposition en 2-joints.

3.6.4 Le squelette. Exemple

Considérons le graphe suivant. Les arêtes dessinées en gras servent seulement à mettre en valeur les composantes.



Son arbre de décomposition bipartitive est le suivant (l'ordre des fils d'un nœud est, on le rappelle, quelconque).



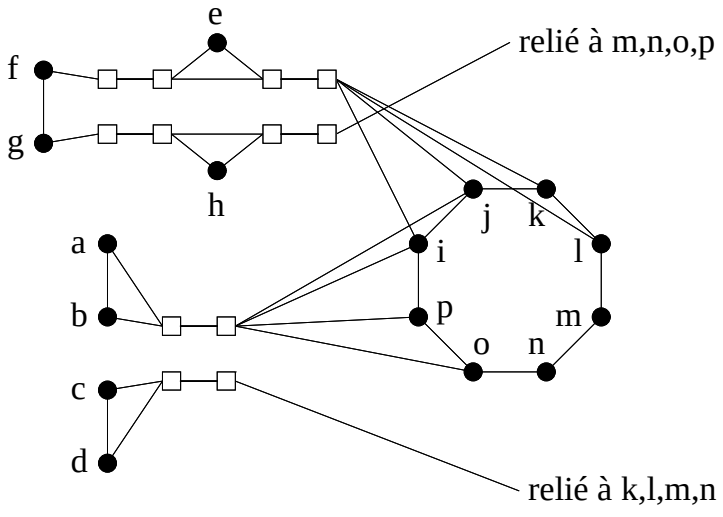
On constate essentiellement trois *composantes* : le $2K_2$ $\{a, b, c, d\}$, le P_4 $\{e, f, g, h\}$ et le C_8 $\{i, j, k, l, m, n, o, p\}$. C_8 est un graphe *premier* pour la décomposition en 2-joint. $2K_2$, P_4 et C_4 sont les trois seuls graphes ayant une décomposition *complète* à la fois K et S.

Cet arbre permet de lire tous les 2-joints de G, qui sont par exemple $(\emptyset, \{a, b\}, \{e, f, i, j, p, o\}, \{c, d, h, g, k, l, m, n\})$, ou $(\{e, f\}, \{g\}, \{a, b, i, j, k, l\}, \{c, d, h, m, n, o, p\})$. En fait, il ne donne que les bipartitions, la donnée du graphe devant servir à retrouver (de façon non-ambigüe) la quadripartition.

Il existe une façon de mélanger l'arbre de décomposition au graphe lui-même, afin de coder le graphe (c'est-à-dire de représenter toutes les adjacences, de façon non-ambigüe) de façon compacte. La *squelettisation* présentée ici s'inspire de cette de [Lan01], qui concerne les coupes d'un graphe. Pour cela, on remplace chaque nœud de l'arbre par le sous-graphe induit par ses

feuilles. Chaque arête est remplacée par le gadget . Chacun des quatre carrés est un *sommet spécial*, auquel peut être adjacent un sommet normal ou un autre sommet spécial. Deux sommets sont adjacents si et seulement si il existe un chemin entre eux ne contenant que des sommets spéciaux (sauf les extrêmités).

Voici comment construire le gadjet d'un 2-joint (A, B, C, D) : on dessine le graphe union de $G[A]$, $G[B]$, $G[C]$, $G[D]$ et d'un gadjet. Tous les sommets de A sont faits voisins du sommet spécial en haut à gauche, ceux de B du sommet spécial en bas à gauche, ceux de C du sommet spécial en haut à droite, ceux de D du sommet spécial en bas à droite. La transformation passant de l'arbre et du graphe au squelette est bijective et non-ambigüe. Le graphe *squelettisé* de l'exemple est (8 arêtes n'ont pas été dessinées, entre deux sommets spéciaux et le C_8, pour plus de lisibilité) :



squelette de G

Sur ce dessin, on peut lire que f et i sont adjacents : il existe un chemin qui les relie, en passant uniquement par des arêtes pleines, et formé uniquement de sommets spéciaux (sauf les deux extrémités). Aucun chemin de la sorte n'existe entre g et i, qui sont donc non-adjacents. Le

codage est plus compact : un 2-joint (A, B, C, D) implique $|A|.|C| + |B|.|D|$ arêtes, replacées dans le squelette par 4 sommets et $|A| + |B| + |C| + |D| + 4$ arêtes, passant du quadratique au linéaire.

3.6.5 Une extension : les k-joints. Algorithme.

Une extension naturelle du 2-joint est le k-joint. [EKR97] établit (sans preuve, il faut étendre un algorithme qui trouve un 2-module) que trouver un k-module peut se faire en temps polynômial. Une classe particulière de k-module est le k-joint, ici présenté.

Définition 19
*Soit $G = (V, E)$ un graphe. Un k-**joint strict** est une partition (stricte) de V en $2k$ ensembles $A_1 \dots A_k$ et $B_1 \dots B_k$ tels que pour tout i, A_i est adjacent à B_i et non-adjacent à B_j pour tout $j \neq i$.*

Un k-joint au sens large est un l-joint strict pour l⩽k. Il est clair que G admet un k-joint strict seulement si $|V| \geqslant 2k$. Dans ce cas, voici un algorithme polynômial qui exhibe un tel k-joint, ou certifie que le graphe est dépourvu de k-joint strict.

Première étape : choix d'une graine

On considère *tous* les choix possibles de couplages de k arêtes, c'est-à-dire tous les couples distincts de V $\{(a_1, b_1) \dots (a_k, b_k)\}$ tel que l'ensemble des arêtes de $G[a_1, b_1, \dots a_k, b_k]$ soit exactement $\{(a_i, b_i) \mid i = 1..k\}$. Un tel couplage est la **graine**. Il sert à initialiser l'algorithme : on fait

$$\forall i \;\; A_i \leftarrow \{a_i\} \;\; et \;\; B_i \leftarrow \{b_i\}$$

Il y a $\frac{m!}{k!(m-k)!} \leqslant m^k$ graines possibles, ce qui est une quantité polynômiale quand k est fixé ; cet algorithme est exponentiel en la taille d'un paramètre.

Deuxième étape : assignations non-ambiguës

A ce point on travaille avec une graine donnée V_0. On commence par vérifier que cette graine est valide, c'est-à-dire que $G[V_0]$ induit le graphe kK_2 et $\forall i (a_i b_i) \in E$.

Posons $A = \bigcup_{i=1}^{k} A_i$ et $B = \bigcup_{i=1}^{k} B_i$. Cela définit les deux *côtés* du k-joint.
Pour tout sommet x de G, d'après la définition, il est nécessaire que si x est d'un côté donné (disons A), x soit adjacent à un et un seul des sommets de la graine situés de l'autre côté (disons b_i) et non-adjacent à tous les autres. De plus, l'adjacence à b_i impose à x de se trouver dans A_i. Donc, pour tout $x \in V$:

- Si $|\Gamma(x) \cap A| \neq 1$ et $|\Gamma(x) \cap B| \neq 1$ alors **échec** : G n'admet pas de k-joint avec la graine donnée.
- Si $|\Gamma(x) \cap A| \neq 1$ et $|\Gamma(x) \cap B| = 1$, alors soit i tel que x est adjacent à b_i. $A_i \leftarrow A_i \cup \{x\}$.

– Si $|\Gamma(x) \cap A| = 1$ et $|\Gamma(x) \cap B| \neq 1$, alors soit i tel que x est adjacent à a_i. $B_i \leftarrow B_i \cup \{x\}$.
– Et si $|\Gamma(x) \cap A| = 1$ et $|\Gamma(x) \cap B| = 1$, alors soient i et j tels que x est adjacent à a_j et b_i. x est **ambigu** : on peut le placer soit dans A_i, soit dans B_j. On le place alors dans un ensemble E_{ij}.

Troisième étape : levée des ambiguïtés

On va attribuer les sommets de E_{ij} soit à A_i, soit à B_j, en se ramenant au problème 2-SAT.

Le problème **2-SAT** consiste, étant donné une expression booléenne sous forme conjonctive avec des clauses de taille 1 ou 2, à déterminer s'il existe une assignation des variables telle que l'expression puisse être satisfaite. Il est bien connu [APT79] que ce problème se résout en temps linéaire en le nombre de clauses plus celui de variables.

Pour chaque sommet x, soit $x\alpha i$ la variable booléenne valant vrai si, et seulement si, $x \in A_i$, et $x\beta j$ la variable booléenne valant vrai si, et seulement si, $x \in B_j$. Il y a donc $2nk$ variables différentes. On fait la modélisation 2-SAT suivante (les $\rightsquigarrow$ représentent la transformation en clauses) :

$$x \in A_i \rightsquigarrow (x\alpha i) \tag{3.1}$$
$$x \in B_i \rightsquigarrow (x\beta i) \tag{3.2}$$
$$x \in E_{ij} \rightsquigarrow (x\alpha i \vee x\beta j) \wedge (\neg x\alpha i \vee \neg x\beta j) \tag{3.3}$$
$$x \notin (A_i \cup E_{ij}) \rightsquigarrow (\neg x\alpha i) \tag{3.4}$$
$$x \notin (B_j \cup E_{ij}) \rightsquigarrow (\neg x\beta j) \tag{3.5}$$
$$(x, y) \in E \rightsquigarrow \wedge_{i \neq j}(\neg x\alpha i \vee \neg y\beta j) \tag{3.6}$$
$$(x, y) \notin E \rightsquigarrow \wedge_i(\neg x\alpha i \vee \neg y\beta i) \tag{3.7}$$

Les cinq premiers ensembles de clauses expriment la place des sommets ambigus et non-ambigus dans un et un seul ensemble. Comme pour un sommet ambigu on hésite entre deux ensembles seulement, il s'agit bien de 2-clauses. Il y a $O(nk)$ clauses générées par les cinq premières lignes.

Les deux derniers ensembles de clauses traduisent les deux axiomes de la définition du k-joint. La ligne 6 génère $O(k^2)$ clauses par arêtes et la ligne 7 en génère $O(k)$ par non-arête ; il y a donc $O(k^2n^2)$ 2-clauses pour traduire le graphe.

Lemme 9 *Si le problème 2-SAT ainsi défini possède une assignation valide des variables, alors G admet un k-joint que cette assignation permet de connaître.*

Démonstration: Chacune des parties A_i et B_i est non-vide puisqu'un sommet lui est attribué à l'étape 1. Les ensembles de clauses de 1 à 5 assurent que chaque sommet appartient à une et une seule partie. Les ensembles de clauses 6 et 7 que la définition du k-joint est respectée. Toutes

ces conditions sont nécessaires : G admet un k-joint seulement si elles sont présentes. Aucune ne manque pour transcrire la définition du k-joint, donc G admet un k-joint si elles sont présentes.

□

La **complexité** de l'algorithme est $O(k^2n^2)$ par graine et $O(k^2n^2m^k)$ en tout. On peut en déduire

Proposition 37
Le calcul de la décomposition en 2-joints d'un graphe est un problème polynômial, et peut se résoudre en $O(n^3m^2)$.

Démonstration: Les 2-joints non-stricts sont des modules, qui peuvent se trouver en $O(n+m)$ (voir algorithmes de [MS99, CH94, DGM01] et ceux de la partie II). On peut trouver en $O(n^2m^2)$ un 2-joint strict $\{J^0, J^1\}$ d'un graphe, avec l'algorithme des k-joints. On relance alors sur $G[J^0]$ et sur $G[J^1]$, et ainsi de suite. L'algorithme est appelé $O(n)$ fois, car à chaque appel on rajoute une arête dans un arbre à n feuilles sans nœud d'arité 2. Cela construit un arbre, non-enraciné. Cet arbre est un sous-arbre de l'arbre bipartitif des 2-joints : tous les nœuds premiers apparaissent correctement, mais des nœuds frêles peuvent être coupé en plusieurs nœuds qu'il faut regrouper (une technique semblable est exposée dans [Cun82] pour la décomposition en coupes). Cela se fait facilement : pour toute arête entre A et B, on teste si en prenant des fils de A et des fils de B on obtient un 2-joint. Si oui, A et B sont regroupés. Reconnaître si une bipartition est un 2-joint est clairement linéaire, donc cela se fait en $O(nm)$.

□

Conclusion

Trouver un k-joint, pour k fixé, est donc un problème polynômial. En revanche, le problème suivant :
Donnée : un graphe G
Résultat : le plus petit entier k tel que G ait un k-joint strict
qui est un problème NP, demeure d'une complexité inconnue.

Conjecture 1
Ce problème est NP-complet.

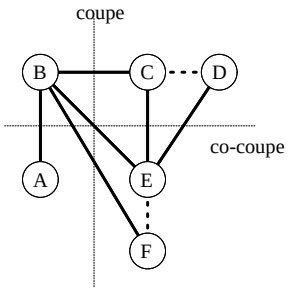
3.7 Incompatibilité des modules, 2-joints, coupes et co-coupes

Les décompositions en 2-joints, en coupes et en co-coupes *étendent* la décomposition modulaire, puisqu'un 1-module est un cas particulier de chacune. Nous allons voir qu'elles sont mutuellement exclusives : un graphe qui est décomposable selon deux de ces décompositions possède une structure très particulière, qui lui donne exactement **une** décomposition de chaque

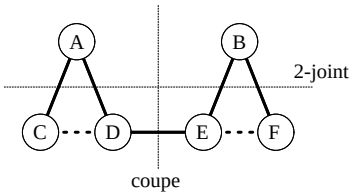
forme, comme le prouvent les lemmes suivants. Ce ne sont donc pas des décompositions très intéressantes pour de tels graphes, puisqu'alors les arbres de décomposition ont hauteur au plus 3.

Il existe donc trois types de graphes ayant une décomposition non-triviale : ceux « de type coupes », ceux « de type co-coupes » et ceux « de type 2-joints ». Dans chacun des cas, la décomposition du même nom est l'extension adéquate de la décomposition modulaire à utiliser.

Proposition 38 ([Lan01])
Un graphe possédant une coupe et une co-coupe non-triviales possède une unique coupe et une unique co-coupe non-triviales, et a la structure suivante :



Proposition 39
Un graphe possédant une coupe et un 2-joint non-triviaux possède une unique coupe et un unique 2-joint non-triviaux, et a la structure suivante :



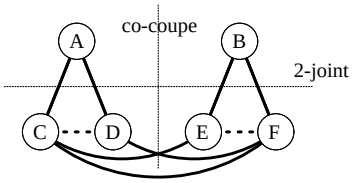
Démonstration: Soit (A, B, C, D) un 2-joint de G. Si il existe une arête entre un sommet de A et un sommet de B, et une arête entre un sommet de C et un sommet de D, alors toute coupe (A', B', C', D') de G est telle que $B' \cup C'$ ne touche pas les sommets extrémités de telles arêtes. Donc $(C \cup D) \subset D'$ et tout sommet de $(A \cup B)$ touche soit D' (et appartient donc à C'), soit C' : $A' = \emptyset$. Dans ce cas toutes les coupes sont triviales.

Sinon, supposons A non-adjacent à B. Puisque G est connexe, il y a des arêtes entre C et D. Soit (A', B', C', D') une coupe non-triviale de G. Supposons $A \cap B' \neq \emptyset$. On a $(A' \cup B' \cup C') \subset A \cup C$, puisque les seuls voisins de A sont dans $A \cup C$. Donc $(B \cup D) \subset D'$. On a $C' = \emptyset$, puisque tout non-voisin de A est dans $B \cup D$. Donc $A \cap B' = \emptyset$. De même $A \cap C' = \emptyset$. Si A intersecte A' et D', C fait de même et il y a des arêtes entre A' et D', contradiction. On peut donc supposer sans perte de généralité $A \subset A'$ et $D \subset D'$. B est donc inclus dans $(A' \cup B')$ (sinon une arête relie A' à C' ou D', et C dans $(C' \cup D')$. Cela amène le résultat.

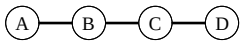
□

En corollaire :

Proposition 40
Un graphe possédant une co-coupe et un 2-joint non-triviaux possède une unique co-coupe et un unique 2-joint non-triviaux, et a la structure suivante :



Corollaire 1
Un graphe possédant une coupe, une co-coupe et un 2-joint non-triviaux ne possède qu'eux, et a la structure suivante :



En particulier, le seul graphe 1-premier possédant les trois est P_4.

Ainsi coupes, co-coupes et 2-joints étendent chacun la décomposition modulaire, car un membre trivial de l'une de ces trois familles est un module, mais de façon *presque* mutuellement exclusive. Donc, pour avoir un arbre de décomposition d'un graphe (qui ne sera plus un arbre partitif, enraciné, mais un arbre bipartitif sans racine) on n'a qu'à détecter si une des configurations précédentes se produit, et sinon on choisit de façon canonique celle des trois décompositions ci-dessus qui est valable pour le graphe (s'il y en a). Cela amène d'ailleurs à se poser la question de savoir les graphes *indécomposables* pour cette famille de décomposition : ce sont exactement ceux qui n'ont pas de 2-module parfait. En conclusion, nous venons de définir une décomposition **canonique** des graphes selon la famille de leurs 2-modules parfaits. Ce n'est pas tout à fait une décomposition bipartitive, car dans les trois cas ci-dessus des bipartitions se chevauchent en violant l'axiome des familles bipartitives, mais *presque* :

Théorème 18
En dehors des trois cas ci-dessus où le graphe ne possède que deux bipartitions parfaites non-triviales, la famille des bipartitions parfaites d'un graphe est bipartitive et est soit la famille de ses coupes, soit la famille de ses co-coupes, soit la famille de ses 2-joints.

3.8 Preuve du théorème 14

Cette section fournit une preuve du théorème 10, et définit la liste des différents cas de *conflits* pouvant arriver. Tout au long de cette section nous considérons deux 2-modules d'un graphe **premier**, $M = \{M_1, M_2\}$ et $M' = \{M'_1, M'_2\}$, qui se *chevauchent* : $(M_1 \cup M_2) \circledcirc (M'_1 \cup M'_2)$. Par souci de concision, ces propriétés importantes ne seront rappelées que sous forme concise ou pas du tout. Dans tous les dessins, M sera dessiné en blanc et M' en grisé. Nous distinguons

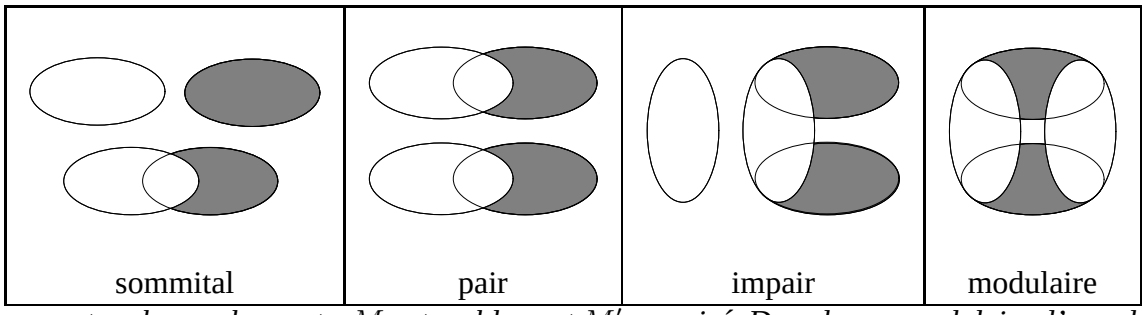
trois grandes classes de chevauchement :

Définition 20

- M *et* M' *sont en chevauchement* ***modulaire*** *si trois au moins des quatre intersections* $M_1 \cap M'_1$, $M_1 \cap M'_2$, $M_2 \cap M'_1$, $M_2 \cap M'_2$ *sont non vides.*
- M *et* M' *sont en chevauchement* ***sommital*** *si seulement une de ces intersection est non vide.*
- M *et* M' *sont en chevauchement* ***pair*** *si exactement deux intersections sont non-vides et*

$$(M_1 \cap M'_1 = \emptyset \ \wedge \ M_2 \cap M'_2 = \emptyset) \ \vee \ (M_1 \cap M'_2 = \emptyset \ \wedge \ M_2 \cap M'_1 = \emptyset)$$

- M *et* M' *sont en chevauchement* ***impair*** *dans le cas restant.*



Les quatre chevauchements. M *est en blanc et* M' *en grisé. Dans le cas modulaire, l'une des intersections entre ovales peut être vide.*

L'outil suivant nous sera très utile pour l'étude de cas :

Lemme 10 (lemme-outil) *Soit* i *et* j *valant 1 ou 2. Si* $M_i \smallsetminus M' \neq \emptyset$ *et* $M'_j \smallsetminus M \neq \emptyset$ *alors* $M_i \smallsetminus M'$ *et* M'_j *sont soit adjacents, soit non-adjacents.*

Démonstration: Soient $x \in (M_i \smallsetminus M')$ et $x' \in (M'_j \smallsetminus M)$. Supposons x voisin de x'. Soient $y \in (M_i \smallsetminus M')$ et $y' \in M'_j$. À cause de la 2-modularité de M', $x \notin M'$ est voisin de y'. À cause de la 2-modularité de M, $x' \notin M$ est voisin de y. À cause de la 2-modularité de M', $y \notin M'$ est voisin de y'. Donc si deux sommets de $(M_i \smallsetminus M')$ et $(M'_j \smallsetminus M)$ sont voisins, $(M_i \smallsetminus M')$ et M_j sont adjacent. La même preuve sur $\overline{G}$ montre leur non-adjacence, dans le cas contraire.

□

3.8.1 Le chevauchement sommital

On va supposer, sans perte de généralité, que $M_2 \cap M'_2 \neq \emptyset$, les trois autres intersections étant vides. Le lemme suivant justifie le nom de *sommital* du chevauchement : l'intersection des 2-modules est réduite à un sommet.

Lemme 11 $M \cap M' = \{m\}$

Démonstration: Nul sommet de $M \setminus M'$ ne peut distinguer $M_2 \cap M'_2$, à cause de la 2-modularité de M'. De même aucun sommet de $M' \setminus M$, et bien sûr aucun sommet extérieur à ces modules. Or $M \cap M' = M_2 \cap M'_2$. Cet ensemble est donc un 1-module. G étant 1-premier, ce module est trivial à un sommet, m.

□

Donc $M \cap M'$ est le 2-module trivial $\{m\}$. Par contre, on remarque que $M \cup M'$ n'a aucune raison d'être un 2-module[10]. Nous allons nous intéresser à la différence $M \setminus M'$. Il nous faut distinguer des cas, selon les relations entre M_2 et M'_2. Le cas $M_2 \cap M'_2 = \emptyset$ étant exclu, reste $M_2 \oo M'_2$, $M_2 = M'_2$, $M_2 \subsetneq M'_2$ et $M'_2 \subsetneq M_2$.

Cas 1

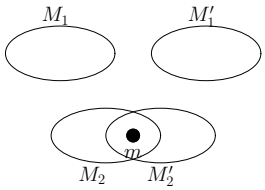
$$M_2 \oo M'_2$$

Lemme 12 *m est M-homogène et M'-homogène*
$M_1 \setminus M_2$ et $M_2 \setminus M_1$ sont des 2-modules.

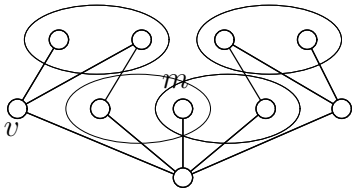
Démonstration: D'après le lemme 11, $M_2 \cap M'_2 = \{m\}$. On applique une première fois le lemme-outil sur M_1 et M'_2 : $m \in M'_2$ est soit adjacent soit non-adjacent à $M_1 \setminus M' = M_1$. Puis une deuxième application sur M_2 et M'_2 : $m \in M'_2$ est soit adjacent soit non-adjacent à $M_2 \setminus M' = M_2 \setminus \{m\}$. m est donc M-homogène. La même preuve montre qu'il est M'-homogène.

Or seul m pourrait distinguer $M \setminus M' = M \setminus \{m\}$, donc $M \setminus M'$ est un 2-module $\{M_1, M_2 \setminus \{m\}\}$. Même preuve pour $M' \setminus M$.

□



Si $M \cup M'$ n'est pas un 2-module, on a alors un **conflit d'union de type I**.



*Exemple de **conflit d'union de type I** : v distingue $M \cup M'$.*

10. C'est, sauf exception, un 3-module $\{M_1, M'_1, M_2 \cup M'_2\}$

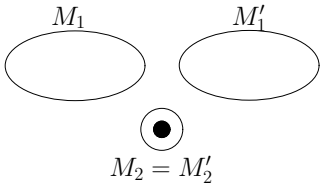
Cas 2

$$M_2 = M_2'$$

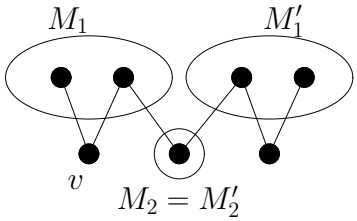
Lemme 13 *M et M' sont boiteux*
$M \smallsetminus M'$ et $M' \smallsetminus M$ sont des 2-modules.

Démonstration: D'après le lemme 11, $M_2 = M_2' = \{m\}$. Il vient par définition que M et M' sont boiteux. $M \smallsetminus M' = M_1$ est un sesquimodule $\{M_1 \cap \Gamma(m), M_1 \cap \overline{\Gamma}(m)\}$ (cf §3.4), de même que $M' \smallsetminus M = M_1'$.

□



Si $M \cup M'$ n'est pas un 2-module, on a alors un **conflit d'union de type II**.



*Exemple de **conflit d'union de type II** : v distingue $M \cup M'$.*

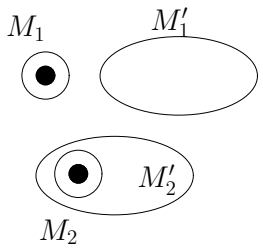
Cas 3

$$M_2 \subsetneq M_2' \; (ou\; l'inverse)$$

Lemme 14 *$M = \{\{m\}, \{m'\}\}$ est un 2-module trivial.*
$M \smallsetminus M' = \{m_1\}$ est un 2-module (trivial).

Démonstration: Posons $M_2 \subsetneq M_2'$. D'après le lemme 11, $M_2 = \{m\}$. On applique le lemme-outil sur M_1 et M_2' : $m \in M_2'$ est soit adjacent soit non-adjacent à $M_1 \smallsetminus M' = M_1$. Or seul m pourraient distinguer M_1. Donc M_1 est un 1-module, singleton d'après la 1-primalité de G.

□



Il n'y a pas de raison que $M' \smallsetminus M$ ou $M \cup M'$ soient des 2-modules. On a alors un **conflit mineur de type I**.

3.8.2 Le chevauchement pair

On va supposer, sans perte de généralité,

$$M_1 \cap M_2' = \emptyset \ \wedge \ M_2 \cap M_1' = \emptyset$$

Lemme 15 *L'intersection de deux 2-modules en chevauchement pair est un 2-module.*

Démonstration: Clairement $\{M_1 \cap M_1', M_2 \cap M_2'\}$ est un 2-module.

□

Pour vérifier l'union et la différence, nous allons traiter tous les cas, selon les relations entre M_1 et M_1', et entre M_2 et M_2'. Certains cas ne sont pas des chevauchement, et les symétries par échange de M et M', ou par échange *simultané* de M_1 et M_2 et M_1' et M_2', réduisent à 4 le nombre de cas à considérer. Dans chaque cas, seul un symétrique est traité.

	$M_1 = M_1'$	$M_1 \subsetneq M_1'$	$M_1 \mathbin{⩒} M_1'$	$M_1' \subsetneq M_1$
$M_2 = M_2'$	$M = M'$	$M \subsetneq M'$	**cas 2**	$M' \subsetneq M$
$M_2 \subsetneq M_2'$	$M \subsetneq M'$	$M \subsetneq M'$	**cas 3**	**cas 4**
$M_2 \mathbin{⩒} M_2'$	**cas 2**	**cas 3**	**cas 1**	**cas 3**
$M_2' \subsetneq M_2$	$M' \subsetneq M$	**cas 4**	**cas 3**	$M' \subsetneq M$

Cas 1

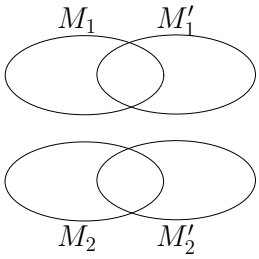
$$M_2 \mathbin{⩒} M_2' \ \wedge \ M_1 \mathbin{⩒} M_1'$$

Lemme 16 *$M \cup M'$, $M \smallsetminus M'$ et $M' \smallsetminus M$ sont des 2-modules.*

Démonstration: $M_1 \smallsetminus M'$ ne saurait être distingué que par des sommets de $M \cap M'$, puisque M est un 2-module. Appliquons le lemme-outil sur M_1 et M_1' : $M_1 \smallsetminus M'$ ne peut être distingué par M_1'. Puis appliquons-le sur M_1 et M_2' : $M_1 \smallsetminus M'$ ne peut être distingué par M_2' non plus. De même ni M_1' ni M_2' ne séparent $M_2 \smallsetminus M'$. Donc $M \smallsetminus M'$ est le 2-module $\{M_1 \smallsetminus M', M_2 \smallsetminus M'\}$. De même, $M' \smallsetminus M$ est le 2-module $\{M_1' \smallsetminus M, M_2' \smallsetminus M\}$.

Soit $z \notin M \cup M'$. Si z est adjacent à M_1, vu que $M_1 \cap M'_1 \neq \emptyset$, z est aussi adjacent à M'_1. De même, z est adjacent à M_2 ssi il est adjacent à M'_2. Donc $M \cup M'$ est le 2-module $\{M_1 \cup M'_1, M_2 \cup M'_2\}$.

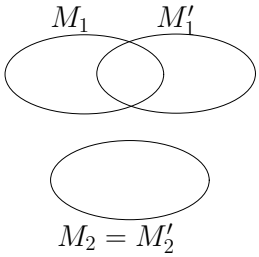
□



Cas 3

$$M_2 = M'_2 \ \wedge \ M_1 \circledcirc M'_1$$

Lemme 17 *$M \cup M'$, $M \setminus M'$ et $M' \setminus M$ sont des 2-modules.*



Démonstration: M'_1 ne peut distinguer $M \setminus M' = M_1 \setminus M'_1$. Tout $x \in M_1 \setminus M'_1$ est soit adjacent, soit non-adjacent à M'_2, à cause de la 2-modularité de M'. $M \setminus M'$ est donc un 2-module $\{(M_1 \setminus M') \cap \Gamma(M'_2), (M_1 \setminus M') \cap \overline{\Gamma}(M'_2)\}$. De même $M' \setminus M$ est un 2-module. Puisque M_1 chevauche M'_1, $M \cup M' = \{M_1 \cup M'_1, M_2\}$ est un 2-module.

□

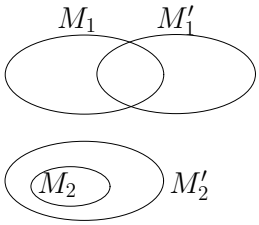
Cas 2

$$M_2 \subsetneq M'_2 \ \wedge \ M_1 \circledcirc M'_1$$

Lemme 18 $M_1 \setminus M'_1 = \{m_1\}$.
m_1 est M-homogène. $M \cup M'$ et $M \setminus M' = \{m_1\}$ sont des 2-modules.

Démonstration: D'après le lemme-outil appliqué sur M_1 et M'_2, $M_1 \setminus M'_1$ ne peut être distingué par M'_2. Or $M_2 \subset M'_2$. Nul sommet hors de $M_1 \setminus M'_1$ ne peut distinguer cet ensemble, qui est donc un 1-module, donc un singleton. Puisque M_1 chevauche M'_1, $M \cup M' = \{M_1 \cup M'_1, M'_2\}$ est un 2-module.

□



Cependant $M' \setminus M$ peut ne pas être un 2-module. On a alors un **conflit de différence de type I**.

Cas 4

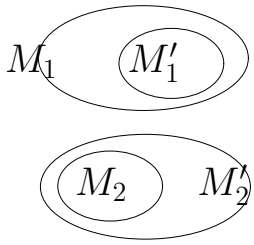
$$M_2 \subsetneq M'_2 \ \wedge \ M'_1 \subsetneq M_1$$

Lemme 19 *$M \cup M'$, $M \setminus M'$ et $M' \setminus M$ sont des 2-modules.*

Démonstration: Il est clair que $M \cup M'$ est le 2-module $\{M_1, M'_2\}$. On a $M \setminus M' = M_1 \setminus M'_1$.

D'après le lemme-outil pris avec M_1 et M'_2, M'_2 ne peut distinguer $M_1 \setminus M'$. $x \in M_1 \setminus M'_1$ est soit adjacent soit non-adjacent à $M_1 \cap M'_1$, à cause de la 2-modularité de M'. $M \setminus M'$ est donc un 2-module $\{(M_1 \setminus M') \cap \Gamma(M'_1), (M_1 \setminus M') \cap \overline{\Gamma}(M'_1)\}$. De même $M' \setminus M$ est le 2-module $\{(M'_2 \setminus M) \cap \Gamma(M_2), (M'_2 \setminus M) \cap \overline{\Gamma}(M_2)\}$.

□



3.8.3 Le chevauchement impair

On va supposer sans perte de généralité

$$M_1 \cap M'_1 = \emptyset \ \wedge \ M_1 \cap M'_2 = \emptyset \ \wedge \ M_2 \cap M'_1 \neq \emptyset \ \wedge \ M_2 \cap M'_2 \neq \emptyset$$

Lemme 20 *$M \cap M'$ et $M \cup M'$ sont des 2-modules.*

Démonstration: Clairement l'intersection $\{M_2 \cap M'_1, M_2 \cap M'_2\}$ est un 2-module. De même pour l'union $\{M_1, M_2 \cup M'_1 \cup M'_2\}$ est un 2-module.

□

Pour ce qui est de la différence, nous allons distinguer quatre cas.

Cas 1

$$M_2 \setminus M' \neq \emptyset \ \wedge \ M'_1 \setminus M \neq \emptyset \ \wedge \ M'_2 \setminus M \neq \emptyset$$

Lemme 21 *$M \setminus M'$ et $M' \setminus M$ sont des 2-modules.*

Démonstration: D'après le lemme-outil pris avec M'_1 et M_2 (resp. M'_2 et M_2), $M'_1 \setminus M$ (resp. $M'_2 \setminus M$) n'est pas distingué par M_2. Donc $M' \setminus M$ est le 2-module $\{M'_1 \setminus M, M'_2 \setminus M\}$.

D'après le lemme-outil pris avec M'_1 et M_2 (resp. M'_2 et M_2), $M_2 \setminus M'$ n'est pas distingué par M'_1 (resp. M'_2) donc $M \setminus M'$ est le 2-module $\{M_1, M_2 \setminus M'\}$.

□

Cas 2

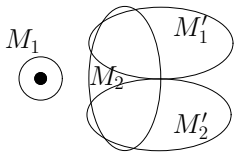
$$M_2 \setminus M' = \emptyset \ \wedge \ M'_1 \setminus M \neq \emptyset \ \wedge \ M'_2 \setminus M \neq \emptyset$$

Lemme 22 $M_1 = \{m\}$
M est boiteux et M' est un m-sesquimodule.
$M \setminus M'$ est un 2-module trivial $\{m\}$.

Démonstration: D'après le lemme-outil pris avec M_1 et M'_1 (resp. M'_2), $M_1 \setminus M = M_1$ ne peut être distingué par M'_1 (resp. M'_2). Or $M_2 \subset M'$. Donc M_1 est un 1-module, donc un singleton.

Soit $z \notin M \cup M'$. z ne peut distinguer M' car $M'_1 \circledcirc M_2 \circledcirc M'_2$. Seul m peut distinguer M', qui est donc un m-sesquimodule. $M_2 \subset M'$ est aussi un m-sesquimodule, donc M est boiteux.

□



$M' \setminus M$ n'est pas nécessairement un 2-module. C'est le **conflit de différence de type II**. Notons que M' comme $M \cup \{m\}$ sont tous deux des m-sesquimodules.

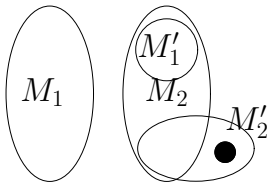
Cas 3

$$M_2 \setminus M' \neq \emptyset \ \wedge \ M'_1 \setminus M = \emptyset \ \wedge \ M'_2 \setminus M \neq \emptyset$$
$$(ou : M_2 \setminus M' \neq \emptyset \ \wedge \ M'_1 \setminus M = \emptyset \ \wedge \ M'_2 \setminus M = \emptyset)$$

Lemme 23 *$M' \setminus M = \{m\}$ est un 2-module trivial*
m est M'-homogène.

Démonstration: Clairement un sommet hors de M_2 ne peut distinguer $M' \setminus M = M'_2 \setminus M$. D'après le lemme-outil pris avec M'_2 et M_2, M_2 ne peut distinguer $M'_2 \setminus M$. Donc $M' \setminus M$ est un 1-module m. Comme m ne distingue pas M_2 et $(M' \setminus \{m\}) \subset M_2$, m est M'-homogène.

□



$M \setminus M'$ n'est pas nécessairement un 2-module. C'est le **conflit de différence de type III**.

Cas 4

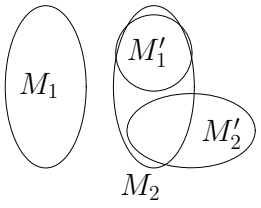
$$M_2 \setminus M' = \emptyset \ \wedge \ M'_1 \setminus M = \emptyset \ \wedge \ M'_2 \setminus M \neq \emptyset$$
$$(ou : M_2 \setminus M' = \emptyset \ \wedge \ M'_1 \setminus M \neq \emptyset \ \wedge \ M'_2 \setminus M = \emptyset)$$

Lemme 24 *$M \setminus M'$ et $M' \setminus M$ sont des 2-modules.*

Démonstration: $M \setminus M' = M_1$ ne peut être distingué que par des sommets de M_2. Or $M_2 \subset M'$. Soit $z \in M'_2 \setminus M$. Si z est adjacent à M_1, alors M_1 est adjacent à M'_2. Et sinon M_1 est non-adjacent à M'_2 : M'_2 ne distingue pas M_1. $x \in M_1$ est soit adjacent, soit non-adjacent à M'_1. M_1 est donc le 2-module $\{M_1 \cap \Gamma(M'_1), M_1 \cap \overline{\Gamma}(M'_1)\}$.

$M' \setminus M = M'_2 \setminus M_2$ ne peut être distingué que par M_2. Or $x \in M'_2 \setminus M_2$ est soit adjacent, soit non-adjacent à M_2. $M' \setminus M$ est donc le 2-module $\{(M' \setminus M) \cap \Gamma(M_2), (M' \setminus M) \cap \overline{\Gamma}(M_2)\}$.

□



3.8.4 Le chevauchement modulaire

le nom du conflit modulaire est justifié par le lemme suivant :

Lemme 25 *Si M et M' sont en chevauchement modulaire alors leur union est $V(G)$.*

Démonstration: Supposons sans perte de généralité que seule l'intersection $M_2 \cap M'_2$ puisse être vide. Tout sommet $x \notin M \cup M'$ ne distingue pas M_2. Supposons-le (l'autre cas étant identique) adjacent à M_2. Comme $M_2 \cap M'_1 \neq \emptyset$, x est adjacent à M'_1. $M'_1 \cap M_1 \neq \emptyset$ entraîne l'adjacence de x à M_1, et $M_1 \cap M'_2 \neq \emptyset$ boucle la chaîne : x est adjacent à $M \cup M'$. Comme le graphe est 1-premier et $|M \cup M'| > 1$, $M \cup M' = V$.

□

Lemme 26 *Si M et M' sont en chevauchement modulaire alors M et M' sont des 2-modules parfaits, ou bien l'un des deux est trivial à $|V|$ ou $|V|-1$ sommets.*

Démonstration: Si M n'est pas parfait, son complémentaire n'est par définition pas un 2-module. Or si $V \setminus M$ n'est pas un 2-module, c'est qu'un sommet $x \in M$ sépare $M'_1 \setminus M$ ou bien $M'_2 \setminus M$. Si on trouve des séparateurs à la fois dans M_1 et M_2 alors, d'après le lemme-outil 2 (énoncé ci-dessous), $M \setminus M' = \emptyset$, donc $M' = V$ est trivial. Sinon, supposons sans perte de généralité que tous les séparateurs de $V \setminus M$ sont dans M_1. Donc $M_1 \setminus M' = \emptyset$.

Nous allons montrer que $M_2 \setminus M' = M \setminus M'$ est un 1-module. Avec le lemme 25, on en conclu que M' est un 2-module à $|V|-1$ sommets, donc trivial $\{\Gamma(m), \overline{\Gamma}(m)\}$ où m est l'unique sommet de $M \setminus M'$.

Seuls des sommets de M peuvent distinguer $M_2 \setminus M'$. Or $M_1 \setminus M' = \emptyset$, donc les sommets qui le distingue sont dans $M_i \cap M'_j$. D'après le lemme-outil 2, si $x \in M'_i$ sépare $M_2 \setminus M'$ alors $M'_i \setminus M = \emptyset$. Donc, si $M_2 \setminus M'$ a des séparateurs, on peut supposer (sans perte de généralité) qu'ils sont dans M'_1.

Résumons : $M_1 \setminus M' = \emptyset$, $M'_1 \setminus M = \emptyset$, $M_2 \setminus M' \neq \emptyset$ (sans quoi M est trivial) et $M'_2 \setminus M \neq \emptyset$ (sans quoi M' est trivial). $V \setminus M = M' \setminus M = M'_2 \setminus M$ est supposé ne pas être un 2-module. Soit $z \in M_2 \setminus M' \neq \emptyset$. Si z est adjacent à M'_2, alors $M'_2 \setminus M'$ est adjacent à M_2, et sinon lui est non-adjacent. Et un sommet de $M'_2 \setminus M$ est soit adjacent, soit non-adjacent à M_1. Donc $M'_2 \setminus M$ est un 2-module $\{(M'_2 \setminus M) \cap \Gamma(M_1), (M'_2 \setminus M) \cap \overline{\Gamma}(M_1)\}$, contradiction.

□

Lemme 27 (lemme-outil 2) *Si un sommet de M_i sépare $M'_j \setminus M$, alors $M_i \setminus M' = \emptyset$.*

Démonstration: Soit $x \in M_i$ qui distingue $\{y, y'\} \subset (M'_j \setminus M)$. On a nécessairement $x \in (M \cap M')$. Si $M_i \setminus M' \neq \emptyset$, alors prenons z dans cet ensemble. Si z est voisin de y, à cause de la 2-modularité de M' z est voisin de y'. Et la 2-modularité de M entraîne que y comme y' sont voisins de x, qui ne les sépare pas, contradiction. (même preuve si z non-voisin de y).

□

D'après le lemme 25, M étant parfait, $M' \setminus M = V \setminus M$ est un 2-module, et $M \setminus M'$ aussi. Et d'après le lemme 26 $M \cup M'$ est un 2-module (trivial). Toutefois, l'intersection $M \cap M'$ n'est en général pas un 2-module[11] C'est le **conflit d'intersection**. Aboutir aux configurations données §3.3 demande un travail fastidieux avec de nombreux cas. Cela prouve que les conflits n'ont lieu qu'entre coupes, entre co-coupes ou entre 2-joints. Cependant comme ce résultat ne sera pas utilisé par la suite je ne le prouverai pas.

3.9 Conclusion et (non)-perspectives

À la fin de ce chapitre, on a une vue plus fine du fonctionnement des 2-modules, et un bon chemin vers leur compréhension a été fait. On sait en particulier que la famille est *presque partitive* et que les conflits, c'est-à-dire les violations des axiomes des familles faiblement partitives, sont de forme bien déterminée (théorème 14). Des sous-familles des 2-modules sont étudiées : 1-modules, sesquimodules et 2-modules boiteux, coupes et co-coupes, 2-joints. En particulier les décompositions 2-modulaires *parfaites*, celle qui coupent un graphe entre deux 2-modules, sont bien caractérisées. Sauf cas particuliers, la famille des 2-modules parfaits d'un graphe est bipartitive, et rentre dans trois cas distincts (coupes, co-coupes et 2-joints). Il existe un algorithme de décomposition efficace pour chacune de ces familles.

3.9.1 Pourquoi pas une décomposition 2-modulaire canonique ?

En revanche, parler de sous-familles produit une grande quantité de 2-modules « laissés-pour-compte » qui ne sont décrit par aucune théorie. Il est frustrant, au terme de ce chapitre, de ne pas pouvoir présenter une famille de 2-modules qui soit partitive et regroupe « presque tous » les 2-modules. On aurait alors un bel arbre de décomposition. Cette famille serait définie par récurrence. Voici comment cette théorie commencerait.

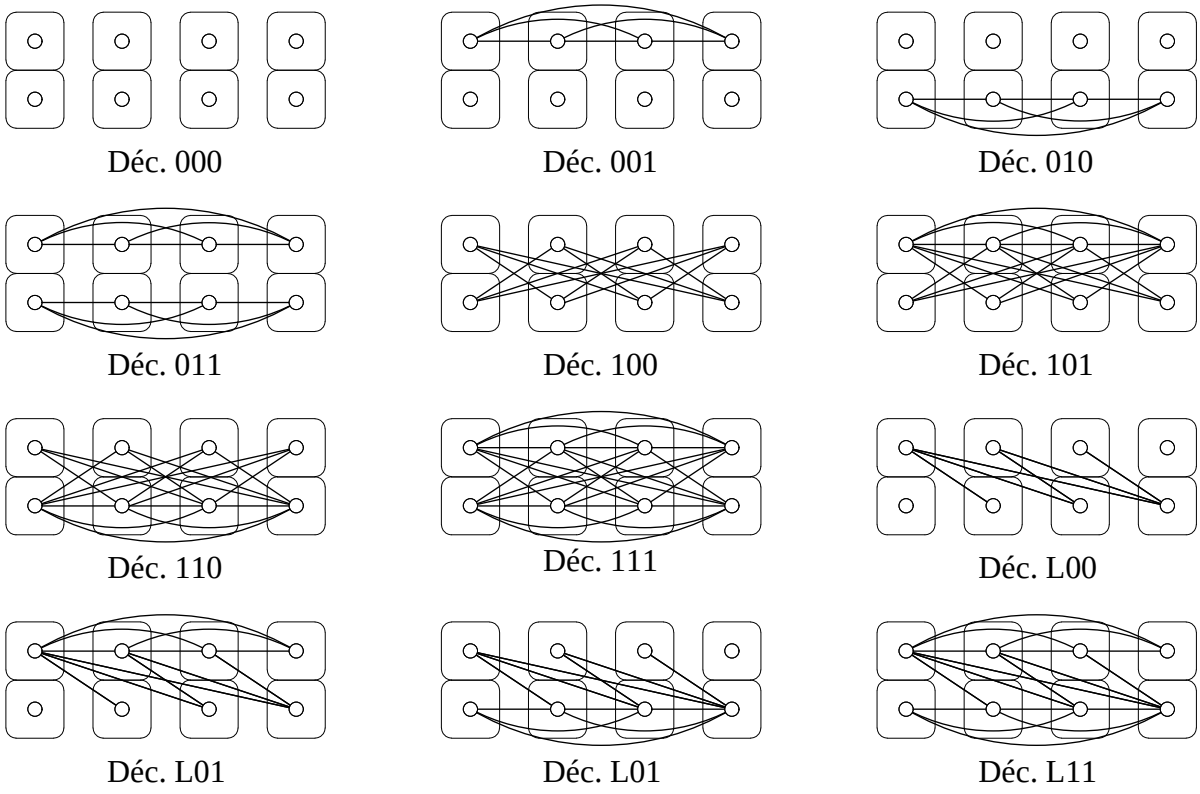
Pour débusquer les décompositions de type `complet` et `linéaire`, une possibilité est de regarder à quoi ressemble une partition de V en 2-modules $M_1 \uplus \ldots \uplus M_k$ tels que toute union arbitraire ou ordonnée de 2-modules soit un 2-module. On trouve les 12 cas de la page suivante.

Bien sûr, puisqu'ils correspondent à des 2-modules parfaits, ils sont déjà connus : modules pour 000, 111, L10 et L01 ; coupes pour 001, 010 et L00 ; co-coupes pour 101, 110 et L11 ; 2-joints pour 011 (décomposition K) et 100 (décomposition S). Les huit premiers sont de type `complet` et les quatre derniers de type `linéaire`. Les incompatibilités entre ces décomposition, étudiées en §3.7, permettent même de définir assez canoniquement quelle décomposition choisir en cas de choix multiple. Et toutes ces décompositions sont calculables polynômialement...

On a alors envie de définir un arbre de décomposition avec au moins ces douze marques de nœuds. Il y a deux obstacles. Tout d'abord, le cas `premier` manque. Or les 2-modules maximaux se chevauchent et peuvent avoir des conflits d'union, mais ce n'est pas trop grave car les

11. selon que trois ou quatre des intersections $M_i \cap M'_j$ sont non-vides, on a un 3-module ou un 4-module.

intersections sont alors des sommets homogènes et peuvent être supprimés. Plus embêtant est le fait que il existe des unions non-triviales de 2-modules maximaux donc l'union est un 2-module.



Les 12 cas de décomposition 2-modulaires de type `complet` *ou* `linéaire`*.*

Mais le vrai obstacle rédhibitoire est la remarque de §3.2 que si $S = \{S_1, S_2\}$ est un 2-module et $M = \{M_1, M_2\}$ un 2-module de $G[S]$, M n'est pas nécessairement un 2-module de G. On peut donc définir un arbre de décomposition de façon récursive, mais on court le risque de créer des *artefacts*, des ensembles de décomposition qui ne sont pas des 2-modules du graphe de départ ! Je n'ai pas trouvé le moyen de contourner cet obstacle. Les seules familles ayant de bonnes propriétés sont 1-modules, coupes et co-coupes, 2-joints, et sesquimodules et 2-modules boiteux.

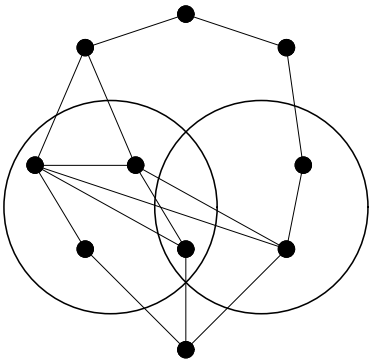
En revanche dans les conditions de la proposition 30 on a bien l'équivalence. On peut la forcer de deux façons :

– En forçant M à être inclus dans S_1 ou S_2.

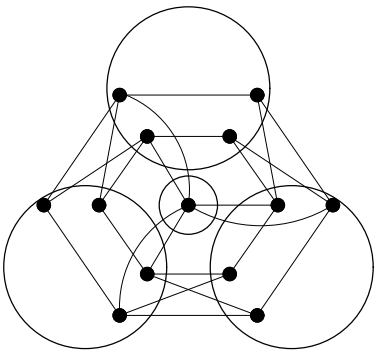
– En forçant M_1 à être inclus dans S_1 et M_2 dans S_2.

La première voie n'a rien donné. Pour la seconde en revanche il y a des résultats. On prend une bipartition $V_1 \uplus V_2$ de V et on force $M_1 \subset V_1$ et $M_2 \subset V_2$. C'est essentiellement ce qui est

fait dans le chapitre suivant, à propos de graphes bipartis, où cette bipartition est naturelle.



Les bimodules non-triviaux maximaux pour l'inclusion sont entourés. Ils sont d'intersection non-vide (c'est un conflit d'union.)



Les bimodules non-triviaux maximaux pour l'inclusion sont entourés. L'union des trois plus gros est un 2-module : c'est la seule union qui fasse un 2-module, et c'est une union non-triviale.

Deux ennuis pour la décomposition `premier`*...*

3.9.2 Décomposition k-modulaire

Le « 2 » qui préfixe *2-module* fait tout de suite penser à une généralisation naturelle de ces objets : les k-**modules**.

Définition 21
*Un k-**module** de $G = (V, E)$ est un ensemble $M = \{M_1, \dots M_k\}$ de k parties deux-à-deux disjointes de V, tel que*

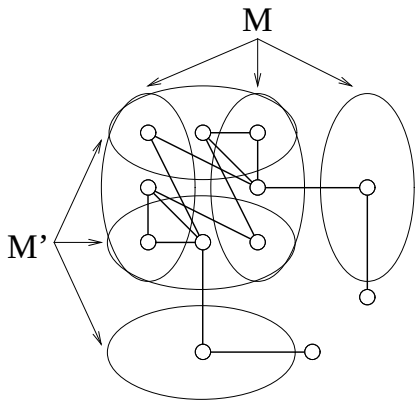
$$\forall i \, \forall x, y \in M_i \, \forall z \notin \bigcup M_i \;\; (x, z) \in E \iff (y, z) \in E$$

Malheureusement on peut observer que

Proposition 41
L'intersection de deux k-modules est, dans le pire des cas, un k^2-module.

Dans le cas des 2-modules, si l'intersection de deux 2-modules est un 3-module ou un 4-module, alors leur union est un 1-module. En se restreignant à la décomposition des graphes 1-premiers, on fait donc apparaître une décomposition parfaite, où M et $V \setminus M$ sont des 2-modules. Dès les 3-modules, cela n'est plus vrai : le contre-exemple suivant montre deux 3-modules dont

l'intersection est un 4-module (et s'échappe donc de la famille) tandis que leur union est un 3-module.



Contre-exemple pour une décomposition 3-modulaire.

Cela est, à mon avis, rédhibitoire pour un analogue du théorème 14 : il semble qu'il faille abandonner l'espoir que toutes les familles de k-modules contiennent un gros noyau partitif. L'approche des décompositions partitives semble donc inapplicable aux k-modules ; et la décomposition k-modulaire doit donc être une chose très compliquée... Il faut se restreindre à l'étude de décompositions k-modulaires particulières, telle que celle en k-joints évoquée §3.6.5 page 79.

Peut-être cependant que dans certaines généralisations des graphes bipartis (k-partis, graphes de [Bra96], ...) un mécanisme semblable à la décomposition bimodulaire du chapitre suivant ; c'est-à-dire des k-modules qui respectent une k-partition « naturelle » des sommets, peut toutefois être mis en place.

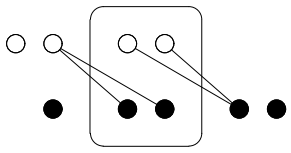
Chapitre 4

Décomposition bimodulaire

Grau, teurer Freund, ist alle Theorie und grün
des Lebens goldner Baum.

GOETHE, *Faust*

G est un graphe biparti si V peut être partitionné entre N (sommet *noirs*) et B (sommets *blancs*) de sorte que $E \subset N \times B$: il n'y a pas d'arête entre sommets de la même couleur. On notera $G = (N \uplus B, E)$ un graphe biparti. On s'intéresse dans ce chapitre à la décomposition des graphes bipartis. La décomposition modulaire est très inadéquate pour les graphes bipartis (qui ont, comme on le verra, jumeaux et composantes connexes pour seuls modules). Cela vient du fait que l'on interdit l'adjacence de sommets de la même couleur. Il faut adapter les modules aux graphes bipartis, ce qui amène à définir les *bimodules* : M est un bimodule si aucun sommet *blanc* (resp. noir) hors de M ne distingue les sommets *noirs* (resp. blancs) de M. Ainsi un sommet x blanc peut être *adjacent* à un bimodule M s'il est adjacent aux noirs de M, et bien sûr non-adjacents aux blancs de M.



Un bimodule.

Nous commençons par faire le lien entre bimodules et 2-modules : le théorème 19 établit que bimodules et coupes sont les seuls 2-modules « intéressants » (correspondants à une décomposition non-triviale). Seul §4.1 parle de 2-modules : ce chapitre peut sinon être lu indépendemment du précédent.

Devant le constat que la famille des bimodules n'est pas *partitive*, même faiblement, ($cf.$ chapitre 1), nous construisons une sous-famille, les *bimodules canoniques*, qui est faiblement partitive. Il y a donc un arbre de décomposition sous-jacent. Cette arbre correspond, comme pour

la décomposition modulaire, à des opérations de *composition* de graphes (en l'occurrence les compositions `série`, `parallèle` et `K+S`) ou bien à des ensembles de décompositions maximaux d'intersection vide. Cette famille est définie de façon récursive par des opérations de composition/décomposition. Ce travail est dans la continuité de celui de Fouquet, Giakoumakis et Vanherpe [GV97b, FGV99, Van99, GV03], qui avaient défini deux analogues des cographes : bicographes et bisplits étendus. Mais la décomposition de type `premier` leur manquait et c'est ce problème, soulevé par Jean-Marie Vanherpe lors d'un groupe de travail à Dagstuhl, qui a motivé mes recherches sur la décomposition bimodulaire puis 2-modulaire.

Ce chapitre propose un algorithme polynômial, basé sur l'approche récursive, calculant cette décomposition. Sont ensuite caractérisés les bimodules manquants, non-canoniques. Ce sont eux qui empêchent la famille des bimodules d'être partitive, en créant des *conflits*. Heureusement, ils sont peu nombreux et le théorème 25 établit qu'ils peuvent se déduire des bimodules canoniques. Ce résultat est fort : l'arbre de décomposition canonique permet de connaître *tous* les bimodules d'un graphe ! Ce chapitre se termine par quelques remarques sur la bicomplémentation, la substitution et le quotient, une application à la décomposition de matrices booléennes, et quelques pistes de réflexion.

4.1 Caractérisation des 2-modules d'un graphe biparti

La décomposition modulaire des graphes bipartis est très pauvre. En effet

Proposition 42
Les seuls modules forts d'un graphe biparti sont ses composantes connexes et ses classes de sommets jumeaux.

Démonstration: Soit M un module de G. Si M ne contient que des sommets de la même couleur, alors tous ces sommets sont faux jumeaux, puisqu'il n'y a pas d'arête entre eux. Sinon, soit $x \notin M$. Si x est voisin d'un noir de M, alors il doit être voisin d'un blanc de M : impossible, quelque soit la couleur de x. M est donc déconnecté du reste du graphe.

□

Un 2-module est une structure avec une bipartition, tout comme un graphe biparti. Il est donc naturel de définir un concept de 2-module qui soit adapté à la spécificité des graphes bipartis.

Définition 22
*Soit $M = \{M_1, M_2\}$ un 2-module d'un graphe biparti $G = (N \uplus B, E)$. M est un **bimodule** si $M_1 \subset N$ et $M_2 \subset B$, ou bien $M_1 \subset B$ et $M_2 \subset N$.*

Un bimodule sera noté $M = M_n \uplus M_b$ avec $M_b \subset N$ et $M_b \subset B$.

Définissons aussi pour l'occasion le concept de 2-module **bijumeau** : il ne contient que des sommets de la même couleur. On montre facilement que pour $M = \{M_1, M_2\}$ tous les sommets

de M_i sont jumeaux. Connaître toutes les classes de jumeaux d'un graphe (une famille de taille $O(n)$) permet de générer la famille (de taille $O(2^n)$) des bijumeaux d'un graphe.

Théorème 19
Les seuls 2-modules d'un graphe biparti sont

- *Les modules*
- *Les bijumeaux*
- *Les côtés de coupe*
- *Les bimodules*

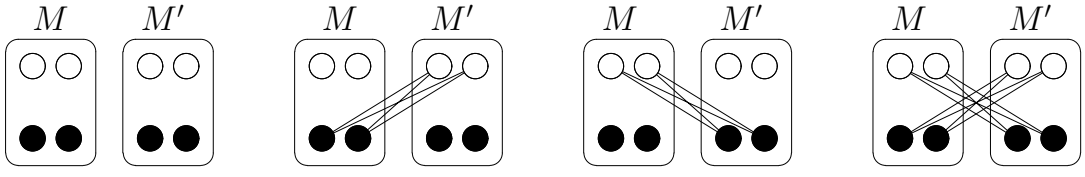
Démonstration: Pour $i = 1$ ou 2, si M_i intersecte à la fois N et B, alors un sommet $x \notin M$ ne peut être adjacent à M_i. Si M_1 et M_2 sont tous deux dans ce cas, alors M est un module, déconnecté du reste du graphe. Si un seul des deux, disons M_1, est dans ce cas, alors $(M_1, M_2, \Gamma(M_2) \setminus M, \overline{\Gamma}(M_2) \setminus M)$ est une coupe de G (qui peut être une coupe triviale). Et si aucun des deux n'est dans ce cas, alors M est un bimodule si M_1 et M_2 ont une couleur différente, et est bijumeau sinon.

□

Les décompositions en modules et en bijumeaux des graphes bipartis sont, comme nous venons de le voir, très pauvres. Restent les deux autres familles : coupes et bimodules. La décomposition en coupes d'un graphe biparti peut être assez riche (G ne contient bien sûr pas de clique, mais des étoiles et des composantes triconnexes sans restriction). La décomposition en coupes ayant déjà été étudiée [Cun82, Lan01], je n'y reviendrai pas. Nous allons nous concentrer sur la dernière famille, les bimodules.

4.2 Premières propriétés des bimodules

Vis-à-vis de l'extérieur, un bimodule (non réduit à un sommet) se comporte comme *deux* sommets, un noir et un blanc. Par abus de langage, on dira qu'un sommet $x \in N$ est **adjacent** à M si x est adjacent à M_b ; même définition pour $x \in B$. $x \in M$ est M**-universel** s'il est adjacent à $M \setminus \{x\}$ et M**-isolé** s'il est non-adjacent à $M \setminus \{x\}$; dans les deux cas il est M**-homogène**. On définit **adjacence**, **demi-adjacence** (avec une relation d'ordre $\prec$) et **non-adjacence** de bimodules comme suit :



Non-adjacence | Demi-adjacence $M \prec M'$ | Demi-adjacence $M \succ M'$ | Adjacence

La propriété suivante, vraie pour les modules mais non pour les 2-modules, ce qui explique les ennuis rencontrés au chapitre précédent, est en revanche vraie pour les bimodules :

Proposition 43
Soit $M \subset S \subset V$. Si S est un bimodule de G et M un bimodule de $G[S]$ alors M est un bimodule de G.

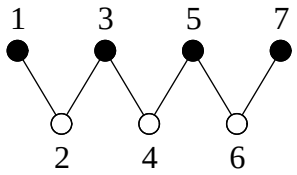
Démonstration: Corollaire de la proposition 30 page 64.

□

Proposition 44
L'intersection de deux bimodules est un bimodule.

Démonstration: trivial. □

Malheureusement, la famille des bimodules n'est pas une famille partitive. En effet, il existe des *conflits*. Par exemple, dans le P_7, $\{1,3,4\}$ et $\{4,5,7\}$ sont deux bimodules qui se chevauchent, mais leur union n'est pas un bimodule.



Un bimodule M est **conflictuel** s'il existe un bimodule N non-trivial tel que $M \circledcirc N$ mais $M \cup N$, $M \smallsetminus N$, $N \smallsetminus M$ ou $M \cap N$ n'est pas un bimodule. On peut les caractériser : en corollaire du théorème 14,

Proposition 45
Soient M et M' deux bimodules. Les seuls conflits qui peuvent exister entre M et M' sont : mineur I, union I, union II, différence I.

Démonstration: Les graphes bipartis connexes sans jumeaux sont premiers (pour la décomposition modulaire). Si G est un graphe biparti non-connexe, on le rend connexe en conservant ses bimodules en ajoutant un sommet blanc universel et un noir universel, ce qui autorise à utiliser le théorème 14. Deux bimodules ne peuvent être qu'en chevauchement *pair* ou *sommital* (voir §3.8 page 83), ce qui n'autorise que ces quatre conflits.

□

Un bimodule est **plein** si aucun module ne le chevauche. Soient $V_1, \ldots V_k$ les classes de jumeaux de G. On peut établir une relation d'équivalence entre parties de V : $A \mathcal{R} B$ ssi A et B intersectent les mêmes classes de jumeaux. Soit $\mathcal{C}$ une classe d'équivalence de $\mathcal{R}$.

Proposition 46
Si $\mathcal{C}$ contient un bimodule alors toutes les parties de $\mathcal{C}$ sont des bimodules. Dans ce cas, $\mathcal{C}$ contient un unique bimodule plein.

Démonstration: On passe d'un élément à un autre de $\mathcal{C}$ par une suite finie d'opérations consistant à ajouter ou retrancher un jumeau à un sommet. Or cette opération préserve clairement la bimodularité. Le bimodule plein est l'unique bimodule qui est l'union des classes de jumeaux, c'est donc le plus gros membre de la classe.

□

Les proposition 42 et 46 permettent de se restreindre au cas des graphes sans jumeaux de la même couleur. En effet, pour obtenir tous les bimodules d'un graphe biparti quelconque, il suffit de le décomposer composante par composante, puis on regarde tous les bimodules du quotient de chaque composante (le graphe obtenu en ne gardant qu'un sommet par classe de jumeaux). On construit facilement la classe selon $\mathcal{R}$ de chacun dans le graphe de départ. On abrégera « sans jumeaux de la même couleur » par « sans jumeaux ».

Dans toute la suite de ce chapitre on considérera donc uniquement les graphes bipartis sans jumeaux, sauf mention contraire[1].

Soit M un bimodule d'un biparti sans jumeaux. Si $|M| = 2$, M est une **paire**, un bimodule trivial formé d'un sommet noir et d'un blanc. Si $|M| = 3$, M est un **triangle noir**, un bimodule formé d'un sommet noir n et de deux blancs b et b' que seuls n distingue (il est adjacent à l'un et non à l'autre) ou un **triangle blanc**.

Le but des sections suivantes est de construire une sous-famille de la famille des bimodules, les *bimodules canoniques*, qui soit une famille faiblement partitive.

4.3 Les décompositions par composantes

Le **bicomplément** d'un graphe biparti $G = (N \uplus B, E)$ est le graphe $\overline{G}^{bip} = (N \uplus B, (N \times B) \setminus E)$.

Proposition 47
M est un bimodule de G ssi c'est un bimodule de $\overline{G}^{bip}$.

Démonstration: trivial. □

Cette proposition découle directement de la définition d'un bimodule et du bicomplément. Soient $G_1 = (N_1 \uplus B_1, E_1) \dots G_k = (N_k \uplus B_k, E_k)$ des graphes bipartis sur des ensembles disjoints de sommets. [GV97b, Van99] définissent les opération de compositions des graphes bipartis suivants :

– **composition parallèle**

$$G = (\bigcup_{i=1}^{k} N_i \uplus \bigcup_{i=1}^{k} B_i, \bigcup_{i=1}^{k} E_i)$$

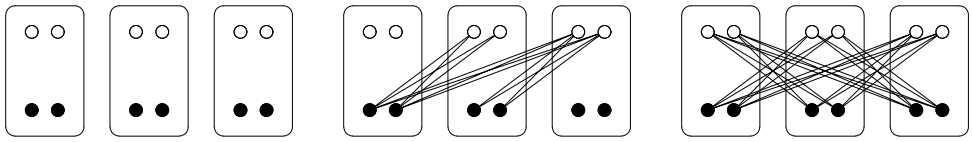
1. La raison est que il est plus facile de parler de sommet M-homogène que de module M-homogène, et surtout que les jumeaux perturbent l'unicité d'un ordre K+S (voir définition page suivante)

– **composition série**

$$G = (\bigcup_{i=1}^{k} N_i \uplus \bigcup_{i=1}^{k} B_i, \bigcup_{i=1}^{k} E_i \uplus \bigcup_{i \neq j} N_i \times B_j)$$

– **composition K+S**

$$G = (\bigcup_{i=1}^{k} N_i \uplus \bigcup_{i=1}^{k} B_i, \bigcup_{i=1}^{k} E_i \uplus \bigcup_{i < j} N_i \times B_j)$$



composition parallèle composition K+S composition série

Il correspond à ces opérations de composition des opérations de décomposition (les restrictions sont pour obtenir trois cas mutuellement exclusifs, comme cela sera ensuite prouvé) :

– Si G n'est pas connexe, sans sommet isolé et possède au moins cinq sommets, la **décomposition parallèle** de G est l'ensemble des composantes connexes de G.
– Si $\overline{G}^{bip}$ n'est pas connexe, sans sommet isolé et possède au moins cinq sommets, la **décomposition série** de G est l'ensemble des composantes connexes de $\overline{G}^{bip}$.
– Une partition $V_1 \dots V_k$ des sommets d'un graphe biparti G est une **partition K+S** si pour tout $1 \leqslant i < j \leqslant k$ les sommets noirs de V_i sont adjacents aux blancs de V_j et les blancs de V_i non-adjacents aux noirs de V_j. Les **composantes K+S** d'un graphe sont les parties de la plus fine partition K+S de G (celle avec le plus de parties) qui est unique, comme le prouve [Van99]. Si G possède plus d'une composante K+S, elles forment sa **décomposition K+S**. En particulier, tout biparti avec un sommet isolé ou universel possède une décomposition K+S. Le nom de K+S vient du fait que (mis à part le cas d'un graphe avec deux composantes K+S seulement dont un sommet isolé) le graphe peut être partitionné entre une biclique K (biparti complet) et un stable S. Plus exactement pour tout $i < k$, $G[(V_1 \cap N) \cup \ldots \cup (V_{i-1} \cap N) \cup (V_i \cap B) \cup \ldots \cup (V_k \cap B)]$ est une biclique, et $G[(V_1 \cap B) \cup \ldots \cup (V_{i-1} \cap B) \cup (V_i \cap N) \cup \ldots \cup (V_k \cap N)]$ est un stable. Réciproquement, tout graphe dont les sommets peuvent être partitionnés en une biclique et un stable [2] possède une décomposition K+S.

2. graphe appelé un *bisplit graph*, ou *graphe bi-cassé*, par analogie avec les *split graphs* ou *graphes cassés* dont les sommets sont partitionnés entre une clique et un stable. Ces graphes sont étudiés par [FJKM90].

Un graphe est **c-décomposable**[3] s'il possède une décomposition série, parallèle ou K+S. Les *ensembles de décomposition* d'un graphe c-décomposable sont ses composantes. En effet

Proposition 48
Un graphe sans jumeaux c-décomposable possède une unique décomposition : il ne peut être à la fois X-décomposable et Y-décomposable pour X,Y∈ {série, parallèle, K+S}.

Démonstration: On montre tout d'abord qu'un graphe à au moins cinq sommets et sans jumeaux ne peut être à la fois non-connexe et de bicomplément non-connexe. Soit G non-connexe, C une composante connexe de G, C_n et C_b les sommets noirs et blancs de C, et D_n et D_b les sommets noirs et blancs de $V \setminus C$. Dans $\overline{G}^{bip}$ C_n et D_b sont adjacents, de même que C_b et D_n. $\overline{G}^{bip}$ est non-connexe ssi il n'y a aucune arête entre C_n et C_b ni entre D_n et D_b : C et $V \setminus C$ sont des stables de $\overline{G}^{bip}$, donc des bicliques de G. Comme G a plus de cinq sommets l'une au moins des bicliques a plus de trois sommets, donc contient des jumeaux de la même couleur, contradiction.

Maintenant montrons que la décomposition K+S exclut les deux autres. Soit G un graphe ayant une composition K+S $C_1 \dots C_k$. Les noirs de C_i sont reliés aux blancs de C_j ssi $i < j$. Les noirs de C_1 sont reliés à tous les blancs (sauf éventuellement ceux de C_1) et les blancs de C_k à tous les noirs (sauf éventuellement ceux de C_k) : $G[V \setminus ((C_1 \cap B) \cup (C_k \cap N))]$ est connexe. Un blanc de C_1, s'il n'est relié à aucun noir de C_1, est isolé. Un noir de C_k, s'il n'est relié à aucun blanc de C_k, est isolé. Donc une composition K+S sans sommet isolé est connexe. Par bicomplémentation, une composition K+S sans sommet universel est de bicomplémentaire connexe.

□

À cause de la maximalité des composantes, une composante connexe (resp. co-connexe, K+S) n'a pas de décomposition parallèle (resp. série, K+S). C'est ainsi que Fouquet, Giakoumakis et Vanherpe [GV97b, FGV99, Van99, GV03] définissent la **décomposition canonique** des graphes c-décomposables : il s'agit d'un arbre correspondant à une famille arborescente. La racine est V et, pour chaque nœud c-décomposable, ses fils sont ses composantes. Le nœud est étiqueté `série`, `parallèle` ou `K+S` selon le type de décomposition[4].

Il est clair qu'une composante connexe de G, une composante connexe de $\overline{G}^{bip}$ ou une composante K+S sont des bimodules. Ils ne se chevauchent pas entre eux. Nous pouvons enfin arriver à la définition attendue :

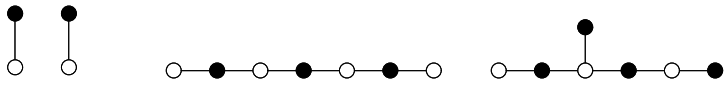
Définition 23
*Soit G un biparti c-décomposable. Les **bimodules canoniques forts maximaux** de G sont ses composantes.*

3. Abbréviation de « décomposable par composantes », mais aussi de « canoniquement décomposable ». Cette définition n'a rien à voir avec les *c-decomposable graphs* de [FJ90] et d'autres.

4. Je présente ici une légère variante : [FGV99, Van99] décomposent les graphes avec jumeaux et le $2K_2$, ce qui nécessite d'avoir des priorités entre les décompositions, qui ne sont plus mutuellement exclusives.

4.4 Décomposition des c-indécomposables

Un graphe G qui n'est pas c-décomposable est c-indécomposable.



$2K_2$ P_7 $Star_{123}$

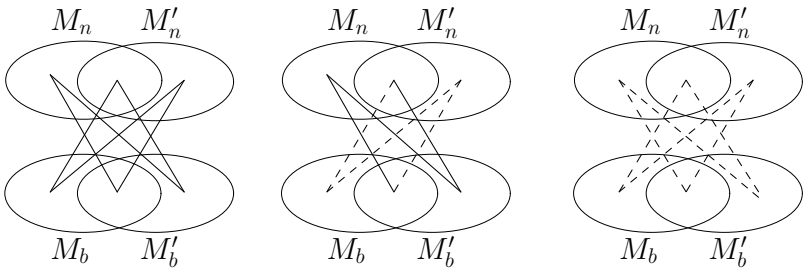
Les trois graphes c-indécomposables à moins de huit sommets.

Nous allons définir une famille de bimodules d'intersection vide de G à partir des bimodules maximaux de G. Ces bimodules serviront d'ensembles de décomposition, comme dans le cas précédent.

Proposition 49
Soient G un graphe sans jumeaux c-indécomposable et $M_1 \dots M_k$ les bimodules non-triviaux maximaux pour l'inclusion de G. Si $M_i \circledcirc M_j$ alors

1. *soit $M_i \cap M_j = \{v\}$, où v est un sommet M_i-homogène et M_j-homogène,*
2. *soit M_i et M_j sont des triangles de la même couleur.*

Démonstration: G a au moins six sommets : en effet tous les bimodules de $2K_2$ sont triviaux et les autres graphes à moins de six sommets sont c-décomposables. Soient M et M' deux bimodules non-triviaux, maximaux pour l'inclusion et qui se chevauchent. Si $M \cup M' = V(G)$ alors on est dans l'un des cas suivants, qui contredit que G soit c-indécomposable (considérer des graphes à plus de quatre sommets élimine le cas de deux bimodules triviaux) :



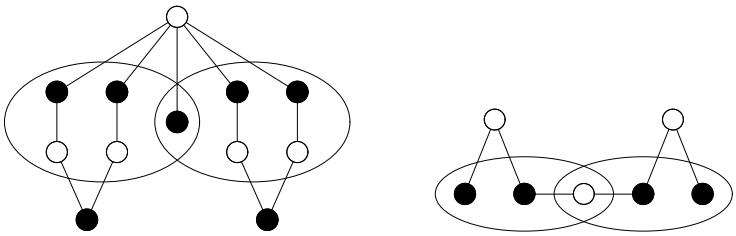
composition série composition K+S composition parallèle

Donc $M \cup M'$ n'est pas un bimodule (sans quoi il serait différent de V donc non-trivial, ce qui contredit la maximalité de M et M') : M et M' sont en *conflit d'union* (voir chapitre 3). D'après le théorème de décomposition 2-modulaire (théorème 14 page 66) :

– soit c'est le conflit d'union I, et leur intersection est un sommet v M-homogène et M'-homogène, ce qui vérifie le point 1 du théorème

– soit c'est le conflit d'union II, et les deux 2-modules impliqués sont boiteux. Or, un bimodule boiteux est un triangle. Cela vérifie le point 2.

□



Conflit d'union I Conflit d'union II

Exemples des deux conflits d'union.

Soient $M_1 \dots M_k$ les bimodules de G non-triviaux maximaux pour l'inclusion. M_i est *conflictuel* s'il chevauche M_j, $j \neq i$. Si $|M_i| \geqslant 4$, M_i a un conflit d'union I, donc $M_i \cap M_j = \{v\}$, et v est M_i-homogène : $M_i \setminus \{v\}$ est un bimodule. Dans ce cas v est un **sommet conflictuel**[5] de M_i.

Proposition 50
Un bimodule M possède au plus deux sommets M-homogènes.

Démonstration: Puisque G est un graphe sans jumeaux de la même couleur, un bimodule M possède au plus quatre sommets M-homogènes : (1) un sommet blanc M-universel, (2) un sommet blanc M-isolé, (3) un sommet noir M-universel, (4) un sommet noir M-isolé. Clairement (1) et (4) sont mutuellement exclusifs, de même que le sont (2) et (3).

□

D'après cette proposition, un bimodule maximal possède M au plus deux sommets conflictuels. Comme un sommet conflictuel v est M-homogène, $M \setminus \{v\}$ est un bimodule. Soit M' le bimodule obtenu en ôtant de M ses éventuels sommets conflictuels. Puisque M est maximal, tout bimodule N chevauchant M' est inclus dans M, donc $M' \setminus N$ ne peut contenir que les sommets conflictuels de M. Construire M' pour tout M *règle* les conflits d'union I. Quant aux conflit d'union II, on les règle plus simplement encore : on ne considère pas les triangles conflictuels ! On définit donc :

Définition 24
*Soit G biparti c-indécomposable. M est un **bimodule canonique fort maximal** de G si*

– *M est un bimodule non-trivial maximal* non-conflictuel *de G, ou*

5. le sommet conflictuel est l'intersection de deux *maximaux* en conflit d'*union I*.

- *il existe N, bimodule non-trivial maximal à au moins quatre sommets, tel que $N = M \uplus \{v\}$ ou $N = M \uplus \{u, v\}$, où u (resp. u et v) son (resp. ses deux) sommet(s) conflictuel(s), ou*
- *$M = \{v\}$ et tout bimodule contenant v est trivial ou est un triangle conflictuel, ou bien v est un sommet conflictuel.*

Cette famille est une partition de V. Qui plus est, pour tout canonique M il existe un bimodule maximal N contenant M et au plus deux autres sommets : les canoniques sont « presque » les maximaux.

4.5 Le théorème de décomposition canonique

Nous allons définir par récurrence une sous-famille des bimodules d'un graphe biparti (sans jumeaux) , les bimodules **canoniques**. Les définitions 23 et 24 construisent les bimodules canoniques forts maximaux $M_1 \dots M_k$ de G. Les **bimodules canoniques forts** de G sont ses bimodules canoniques forts maximaux, ainsi que les bimodules canoniques forts de $G[M_i]$ pour tout $|M_i| \geqslant 2$. D'après la proposition 43 ce sont bien des bimodules de G.

La famille des bimodules canoniques forts est arborescente, puisqu'ils ne peuvent se chevaucher. Soit $\mathcal{T}_G$ (ou $\mathcal{T}$ sans ambiguïté) leur arbre d'inclusion. Un bimodule canonique fort M est identifié avec un nœud de $\mathcal{T}$. Il est étiqueté `série` (resp. `parallèle`, `K+S`) si $G[M]$ admet une décomposition série (resp. parallèle, K+S) et `premier` si $G[M]$ est c-indécomposable. Si M est `K+S` ses fils sont ordonnés selon l'ordre K+S.

Définition 25
*Un **bimodule canonique** est un bimodule canonique fort, ou bien l'union de plusieurs bimodules canoniques forts fils d'un nœud* `série`, `parallèle` *ou* `K+S`*. Pour les fils d'un* `K+S` *on impose la condition supplémentaire que ces fils soient consécutifs selon l'ordre K+S.*

Théorème 20
La famille des bimodules canoniques est faiblement partitive. Tout bimodule canonique est un bimodule.

Démonstration: Par construction la famille est faiblement partitive, les nœuds `série` ou `parallèle` étant `complet` et les nœuds `K+S` étant `linéaire`. Les bimodules canoniques forts sont, comme on l'a vu, des bimodules. Soit M un bimodule canonique fort et $M_1 \dots M_k$ ses fils. Seul un sommet de M peut distinguer une union de fils de M. Si M est `série` ou `parallèle`, M_i ne peut distinguer $M_j \cup M_k$ car il lui est adjacent ou non selon le type de la composition. Si M est `K+S`, pour $i < j < k$, M_i ne distingue pas $M_j \cup M_k$ et M_k ne distingue pas $M_i \cup M_j$. Donc un bimodule canonique faible est un bimodule.

□

Donc l'union, l'intersection et la différence de deux bimodules canoniques est un bimodule, de plus c'est un bimodule canonique.

Formulation récursive du théorème

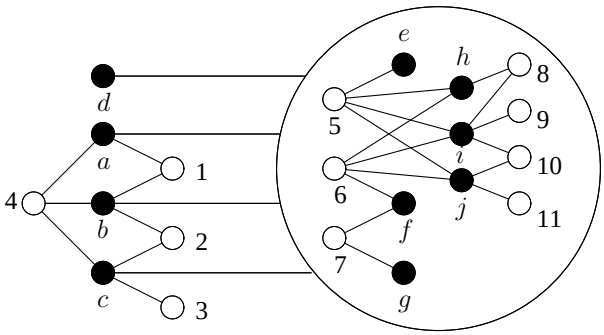
La décomposition modulaire peut être formulée par un théorème non-récursif (théorème 11 page 55 : c'est une famille partitive. Donc structure d'arbre, etc.) ou de façon récursive (théorème 7 page 48 : caractérisation en quatre cas possibles des modules forts maximaux d'un graphes). Nous venons de formuler la décomposition canonique de façon non-récursive. La formulation récursive est plus naturelle. C'est celle qui a été adoptée par Fouquet Giakoumakis et Vanherpe pour la décomposition des bicographes et graphes bisplit étendus (voir §4.7).

Théorème 21
Soit G un graphe biparti sans jumeaux. Une et une seule des assertions suivantes est vraie :

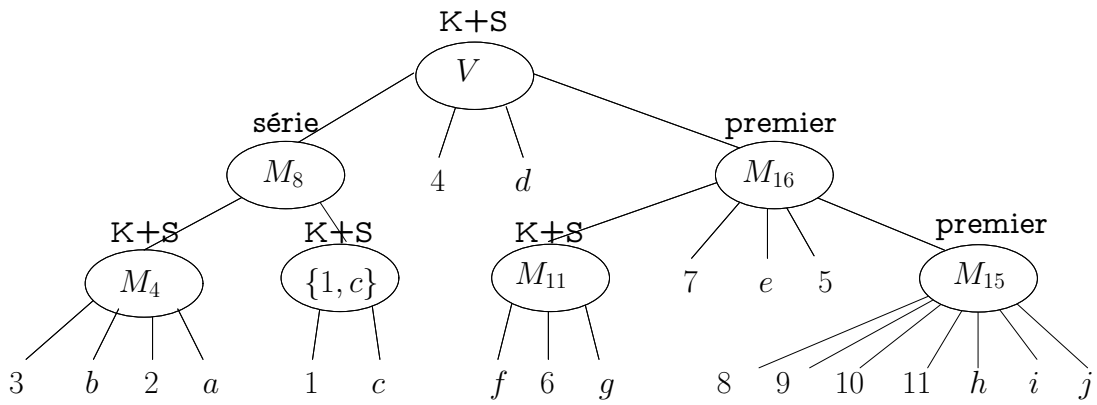
1. *G n'a qu'un sommet.*
2. *G n'est pas connexe, sans sommet isolé et a plus de cinq sommets. Les bimodules canoniques forts maximaux de G sont ses composantes connexes.*
3. *$\overline{G}^{bip}$ n'est pas connexe, sans sommet isolé et a plus de cinq sommets. Les bimodules canoniques forts maximaux de G sont les composantes connexes de $\overline{G}^{bip}$.*
4. *G est une composition K+S. Les bimodules canoniques forts maximaux de G sont ses composantes K+S.*
5. *sinon, les bimodules canoniques maximaux de G sont forts.*

4.6 Exemple

Voyons un exemple. Prenons le graphe suivant :



Son arbre de décomposition canonique est :



On calcule la liste de tous les bimodules à plus de deux sommets du graphe :

$M_1 = \{a, 2, 4\}$ $M_2 = \{b, 2, 3\}$
$M_3 = \{a, b, 2\}$ $M_4 = \{a, b, 2, 3\}$
$M_5 = \{a, b, 2, 4\}$ $M_6 = \{a, b, 2, 3, 4\}$
$M_7 = \{c, 1, 4\}$ $M_8 = \{a, b, c, 1, 2, 3\}$
$M_9 = \{a, b, c, 1, 2, 3, 4\}$ $M_{10} = \{a, b, c, d, 1, 2, 3, 4\}$
$M_{11} = \{f, g, 6\}$ $M_{12} = \{h, 8, 9\}$
$M_{13} = \{i, 10, 11\}$ $M_{14} = \{j, 9, 10\}$
$M_{15} = \{h, i, j, 8, 9, 10, 11\}$ $M_{16} = \{e, f, g, h, i, j, 5, 6, 7, 8, 9, 10, 11\}$
$M_{17} = \{d, e, f, g, h, i, j, 5, 6, 7, 8, 9, 10, 11\}$ $M_{18} = \{d, e, f, g, h, i, j, 4, 5, 6, 7, 8, 9, 10, 11\}$
$V = \{a, b, c, d, e, f, g, h, i, j, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$

Les bimodules non-triviaux non-canoniques (voit §4.8) sont M_1, M_5, M_6 et M_7, qui sont des trans-KS-série[6] de V et M_8, ainsi que trois trans-premiers (triangulaires) de M_{15} : M_{12}, M_{13} et M_{14}.

4.7 Classes de graphes bipartis

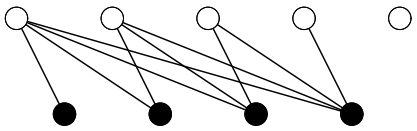
4.7.1 Graphes de degrés inégaux

Un graphe biparti est **de degrés inégaux** si deux sommets noirs n'ont jamais même degré, et deux sommets blancs non plus. Cela implique qu'il est sans jumeaux[7]. On montre grâce au théorème du colombier[8] que la différence entre le nombre de blancs et le nombre de noirs est d'au plus 1.

6. le lecteur ne pourra comprendre ce mot mystérieux qu'après la lecture de la section 4.8. Mais j'ai jugé utile de présenter cet exemple *avant* cette section, où sont définis les trans-KS et trans-premiers...

7. on rappelle que « sans jumeaux » sous-entend « de la même couleur ».

8. *pigeon-hole principle* : si on prend n nombres entiers compris entre 1 et $n-1$ inclus, il en existe deux identiques



Un graphe de degrés inégaux.

Proposition 51
Un graphe biparti est de degrés inégaux si et seulement si toutes ses composantes K+S ont un seul sommet.

Démonstration: Le « seulement si » vient immédiatement de la définition d'une composition K+S. Réciproquement si G est de degrés inégaux, il possède un sommet isolé ou bien un universel. On ôte ce sommet (qui fait une composante K+S) et on termine par récurrence.

□

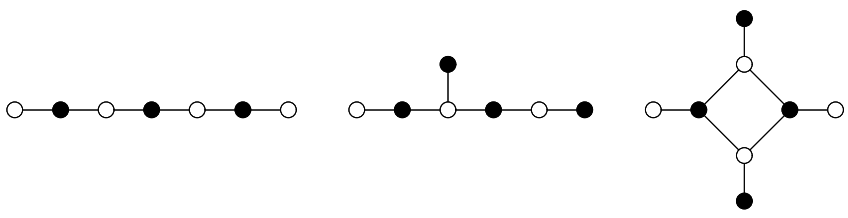
4.7.2 Bicographes

Un bicographe est un graphe dont la décomposition canonique ne comporte que des nœuds `série` et `parallèle`. La classe est définie préalablement à la décomposition canonique chez Giakoumakis & Vanherpe, comme suit :

Définition 26
*La classe des **bicographes** est la plus petite classe de graphes contenant le graphe à un sommet et fermée pour les opérations d'union disjointe et de bicomplémentation.*

Cette définition ne diffère de celle des cographes que dans le « bi » qui précède le dernier mot. On ne sera donc pas surpris de trouver des ressemblances avec les cographes. En particulier, puisque c'est une classe héréditaire, elle peut être définie par sous-graphes exclus :

Théorème 22 [GV97b, Van99]
Les bicographes sont les graphes bipartis sans P_7, $Star_{123}$ et Sun_4 induits.



P_7 $Star_{123}$ Sun_4

4.7.3 Bisplits étendus

La classe des bisplits étendus (*weak-bisplit*) est introduite par Fouquet, Giakoumakis et Vanherpe pour généraliser les bicographes, en autorisant en plus la décomposition `K+S`.

Définition 27
*Un graphe biparti G est **bisplit étendu** si tout sous-graphe induit H de G est c-décomposable.*

Cette classe peut aussi être caractérisée par sous-graphes exclus :

Théorème 23 [FGV99, Van99]
Les bicographes sont les graphes bipartis sans P_7 ni $Star_{123}$.

La *cliquewidth* des graphes bisplit étendus est d'au plus 4 [FGV99]. [Van99] fournit par ailleurs un algorithme en $O(n+m)$ de reconnaissance des bisplit étendus, produisant l'arbre de décomposition canonique pour les membres de la classe (il reconnaît donc au passage les bicographes). Il est incrémental et s'inspire de l'algorithme de [CPS85] pour les cographes. Il serait intéressant de construire un algorithme décrémental de reconnaissance de cette classe, en trouvant un concept similaire à l'élagage de jumeaux.

4.7.4 Graphes sans $Star_{123}$

Lozin étend la caractérisation par sous-graphes exclus de la classe précédente et prouve un théorème que l'on peut reformuler ainsi :

Théorème 24 *[Loz02a]*
G est graphe biparti sans $Star_{123}$ induit si et seulement si tous les membres `premier` *de l'arbre de décomposition canonique de G induisent des C_n ou des P_n, avec $n \geqslant 7$, ou leur complément au sens biparti.*

En corollaire, tous les graphes sans $Star_{123}$ sont représentés de façon unique par leur arbre de décomposition et donc stockables en espace $O(n)$. De plus Lozin montre que la *cliquewidth* de ces graphes est d'au plus 5, ce qui ouvre la porte aux algorithmes polynômiaux pour les problèmes MSOL.

Par exemple, pour la classe (plus petite) des graphes sans $Star_{123}$ et sans Sun_4, Lozin [Loz02b] montre que le problème de trouver un couplage induit maximal (exhiber un nK_2 induit maximisant n), NP-complet en général, peut être résolu en $O(n^3)$.

4.8 Bimodules non-canoniques

Forts de cette décomposition canonique, nous pouvons nous demander quels sont les bimodules « laissés pour compte » de la décomposition canonique. Or, ils sont fort peu nombreux, ce qui montre la puissance de cette décomposition. Nous commençons par les caractériser, puis

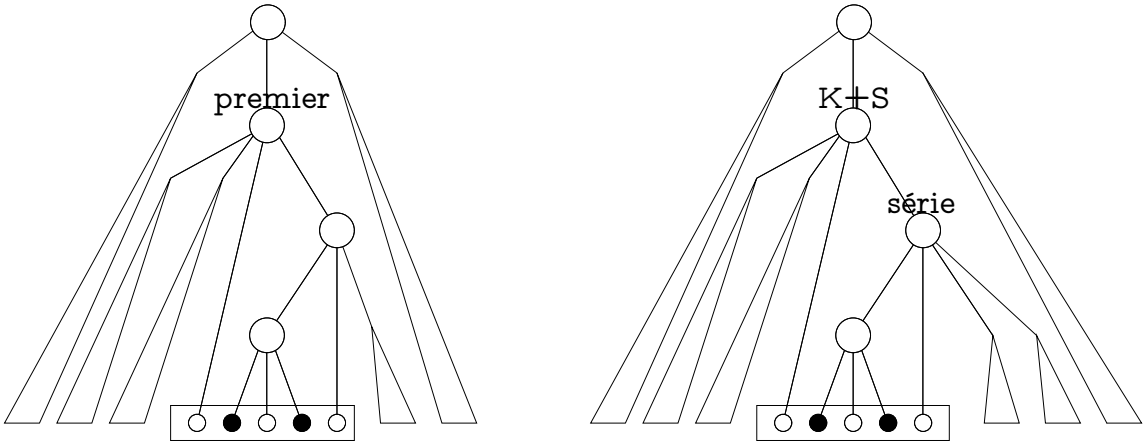
montrons comment on peut les lire sur l'arbre de décomposition canonique. L'arbre de décomposition canonique représente donc *tous* les bimodules à plus de trois sommets d'un graphe biparti ! Ce résultat fort justifie amplement la définition de la décomposition canonique.

4.8.1 Théorème de caractérisation

La première question à se poser est *quels bimodules peuvent être non-canoniques ?*

$M \subset V$ est **trans-premier** si le plus petit bimodule canonique fort P qui contienne M est `premier` et si M intersecte au plus un fils non-singleton de P.

$M \subset V$ est **trans-K+S-série** si le plus petit bimodule canonique fort F qui contienne M est `K+S` et si M intersecte deux composantes K+S consécutives de F, l'une étant un singleton et l'autre un bimodule fort `série`. On définit de même les **trans-K+S-parallèle**.



trans-premier trans-K+S-série

Théorème 25
Les bimodules non-canoniques d'un graphe sont :

- *des paires (bimodules triviaux)*
- *des trans-K+S-série et trans-K+S-parallèle*
- *des trans-premiers*

Démonstration: Nous allons distinguer deux cas : les non-canoniques ne chevauchant aucun canonique fort, et les autres.

Lemme 28 *Un bimodule qui ne chevauche aucun bimodule canonique fort est canonique ou trans-premier.*

Démonstration: D'après la proposition 3 page 26, ce bimodule (appelons-le M) est une union de bimodules canoniques forts frères. Soit N le plus petit bimodule canonique fort contenant

M. Si N est `série` ou `parallèle`, M est canonique par définition. Si N est `K+S`, M doit être une union de frères consécutifs selon l'ordre K+S : une composante K+S intercalée distinguerait en effet M. Donc M est canonique. Voyons si M est premier. Si $M = N$ le lemme est vrai. Si M intersecte plusieurs bimodules non-triviaux maximaux de $G[M]$ alors M est un singleton, contenant seulement un sommet conflictuel. Si M intersecte un seul bimodule N' non-trivial maximal de $G[M]$, M est formé de N' et d'au moins un et au plus deux sommets N'-homogènes : il est trans-premier. Et si M n'intersecte aucun bimodule non-trivial maximal de $G[M]$, M est une paire ou un triangle, et est trans-premier.

□

Cela prouve le théorème pour un bimodule non-canonique qui ne chevauche aucun canonique fort. Il reste à traiter le cas des bimodules chevauchant un bimodule canonique fort, qui sont de ce fait non-canoniques. Le théorème 21 fournit récursivement les bimodules canoniques forts : dans chacun des quatre cas non-terminaux du théorème nous verrons quels bimodules chevauchent les canoniques forts.

Lemme 29 (cas 2) *Si G est non-connexe sans sommet isolé, aucun bimodule non-trivial ne chevauche les composantes connexes de G.*

Démonstration: Soit un bimodule M chevauchant une composante connexe C de G. M intersecte une autre composante connexe C'. Supposons que M ait un sommet blanc dans C et un dans C'. S'il existe un sommet noir x de C qui n'appartient pas à M, alors il existe un sommet noir y de C qui n'appartient pas à M et qui est adjacent à un blanc de M (car C est connexe, donc un chemin relie x à M). y distingue donc M, puisqu'il ne peut être adjacent aux blancs de $M \cap C'$, contradiction. Donc M contient tous les noirs de C. On prouve de la même façon que si un noir de C' est hors de M alors il en existe un qui distingue M, contradiction. Or chaque composante contient à la fois des sommets noirs et des blancs, puisque le graphe est sans jumeaux et sans sommet isolé. Donc tous les noirs de C et C' appartiennent à M. Si un blanc x de C est hors de M, il en existe un y hors de M et adjacent aux noirs de M, qui distingue M, contradiction.

Donc si les blancs de M intersectent plusieurs composantes (la même preuve vaut avec les noirs) M doit contenir ces composantes et est une union de composantes. Sinon le seule façon pour M de chevaucher une composante est d'avoir tous ses sommets blancs dans une composante et tous ses noirs dans une autre. Comme G est sans jumeaux, M est une paire, donc trivial.

□

Lemme 30 (cas 3) *Si $\overline{G}^{bip}$ non connexe sans sommet isolé, aucun bimodule non-trivial ne chevauche les composantes connexes de $\overline{G}^{bip}$.*

Démonstration: D'après la proposition 47, M est un bimodule de G ssi c'est un bimodule de $\overline{G}^{bip}$. On se ramène au cas précédent par bicomplémentation.

□

Lemme 31 (cas 4) *Si G est une composition K+S, tout bimodule non-canonique non inclus dans une composante de G est un trans-K+S-série ou un trans-K+S-parallèle.*

Démonstration: Soit M un bimodule chevauchant une composante C de la décomposition K+S.

Remarquons qu'à cause de la maximalité des composantes, une composante K+S ne peut avoir de sommet C-homogène. Or si $C \setminus M = \{x\}$, x est C-homogène, puisque adjacent ou non-adjacent à M. De même si $C \cap M = \{x\}$, comme C et M sont deux 2-modules qui, d'après le théorème de décomposition 2-modulaire (théorème 14 page 66) ne peuvent être en conflit de différence, $C \setminus M$ est un bimodule. Là encore x est alors C-homogène.

C contient donc au moins deux sommets de M et deux sommet n'appartenant pas à M. Comme G est un graphe sans jumeaux, il existe dans C un sommet noir v_1 et un blanc v_2 appartenant à M, ainsi qu'un noir v_3 et un blanc v_4 n'appartenant pas à M.

Supposons que $|M \setminus C| \geqslant 2$. Comme on a aussi $|C \setminus M| \geqslant 2$, les 2-modules M et C ne peuvent être en conflit de différence, donc $P = C \setminus M$ et $Q = M \setminus C$ sont des 2-modules (et même des bimodules). $R = M \cap C$ est également un bimodule (proposition 44). Comme G est K+S, C et Q sont demi-adjacents. Donc P et Q sont demi-adjacents. Mais $M = Q \cup R$ est un bimodule. Donc P et R sont demi-adjacents. Cela contredit le fait que C soit une composante K+S, puisqu'on pourrait la scinder en deux composantes P et R, contradiction.

Donc $M \setminus C = \{v\}$. Soit C' la composante K+S contenant v. On a $C' \cap M = \{v\}$. On vient de voir (avec C au lieu de C') que si $C' \between M$ alors v est C'-homogène, contradiction. Donc $C' = \{v\}$.

Montrons que C et C' se suivent dans l'ordre K+S. Supposons sans perte de généralité que v soit noir. On a vu que $M \cap C$ contient un sommet noir v'. Tout sommet blanc d'une composante K+S intercalée entre C et C' distinguerait v de v' et violerait la bimodularité de M. Et toutes les composantes placées entre C et C' ne peuvent pas contenir que des sommets noirs, qui seraient jumeaux de M. Donc C et C' se suivent.

Enfin il reste à monter que C est une composition série ou parallèle. C est l'union de deux bimodules P et Q. Si P ou Q est trivial, alors C contient un sommet C-homogène. Donc C est c-décomposable. Mais sa décomposition de peut être K+S à cause de la maximalité des composantes de G. Donc elle est série ou parallèle et G est, respectivement, un trans-K+S-série ou un trans-K+S-parallèle.

□

Lemme 32 (cas 5) *Si G est c-indécomposable, tout bimodule chevauchant un des bimodules forts maximaux de G est trans-premier.*

Démonstration: Soit N un des bimodules forts maximaux de G et M un bimodule chevauchant N. N n'est pas trivial. Il existe donc un unique bimodule maximal N' de G contenant N avec $|N' \setminus N| \leqslant 2$, par construction des canoniques. Comme $M \neq V$, $M \subsetneq N'$: N est trans-premier.

□

□

4.8.2 Réciproque : un codage en $O(n \log n)$ de tous les bimodules

Nous savons maintenant que les bimodules non-canoniques et non-triviaux sont des trans-premiers et des trans-K+S. Nous allons maintenant définir un codage permettant de représenter tous les bimodules non-canoniques de G. Ce codage est une modification (par marquage supplémentaire) de l'arbre $\mathcal{T}$ qui représente les bimodules canoniques de G : $\mathcal{T}$ ainsi marqué représentera *tous* les bimodules de G à plus de trois sommets, dans un espace $O(n)$! Le terme $O(n \log n)$ vient des bimodules à trois sommets.

Définition 28
*Soit N un bimodule canonique fort et P son père dans $\mathcal{T}$. $v \in P \setminus N$ est un sommet N-**augmentant** si $N \cup \{v\}$ est un bimodule.*

De la définition des bimodules canoniques forts on déduit :

- si P est `série` ou `parallèle` N n'a pas de sommet augmentant, car toute composante de $G[M]$ a au moins 2 sommets.
- si P est `premier` les sommets augmentants associés à N sont ceux que $N' \setminus N$, où N' est le bimodule non-trivial maximal contenant N. N' a un conflit d'union I, et ces sommets, au nombre de un ou deux, sont conflictuels.
- si P est `K+S` alors les sommets augmentants associés à N sont celui de la composante singleton qui précède N (si elle existe) et celui de la composante singleton qui suit N (si elle existe).

Théorème 26
M, $|M| \geqslant 4$, est un bimodule non-canonique si et seulement si il existe un bimodule canonique N tel que

- *$M \cap N$ est un bimodule canonique ;*
- *$M \setminus N$ est constitué de un ou deux sommets N-augmentants ;*
- *si $M \setminus N$ contient un noir N-universel alors tous les blancs de $N \cap M$ sont adjacents à $M \setminus N$;*
- *si $M \setminus N$ contient un blanc N-universel alors tous les blancs de $N \cap M$ sont adjacents à $M \setminus N$;*
- *si $M \setminus N$ contient un noir N-isolé alors tous les blancs de $N \cap M$ sont adjacents à $M \setminus N$;*
- *si $M \setminus N$ contient un blanc N-isolé alors tous les blancs de $N \cap M$ sont adjacents à $M \setminus N$.*

Démonstration: Le « seulement si » vient du théorème 25 : le fait d'être un trans-premier ou un trans-K+S implique les deux premiers alinéa. Les quatre derniers viennent simplement du fait que M est un bimodule.

Réciproquement montrons que tout M vérifiant ces conditions est un bimodule non-canonique. Soient u et v les sommets N-augmentants de N. Si N n'a qu'un sommet augmentant il est commode de prendre $u = v$. M est un bimodule : un sommet $x \notin N$ ne peut distinguer M,

puisque $M \subset (N \cup \{u, v\})$. Et $x \in N$ ne peut distinguer M, car les quatre alinéas terminant le théorème l'empêchent.

Si $M \circledcirc N$ alors M est non-canonique, car nul bimodule canonique ne chevauche de bimodule canonique fort. Et si $N \subsetneq M$ et le père P de N dans $\mathcal{T}$ est `premier`, alors $N \subsetneq M \subsetneq P$ car $P \diagdown N$ ne peut contenir seulement les sommets N-augmentant sous peine que P ait une décomposition K+S. M est donc non-canonique.

□

Définition 29
Soit N un bimodule canonique fort et P son père dans $\mathcal{T}$. N est

- *marqué $\overline{B}$ si aucun blanc de N n'est adjacent à un noir de $P \diagdown N$;*
- *marqué B si tous les blancs de N sont adjacents à tous les noirs de $P \diagdown N$;*
- *marqué $\overline{N}$ si aucun noir de N n'est adjacent à un blanc de $P \diagdown N$;*
- *marqué N si tous les noirs de N sont adjacents à tous les blancs de $P \diagdown N$.*

M est également marqué uT pour tout sommet u M-augmentant, avec $T \in \{B, \overline{B}, N, \overline{N}\}$ indiquant le type du sommet.

Proposition 52
Soit $N \subset M$ deux bimodules canoniques. Les blancs de N sont tous adjacents aux noirs de $M \diagdown N$ (resp. blancs... non-adjacents, noirs...adjacents, noirs...non-adjacents) si tous les bimodules du chemin reliant N (inclus) à M (exclus) dans $\mathcal{T}$ sont marqués B (resp. $\overline{B}$, N, $\overline{N}$).

Démonstration: trivial. □

Théorème 27
Un codage de taille $O(n \log n)$ permet de connaître[9] *tous les bimodules d'un graphe.*

Démonstration: Nous allons distinguer quatre types de bimodules : paires, canoniques, triangles et autres non-canoniques.

Les paires sont en quantité $O(n^2)$, et toutes sont des bimodules : il n'est nul besoin de les marquer pour les reconnaître. L'arbre de décomposition bimodulaire est une structure de taille $O(n)$ permettant de connaître tous les bimodules canoniques.

Lemme 33 *Il y a $O(n \log n)$ bimodules triangulaires dans un biparti sans jumeaux.*

Démonstration: Nous allons montrer qu'un graphe biparti possède $O(n \log n)$ *triangles noirs*[10], l'autre preuve étant similaire. Soit x un sommet noir. On prend un sommet $u \in \Gamma(x)$. Il existe *au plus un* sommet noir $v \in \overline{\Gamma}(x)$ tel que $\{x, u, v\}$ soit un bimodule : s'il en existait plusieurs, ils seraient jumeaux. Donc il y a au plus $min(|\Gamma(x)|, \overline{\Gamma}(x))$ triangles noirs contenant x. Soit $\{x', u', v'\}$, $x' \neq x$, un autre triangle noir. On a $\{u', v'\} \subset \overline{\Gamma}(x)$ ou $\{u', v'\} \subset \Gamma(x)$, sans quoi

9. connaître les bimodules, c'est pouvoir les générer tous, mais aussi tester juste en ayant cette structure sous les yeux si un ensemble donné est un bimodule ou non.

10. rappel : bimodules formés d'un sommet noir et de deux blancs.

x distinguerait ce bimodule. Le **support** d'un triangle noir est ses deux sommets blancs. Soit $tri(S)$ le nombre de triangles noirs dont le support est S, et soit $f : \mathbb{N} \to \mathbb{N}$ telle que $f(n)$ est le nombre *maximal* de triangles noir donc un ensemble de taille n peut être le support. On a $f(0) = f(1) = 0$ et $f(2) = 1$. D'après ce que l'on vient de voir, pour un sommet x donné, pour tout $S \subset B$,

$$tri(S) \leqslant min(S \cap \Gamma(x), S \cap \overline{\Gamma}(x)) + tri(S \cap \Gamma(x)) + tri(S \cap \overline{\Gamma}(x))$$

et puisque cette équation vaut pour tout x coupant S en $S \cap \Gamma(x)$ et $S \cap \overline{\Gamma}(x)$ on a

$$f(n) \leqslant max\{min(n_1, n_2) + f(n_1) + f(n_2) \mid n_1 + n_2 = n\}$$

Supposons sans perte de généralité $n_1 \leqslant n_2$. On a donc $n_1 \leqslant \frac{n}{2} \leqslant n_2 \leqslant n$; et donc $\log(n_1) \leqslant \log(n) - 1$ et $\log(n_2) \leqslant \log(n)$. Démontrons par induction le théorème : on suppose que, pour tout $x < n$, $f(x) \leqslant x \log(x)$. Donc

$$f(n) \leqslant max\{n_1 + n_1(\log(n) - 1) + n_2 \log(n) \mid n_1 + n_2 = n\}$$

Ce qui donne $f(n) \leqslant n \log n$ puisque $n = n_1 + n_2$.

□

Une liste de taille $O(n \log n)$ suffit donc à représenter tous les triangles d'un graphe. Intéressons-nous enfin aux bimodules non-canoniques à plus de quatre sommets. Pour les retrouver il suffit, d'après le théorème précédent, de savoir identifier les sommets augmentants et les bimodules dont les blancs ou les noirs sont isolés ou universels par-rapport au reste du graphe. Or, c'est précisément ce que nous permet de faire le marquage que nous avons placé sur $\mathcal{T}$, grâce à la proposition 52. Il suffit de dire qu'un sommet M-augmenant pour un bimodule canonique fort M est N-augmentant pour tout bimodule canonique faible N qui est une union de composantes de $G[M]$.

□

Discutons sur le $n \log n$ de ce théorème. Il vient uniquement des triangles. Malheureusement, il existe des graphes bipartis sans jumeaux ayant plus de n triangles et qui atteignent cette borne. Le **biparti hypercubique** est un graphe biparti qui possède m sommets noirs, 2^m blancs, et tel que l'écriture en binaire du numéro d'un sommet blanc est son vecteur d'adjacence (le sommet blanc n° 0 est isolé, le blanc n° 1 uniquement adjacent au noir n° 0, etc.) Deux blancs font partie d'un triangle si et seulement si leurs numéros diffèrent de 1 bit : il y a $m2^{m-1} = \Omega(n \log n)$ triangles dans le graphe.

Exemple

Revenons à l'exemple de §4.6. M_8 possède un augmentant blanc universel : 4. Comme M_8 est `série`, on sait que toute union U de fils de M_8 donnera un bimodule dont les blancs seront adjacents à $M_8 \setminus U$. Comme il n'y a que deux composantes, les unions possibles sont M_4 (et $M_4 \cup$

$\{4\}$ donne le bimodule non-canonique M_6), $\{1, c\}$, ce qui conduit au bimodule non-canonique M_7 (qui est par ailleurs un triangle) et l'union des deux, égale à M_8 donc ne vérifiant pas le premier alinéa du théorème précédent. Par ailleurs M_4 étant K+S, tout bimodule formé d'une union des dernières composantes sera constituée de blancs tous adjacents au reste. La seule union non-triviale est celle des trois dernières composantes, M_3, dont l'union avec 4 donne M_5.

Le seul autre bimodule possédant un sommet augmentant est M_{16}. Comme il n'est pas `série`, on sait qu'aucun bimodule inclus dans M_{16} ne pourra vérifier la condition du théorème 26 : il ne peut être trans-K+S-série.

4.9 Algorithme de décomposition

La formulation récursive du théorème de décomposition donne l'idée d'un algorithme polynômial simple de décomposition. On peut en effet reconnaître en temps polynômial auquel des cinq cas du théorème 21 correspond un graphe biparti donné, et calculer ses bimodules forts maximaux. Il n'y a plus qu'à itérer récursivement sur chacun des graphes induits par ces bimodules : on construit de façon descendante l'arbre de décomposition canonique. Giakoumakis et Vanherpe [Van99, GV03] proposent un algorithme en $O(n + m)$ de décomposition des bisplit étendus. Celui proposé ici tourne en $O(n^4)$, mais vaut pour le cas général.

4.9.1 Description de l'algorithme

On note $PPB(u, v)$ le plus petit bimodule contenant deux sommets u et v. Nous allons montrer que cette fonction peut être utilisée pour trouver les bimodules canoniques maximaux dans le cas d'un graphe G c-indécomposable, puis qu'elle peut être calculée en temps $O(n+m)$ par l'algorithme 1 page 119. Cela conduit à un algorithme de décomposition en $O(n^4)$ (théorème 28).

Tout d'abord, notons que la notion de « plus petit bimodule contenant X » est bien définie. Si deux bimodules $M = M_n \cup M_b$ et $M' = M'_n \cup M'_b$ contiennent X et ne sont pas inclus l'un dans l'autre, alors ils se chevauchent et sont un bimodule $(M_n \cap M'_n) \cup (M_b \cap M'_b)$. Il y a donc un plus petit pour l'inclusion contenant X.

Proposition 53
Soit G un graphe biparti sans jumeaux.

1. *si G est* `premier` *alors pour tout sommet u il existe un sommet v tel que $PPB(u, v) = V$*
2. *si G est* `série` *ou* `parallèle` *alors pour tous sommets u et v situés dans des composantes différentes, respectivement C_u et C_v, $PPB(u, v) = C_u \cup C_v$. Et pour tous sommets u et v situés la même composante C, $PPB(u, v) \subset C$.*
3. *si G est* `K+S`*, en notant $C_1 \dots C_k$ les composantes K+S ordonnées de G, alors pour tous sommets u et v situés dans des composantes différentes non-singleton, respectivement C_i*

et C_j, $i < j$, $PPB(u,v) = C_i \cup C_{i+1} \cup \ldots \cup C_{j-1} \cup C_j$. Et s'ils sont situés la même composante C, $PPB(u,v) \subset C$.

Démonstration: Prouvons-le point par point.

1. Soit M un bimodule non-trivial maximal de G contenant u. Tout $v \notin M$ possède la propriété recherchée
2. D'après le lemme 29, si G est une composition parallèle, aucun bimodule ne chevauche de composante connexe de G. Donc $PPB(u,v)$ contient au moins C_u et C_v ; or c'est un bimodule. Le lemme 30 donne le cas série.
3. Voyons le cas où G est K+S. Supposons u et v tous deux noirs. S'ils sont dans la même composante, la proposition est vraie. Sinon, les sommets blancs de toute composante entre C_u et C_v distinguent $\{u,v\}$. D'après le lemme 31, si M est une K+S, un bimodule qui chevauche des composantes de $G[M]$ n'en chevauche que deux, consécutives dans cet ordre. Donc toutes les composantes entre C_u et C_v se suivent. Toujours d'après le lemme 31, si C_u et C_v se suivent mais ne sont pas des singletons, auncun bimodule ne les chevauche donc $PPB(u,v) = C_u \cup C_v$.

□

Cette proposition va nous servir à trouver les bimodules canoniques maximaux d'un graphe c-indécomposable.

Proposition 54
Soit G un graphe c-indécomposable et, pour chaque $u \in V$, M_u un bimodule de type $PPB(u,v)$, différent de V, et de taille maximale parmi ceux ayant cette propriété.

Soit $\mathcal{F} = \{M_u\}_{u \in V}$ la famille de tels bimodules, R la relation sur $\mathcal{F}$ telle que $M_u R M_v$ ssi $M_u \cup M_v$ est un bimodule, et $\tilde{R}$ la fermeture transitive de R. Soit $\mathcal{C} = M_1, \ldots M_k$ une classe d'équivalence de $\tilde{R}$.

$\bigcup_{M \in \mathcal{C}} M$ contient un et un seul bimodule canonique maximal non-trivial de G.

Démonstration: On considére un bimodule canonique maximal M de G. Si M est trivial, M n'appartient à aucun bimodule de type $PPB(u,v)$ différent de V.

Si M est de type `premier`, d'après la proposition 53, pour tout $u \in M$, le plus gros bimodule de la forme $PPB(u,v)$ contient M (il peut avoir un ou deux sommets en plus). Pour tout autre bimodule M' de $\mathcal{F}$, soit $M' \subset M$ soit $M \cup M'$ n'est pas un bimodule.

Si M est de type `K+S`, supposons sans perte de généralité que sa première composante K+S contienne un sommet blanc u. Si sa dernière en contient aussi un, v, alors $M = PPB(u,v)$ d'après la proposition 53. Sinon, l'avant-dernière composante contient un sommet blanc v. La dernière contient un sommet noir v' et la première ou la deuxième un sommet noir v', et on a $M = PPB(u,v) \cup PPB(u',v')$. Pour tous les v de la dernière composante (resp. v de l'avant-dernière composante et v' de la dernière composante), $PPB(u,v)$ (resp. $PPB(u',v')$) est le même bimodule, et tout bimodule plus grand est égal à V. Donc $PPB(u,v)$ (resp. $PPB(u,v)$ et $PPB(u',v')$) appartiennent bien à $\mathcal{F}$.

Enfin si M de type est `série` ou `parallèle` alors d'après la proposition 53, pour toute composante C de M et pour tout $u \in C$, il existe un sommet v situé dans une composante différente de C (peut importe laquelle), tel que $PPB(u, v)$ contienne C. De plus $PPB(u, v)$ est plus grand que $PPB(u, v')$ pour tout $v' \in C$. Donc M_u contient C et une autre composante. L'union de toutes ces composantes est bien le bimodule M.

□

Pour obtenir les bimodules canoniques maximaux, il suffit de prendre l'intersection de tous les membres de $\mathcal{F}$, ce qui supprime les conflits d'union et donne les canoniques maximaux non-triviaux, et de rajouter les singletons. Nous allons voir maintenant comment cet algorithme peut être implémenté en $O(n^3)$, d'où un algorithme récursif de décomposition en $O(n^4)$.

4.9.2 Calcul du plus petit bimodule

Algorithme 1: Plus petit bimodule contenant deux sommets donnés

Données : Deux sommets x et y de la même couleur
Résultat : Le plus petit bimodule M tel que $\{x, y\} \subset M$
Soit F une file (FIFO)
début
  $M \leftarrow \{x\}$
  Enfiler y dans F
  tant que F *n'est pas vide* **faire**
    Extraire u de F
    Soit v le dernier sommet de même couleur que u ajouté à M
    Enfiler dans F tout sommet qui distingue u de v
    Ajouter u à M
  fin
  retourner M
fin

Proposition 55
L'algorithme 1 calcule $PPB(u, v)$ en temps $O(m + n)$.

Démonstration: L'algorithme 1 maintient l'invariant suivant :

Invariant 1
*À chaque entrée dans la boucle « **tant que** », tout sommet noir (resp. blanc) qui distingue deux sommets blancs (resp. noirs) de M est dans F.*

Démontrons-le par récurrence. Il est initialement vrai pour M singleton. Considérons les valeurs de M et F lors d'une entrée dans la boucle, et soit u le sommet extrait de F. Supposons sans perte de généralité que u soit blanc. Tout sommet distinguant M appartient à F par hypothèse d'induction. Un sommet distinguant $M \cup \{u\}$ doit donc être un sommet noir w, distinguant u d'un autre sommet $u' \in M$. Mais comme w ne distingue aucun sommet blanc de M, w distingue $\{u, v\}$ pour *tout* v blanc de M, en particulier le dernier sommet blanc inséré dans M. w est donc mis dans F et quand M devient $M \cup \{u\}$ et l'invariant est vrai.

Il est clair que M est un bimodule ssi la file est vide, ce qui est la condition d'arrêt de l'algorithme. S'il existait un plus petit bimodule que M contenant x et y, un sommet $u \in M$ distinguerait ce bimodule, vu la façon dont M est construit, contradiction. Le résultat obtenu est correct.

Calculer les séparateurs de deux sommets u et v, c'est-à-dire $\Gamma(u)\Delta\Gamma(v)$, peut se faire en $O(|\Gamma(u)| + |\Gamma(v)|)$, voir §5.4.2. La complexité vient du fait qu'un sommet est inséré au plus une fois dans F et que l'adjacence d'un sommet est examinée au plus deux fois, pour le calcul des séparateurs.

□

4.9.3 Implémentation

Notre algorithme est en deux temps. Une première phase calcule $PPB(u, v)$ pour tous les couples $\{u, v\}$ de sommets de la même couleur. Pour chaque sommet u, un tableau PPB_u stocke les $O(n)$ bimodules $PPB(u, v)$, triés par tailles décroissantes. Calculer PPB_u prend $O(nm)$ avec $O(n)$ appels à l'algorithme 1, et le trier prend $O(n)$ car la taille est bornée par n : un tri linéaire peut être utilisé. Un bimodule est stocké comme un vecteur de taille n ainsi qu'un entier donnant sa taille. Cette première phase prend donc un temps $O(n^2m)$ et donne un résulat de taille $O(n^3)$.

La deuxième phase est récursive. On part de $M = V$. On calcule les composantes connexes de $G[M]$ et $\overline{G[M]}^{bip}$ par simple parcours en largeur, en $O(|M| + m)$, et les composantes K+S grâce à l'algorithme de [Van99], en $O(|M|)$. Si $G[M]$ est c-décomposable, cela donne les bimodules forts canoniques sur lesquels repartir récursivement.

Et si $G[M]$ est c-indécomposable, pour tout $u \in M$, on extrait de PPB_u le (ou *un des*) plus grand bimodule de taille strictement inférieure à $|M|$ inclus dans M. Tester si un des $O(n^2)$ bimodules stockés est strictement inclus dans M se fait en $O(n)$: cette phase prend $O(n^3)$.

On obtient ainsi une famille de $O(n)$ bimodules $\mathcal{F} = \{M_u \mid u \in M\}$. D'après la proposition 54, pour trouver les bimodules non-triviaux maximaux de $G[M]$ on n'a qu'à appliquer le processus suivant : pour tout $u \neq v$, si $M_u \cup M_v$ est un bimodule, on remplace M_u et M_v par $M_u \cup M_v$ dans $\mathcal{F}$. Tester si l'union de deux bimodules est un bimodule se fait en prenant un sommet blanc $b \in M_u$ et un blanc $b' \in M_v$ et en testant si $Sep(\{u, v\}) \in (M_u \cup M_v)$, et de même en prenant deux noirs : le test peut se faire en $O(n)$. On réalise $O(n^2)$ tests : cette phase prend un temps $O(n^3)$.

Maintenant $\mathcal{F}$ est la famille des bimodules maximaux de G. On calcule l'intersection de tous les membres de $\mathcal{F}$. Cela peut se faire en $O(n^2)$ en utilisant un vecteur représentant M : on marque chaque sommet du bimodule qui le contient ; et si un sommet est marqué plusieurs fois il est conflictuel et est enlevé de tous les bimodules le contenant. On obtient ainsi les bimodules canoniques maximaux de $G[M]$.

Cet algorithme récursif est appelé $O(n)$ fois et tourne en $O(n^3)$. Donc

Théorème 28
L'arbre de décomposition canonique d'un graphe peut être calculé en temps $O(n^4)$.

4.10 Compléments sur la décomposition bimodulaire

4.10.1 Bicomplémentation

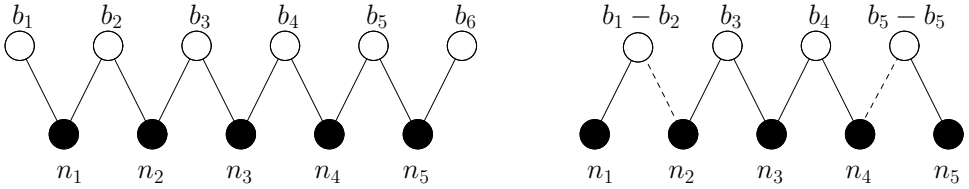
On a vu que M est un bimodule de G ssi c'est un bimodule de $\overline{G}^{bip}$. De plus un bimodule est maigre (resp. K+S, trivial) dans G ssi il est maigre (resp. K+S, trivial) dans $\overline{G}^{bip}$. La preuve est immédiate. On en déduit que les bimodules canoniques de G sont les mêmes que ceux de $\overline{G}^{bip}$, puisque la définition des bimodules canoniques forts et canoniques faibles fait seulement appel à ces notions. G et $\overline{G}^{bip}$ ont donc le même arbre, et les nœuds `premier` (resp. `linéaire`, `complet`) restent `premier` (resp. `linéaire`, `complet`). Cependant, on montre facilement que les nœuds `série` deviennent `parallèle` et inversement[11], et que l'ordre des nœuds `K+S` est inversé. La décomposition bimodulaire des graphes bipartis se comporte donc exactement comme la décomposition modulaire des graphes, vis-à-vis de l'opération de complément adaptée.

4.10.2 Graphe quotient

La puissance de la décomposition modulaire vient aussi de l'existence d'un *graphe quotient*, construit en prenant un sommet par module fort maximal. Dans le cas des graphes bipartis, afin d'obtenir un quotient qui soit également biparti, il faut prendre *deux* sommets par bimodule canonique fort maximal non-trivial : un pour les noirs et un pour les blancs. Selon les adjacences entre ces bimodules on construit facilement les adjacences entre sommets représentatifs différents. Par contre se pose la question de savoir si les deux sommets représentant un bimodule maximal sont adjacents ou non. Si on choisit l'un ou l'autre, le graphe quotient risque de ne pas être premier.

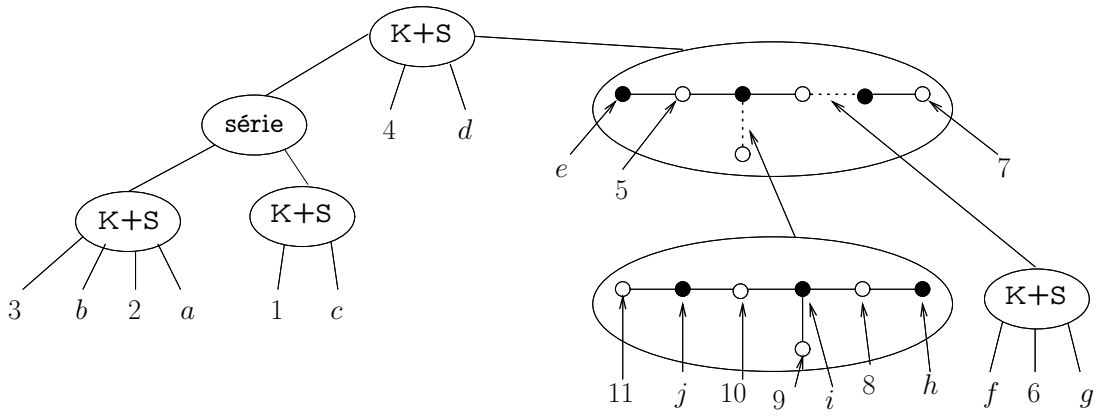
Un moyen de palier à cet inconvéniant est de placer une **arête spéciale** entre les deux sommets représentant le même bimodule. Deux arêtes spéciales ne peuvent être adjacentes. Le quotient n'est donc plus un vrai biparti.

11. c'est pour que cette jolie propriété soit vraie que $2K_2$, le seul cas pathologique, est défini c-indécomposable au lieu d'être `série` ou `parallèle`.



Un problème de quotient. Le graphe de gauche a comme bimodules non-triviaux $\{b_1, b_2, n_2\}$ et $\{b_5, b_6, n_4\}$. Celui de droite est son quotient. Si les arêtes pointillées comptent comme adjacence alors il n'est pas premier : $\{b_3, n_1, n_2\}$ est un bimodule. Et si elles comptent comme non-adjacence alors il n'est même plus 1-premier et $\{n_2, n_3, b_4\}$ est un bimodule. Donc ces arêtes doivent être spéciales.

Par contre l'arbre de décomposition bimodulaire, où chaque nœud premier est étiqueté par son quotient, est un codage compact et non-ambigu du graphe de départ. Chaque bimodule fils d'un premier est relié par une flèche à l'arête spéciale qui lui correspond, comme dans l'exemple suivant :



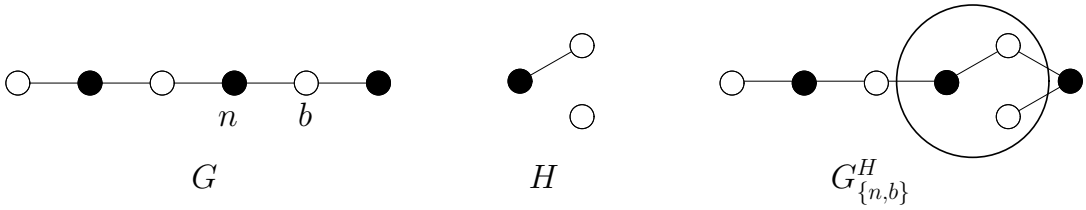
Arbre de décomposition bimodulaire du graphe de la section 4.6 page 107. Les quotients on été dessinés dans les nœuds `premiers`*. Une telle représentation est un dessin très lisible et compact du graphe.*

4.10.3 Une opération de substitution

Étant donnés deux graphes bipartis $G = (N_G, B_G, E_G)$ et $H = (N_H, B_H, E_H)$ et deux sommets $n \in N_G$ et $b \in B_G$, la **substitution** de $\{b, n\}$ par H au sein de G produit le graphe biparti

$$G^H_{\{n,b\}} = ((N_G \setminus \{n\}) \cup N_H, (B_G \setminus \{b\}) \cup B_H, E_G \cup E_H \cup$$

$$\{(u, v) | u \in (N_G \cup B_G), v \in (N_H \cup B_H) \text{ et } (u, b) \in E_G \text{ ou } (u, n) \in E_G\})$$



Manifestement H est un bimodule de $G^H_{\{n,b\}}$ et si G et H sont sans jumeaux $G^H_{\{n,b\}}$ l'est aussi. En revanche, l'opération de **compactage** qui serait l'inverse de cette substitution n'est pas définissable : on a en effet perdu l'information de l'adjacence entre b et n, et dans le doute on doit mettre une arête spéciale.

Notons que substituer un bicographe par un bicographe peut produire un graphe qui n'est pas un bicographe (ni bisplit étendu), car dans l'exemple un P_7 est produit.

4.10.4 Décomposition de matrices booléennes

Un graphe biparti $G = (N \uplus B, E)$ peut être vu comme une matrice booléenne. En numérotant de 1 à n les éléments de N et de 1 à m ceux de B, on construit une matrice M de dimension $n \times m$ telle que $m_{i,j} = 1$ si $(n_i, b_j) \in E$. Un **bloc** de M est une sous-matrice constituée de lignes adjacentes et de colonnes adjacentes.

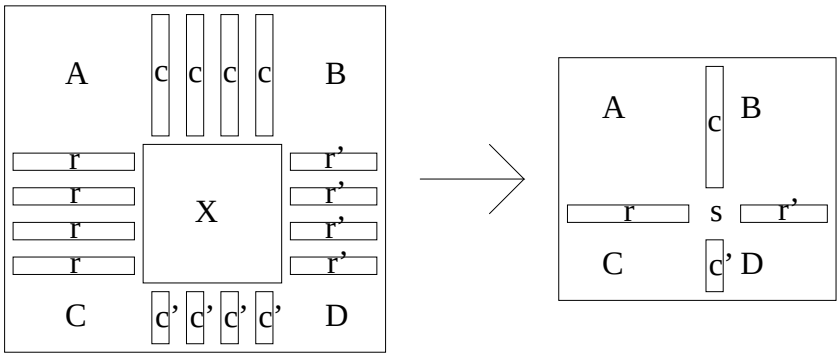
Une matrice booléenne M correspondant à un graphe biparti G est sous **forme normale bimodulaire** si tout bimodule canonique fort de G est un bloc de M. *Mettre sous forme normale bimodulaire* une matrice booléenne consiste à permuter ses lignes et colonnes jusqu'à obtenir une matrice sous forme normale bimodulaire.

Proposition 56
Toute matrice peut être mise sous forme normale bimodulaire.

Démonstration: On construit l'arbre de décomposition canonique du graphe biparti associé à M et on le dessine dans le plan. Cela fournit un ordre des sommets blancs, correspondant à une permutation des lignes, et un ordre des sommets noirs, correspondant à une permutation des colonnes, qui produit la propriété désirée.

□

Si X est un bloc de M toutes les lignes intersectant M sont identiques quant à leur partie hors de M. Les colonnes possèdent aussi cette propriété. On peut donc compacter ce bloc en un sommet. Cependant se pose toujours la question de savoir si on met un 1 ou un 0 dans la case correspondante.



4.10.5 Bimodules gras et maigres

Un bimodule M est **maigre** s'il ne possède pas de sommet M-homogène. Cela exclut en particulier les singletons, les paires et les triangles : un bimodule maigre possède au moins quatre sommets (les seuls à exactement quatre sommets induisent un $2K_2$). Un bimodule non-maigre est **gras**.

Un bimodule gras M contient un sommet M-homogène, qui peut être un sommet blanc M-universel (un blanc adjacent à tous les noirs de M) ce que l'on abrège en B, blanc M-isolé (non-adjacent aux noirs de M) abrégé en $\overline{B}$, noir M-universel N ou noir M-isolé $\overline{N}$. Comme G est un graphe sans jumeaux, M ne peut avoir plusieurs sommets du même type (par exemple plusieurs noirs M-universels) sans quoi ces sommets seraient jumeaux. Qui plus est, N et $\overline{B}$ ne peuvent arriver simultanément, de même que B et $\overline{N}$.

Si l'on *enlève* à un bimodule gras M un sommet v M-homogène, on obtient un bimodule $M - v$. Supposons que v soit blanc isolé. On note

$$M = \overline{B}.(M - v)$$

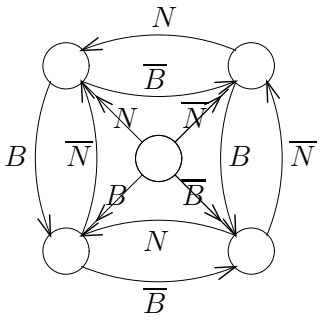
$M - v$ peut être un bimodule maigre. Il n'a pas de blanc isolé (qui serait un jumeaux de v). M_v possède un noir isolé ssi M possède un noir isolé, et un blanc isolé ssi M possède un blanc isolé. En revanche il peut apparaître dans $M - v$ un noir universel. Une **séquence d'émondage** consiste à itérativement enlever des sommets à un bimodule gras, jusqu'à obtenir un bimodule maigre. On note une séquence d'émondage

$$M = X_1 \dots X_k M_0$$

M est un bimodule gras, M_0 est maigre et $X_i \in \{N, \overline{N}, B, \overline{B}\}$. On observe qu'un noir M-isolé peut apparaître seulement si l'on enlève un sommet blanc M-universel. Réciproquement, il faut enlever un blanc M-universel pour faire apparaître un noir M-isolé. Comme par ailleurs noirs M-isolés et blancs M-universels sont incompatibles dans le même bimodule on a :

Proposition 57
B et $\overline{N}$ sont alternés dans toute séquence d'émondage : entre deux B il y a un $\overline{N}$, entre deux N il y a un B. De même N et $\overline{B}$ sont alternés dans toute séquence d'émondage.

La deuxième partie de la proposition se prouve comme la première. C'est la seule contrainte : on commence et finit par n'importe laquelle des quatre lettres, entre deux B et $\overline{N}$ minimalement proches on peut trouver une suite alternée de N et $\overline{B}$ de longueur quelconque, et même chose entre deux N et $\overline{B}$ minimalement proches. On en déduit que la grammaire qui génère toutes les séquences d'élagage est $L.M_0$, où L est reconnu par l'automate suivant dont chaque est final et dont l'état initial est au centre :



Proposition 58
G est un graphe de degrés inégaux si et seulement si une séquence d'élagage des sommets de G mène au graphe vide.

Démonstration: un graphe de degrés inégaux possède forcément un sommet isolé ou universel, ce qui démontre le « seulement si ». Inversement enlever un sommet isolé ou universel d'un graphe où tous les sommets ont degré différent préserve cette propriété, ce qui démontre le « si ».

□

On en déduit :

Proposition 59
M est un bimodule gras si et seulement si on peut écrire $M = Z \uplus M_0$ où M_0 est un bimodule maigre (éventuellement vide) et Z un bimodule induisant un biparti de degrés inégaux. Z et M_0 sont uniques.

Z est la *partie grasse* de M et M_0 sa *partie maigre*.

4.11 Conclusion

Le principal résultat de ce chapitre est qu'une structure de donnée de taille $O(n)$, l'arbre de décomposition canonique muni de ses marques, permet de retrouver tous les bimodules de plus de trois sommets d'un graphe biparti !

Tout ce qui est dit dans cette section s'adapte trivialement au cas des graphes *cobipartis*, qui sont tout simplement le complémentaire d'un biparti. Cela pourrait aussi s'adapter au cas des

split graphs[12], qui sont des graphes où V est partitionné entre une clique K et un stable S. Le théorème de caractérisation est presque le même. On peut définir des *splimodules* semblables aux 2-modules. La construction de la famille canonique serait serait sans doute semblable...

On pourrait améliorer l'algorithme de décomposition en recherchant non pas les $O(n^2)$ plus *petits* bimodules contenant deux sommets, mais, pour un v donné, les plus *grands* bimodules ne contenant pas v. C'est ainsi que [EGMS94, DGM97, DGM01] construisent des algorithmes efficaces de décomposition modulaire.

12. graphes cassés selon la francisation de [Lan01], qui prouve entre autres qu'un graphe cassé 1-premier possède aux plus deux telles bipartitions, différentes d'un sommet.

Deuxième partie

Algorithmes de décomposition modulaire

Chapitre 5

Permutations factorisantes. Outils algorithmiques

Notre lieutenant général, Léonard Euler, déclare par notre bouche ce qui suit : «[...] Que pour rentrer en grâce auprès des géomètres, il tâchera de mettre à l'avenir un peu d'élégance dans l'analyse qu'il leur offrira ; qu'il n'emploiera plus soixante pages de calcul pour arriver à une conclusion qu'on peut établir par un raisonnement de dix lignes ; item, que toutes les fois qu'il retroussera ses bras pour calculer trois jours et trois nuits de suite, il se donnera la patience de raisonner auparavant un quart d'heure sur le choix des principes qu'il conviendra d'employer.»

VOLTAIRE*, Diatribe du docteur Akakia*

L'ALGORITHMIQUE DE DÉCOMPOSITION MODULAIRE est une base pour de nombreux problèmes de théorie des graphes. De nombreux algorithmes de reconnaissance de classes de graphes ont besoin de la décomposition modulaire, parce que les nœuds de l'arbre de décomposition ont une propriété particulière. Citons en vrac : les cographes n'ont pas de nœud `premier` [CLS81], les graphes P_4-sparse, P_4-reducible et P_4-tidy dont les nœuds `premier` ont des propriétés particulières [Hoà85, JO89, GRT97], les graphes de comparabilité premiers n'ont qu'une seule classe de forçage ce qui permet leur reconnaissance [Gal67], et ce qui s'adapte à des sous-classe comme les graphes de permutation : dans tous les algorithmes cités, si la reconnaissance peut se faire en temps linéaire, c'est parce que la décomposition modulaire peut se faire en temps linéaire. §2.2.5 donne d'autres exemples.

Ce chapitre qui prélude aux algorithmes de décomposition modulaire, qui seront exposés dans les trois chapitres suivant, met en place tous les outils et notions nécessaires. §5.1 commence par brosser un tableau de l'histoire de la décomposition modulaire, afin de situer cette thèse dans un

son contexte. Puis est présenté l'outil nouveau, qui est l'idée de base de tous ces algorithmes : les *permutations factorisantes*. §5.2 établit le lien entre permutations factorisantes et familles partitives et bipartitives (*via* les PQ-trees et PC-trees) et présente le schéma d'algorithme *deux-temps* que nous utilisons : calcul d'une permutation factorisante, puis calcul de l'arbre de décomposition en se basant sur cette permutation. Est alors présentée une méthode générique pour obtenir une permutation factorisante, l'*affinage d'ordres factorisants*.

Sont ensuite exposés quatre outils algorithmiques qui serviront à plusieurs reprises à nos algorithmes. Il est également présenté §5.5 un algorithme calculant l'intersection de familles partitives, fruit d'un travail réalisé avec Ross McConnell. Cet algorithme, intéressant en lui-même, est un outil de l'algorithme de décomposition modulaire des graphes orientés présenté chapitre 7. Enfin un théorème établit que la *taille* de la famille des modules forts d'un graphe, $\sum_{M \text{ est un module fort}} |M|$, est linéaire en la taille du graphe (et non quadratique comme on pourrait le penser) ce qui servira à des preuves de complexité. Les algorithmes proprement dit pourront alors être proposés.

5.1 Histoire de l'algorithmique de décomposition modulaire

Le tableau page suivante décrit les principaux algorithmes de décomposition modulaire publiés (ou non) à ce jour, à ma connaissance. Sauf précision contraire, les algorithmes sont séquentiels et concernent les graphes non-orientés. On peut observer une course à la généralité et à l'efficacité, typique des grands problèmes algorithmiques. Ce que le tableau ne montre pas est la course à la *simplicité*. En effet, les auteurs depuis 1994 (date des premiers algorithmes linéaires) cherchent à écrire des algorithmes plus simple à comprendre, à prouver et à programmer. Ainsi la version linéaire de [DGM01] est présentée comme une sophistication (clairement sous-entendue purement théorique) d'un algorithme en $O(n + m\alpha(m, n))$ [1]. [MS00] prend carrément le parti de présenter un algorithme surlinéaire, mais simple. C'est dans cette optique de simplification qu'il faut comprendre cette thèse.

Pour écrire des algorithmes séquentiels linéaires, quatre écoles principales émergent. [MS99] part d'un P_4-*tree* [Spi92] et parvient à l'arbre final par diverses manipulations d'arbre. [CH94] se base sur les propriétés d'inclusion des graphes premiers. [EGMS94, DGM97, DGM01], pour un sommet v donné, calculent les modules maximaux de $G - v$. Cela fournit la branche gauche de l'arbre de décomposition modulaire, allant de la racine à v : il ne reste plus qu'à attaquer récursivement ces modules et à recoller les morceaux. Enfin, [CH97, HPV99] et le présent ouvrage utilisent une stratégie *deux-temps* passant par la production d'une permutation factorisante (qui sera exposée en détails).

1. $\alpha(m, n)$ est l'inverse de la fonction d'Ackermann. Seuls les combinatoristes sont assez « audacieux » pour produire des valeurs de m et n pour lesquelles cette fonction dépasse 5, tellement ces nombres défient le sens commun et la réalité matérielle : si l'on prend l'âge de l'univers exprimé en picosecondes, mis à la puissance du nombre d'atomes de l'univers observable, on est encore bien en-deça...

Réf.	publié en	complexité	notes
[CJS72]	1972	$O(n^4)$	
[Mau77]	1977	$O(n^4)$	graphes orientés
[Bla78]	1978	$O(n^3)$	
[HM79]	1979	$O(n^3)$	
[Hab81]	1981	$O(n^3)$	graphes orientés
[Cun82]	1982	$O(n^3)$	graphes orientés
[BM83]	1983	$O(n^3)$	
[McC87]	1987	$O(n^3)$	
[MS89]	1989	$O(n^2)$	incrémental
[AP90]	1990	$O(\log^2 n)$, $O(nm)$ proc.	//, cographes, CRCW-PRAM
[LO91]	1991	$O(\log n)$, $O(nm)$ proc.	//, cographes, EREW-PRAM
[EHR94]		$O(n^3)$	2-structures, incrémental
[Spi92]	1992	$O(n + m\alpha(m, n))$	
[Cou93, CH93]	1993	$O((n + m) \log n)$	
[EGMS94]	1994	$O(n^2)$	2-structures
[MS94b, MS99]	1994	$O(n + m)$	
[CH94]	1994	$O(n + m)$	
[BDV95, BDV99]	1995	$O(n^{3k-5})$	dec. comités des k-hypergr.
[Dah95a]	1995	$O(\log^2 n)$, $O(n + m)$ proc.	//, CRCW-PRAM
[Dah95b]	1995	$O(\log^2 n)$, $O(n + m)$ proc.	//, cographes, CRCW-PRAM
[HHS95]	1995	$O(n + m)$	graphes d'héritage
[McC95]	1995	$O(n^2)$	2-structure, incrémental
[MV95]	1995		parallèle
[CH97, Cap97b]	1997	$O(n + m)$	si permu. facto. fournie
[DGM97, DGM01]	1997	$O(n + m)$	
[DGM99, DGM02]	1999	$O(n + m \log n)$	graphes orientés
[HPV99]	1999	$O(m \log n)$	permu. facto. seule
[MS00]	2000	$O(n + m \log n)$	
[HP01]	2001	$O(n + m)$	cographes
[CHM02], chap. 8	2002	$O(n + m)$	gr. orientés, p. facto. fournie
[SS03]	2003	$O(d)$/sommet, $O(1)$/arête	cographes, fully-dynamic
[BCHP03]	2003	$O(n + m)$	cographes
chap. 6	2003	$O(n + m)$	permu. facto. seule
[MM03], chap. 7	2003	$O(n + m)$	graphes orientés

Historique des algorithmes de décomposition modulaire

Les outils ont évolués. Les méthodes actuelles sont à base d'*affinage de partition*, en utilisant le voisinage d'un sommet comme pivot (*vertex-splitting*). L'idée de base est que le voisinage d'un sommet ne peut chevaucher de module. Pour un ensemble $C \subset V$ (en particulier pour une classe d'une partition de V) et un sommet $x \notin C$, couper C en $C \cap \overline{\Gamma}(x)$ et $C \cap \Gamma(x)$ ne

coupe aucun module auquel x n'appartient pas. Un module peut néanmoins intersecter plusieurs classes : l'ordre des classes intervient, comme expliqué en §6.1.

Cette stratégie, ou sa généralisation aux *partitions ordonnées de chaînes* (voir chap. 6), est peu adaptée aux algorithmes dynamiques voire « **fully-dynamic** ». Dans ce problème, on considère un graphe pouvant évoluer (par ajout ou retrait de sommets ou arêtes) et l'on souhaite pouvoir disposer de sa décomposition modulaire à chaque instant. Il faut disposer d'une bonne structure de données, afin de ne pas tout recalculer à chaque fois !

Question ouverte 2
Écrire un algorithme complètement dynamique de décomposition modulaire, de complexité amortie meilleure que $O(n+m)$ par opération.

À l'heure actuelle on ne connaît que de bons algorithmes incrémentaux [2] en $O(n^2)$.

5.2 Permutations factorisantes

Les permutations sont un concept fondamental des mathématiques comme de l'informatique. Cette deuxième partie de thèse tourne autour d'une classe particulière de permutations d'un ensemble V, compatibles avec une famille partitive de V : les permutations factorisantes. En se restreignant aux cas de la décomposition modulaire, sont présentés des algorithmes de calcul de permutations factorisantes, pour les graphes non-orientés, les tournois et les graphes orientés, de complexité $O(n+m)$. Leur intérêt est que, couplés à l'algorithme du chapitre 8, ils fournissent un algorithme linéaire et simple de décomposition modulaire. Ils utilisent des techniques d'*extension d'ordres factorisants*, qui permettent de passer d'un ordre vide à un ordre total (permutation) en maintenant une propriété de compatibilité avec la décomposition modulaire du graphe.

Soit E un ensemble fini à n éléments. Les trois définitions suivantes définissent le même objet mathématique :

- une bijection de E dans $[\![1, \dots n]\!]$,
- un ordre total sur E,
- un mot de E^* où chaque lettre apparaît exactement une fois.

Cet objet sera appelé une **permutation** de E. Elle est usuellement notée σ.

Le vocabulaire et les notations relatifs aux permutations utilisent le champ lexical de la définition la plus pratique. $\sigma^{-1} : [\![1, \dots n]\!] \mapsto E$ est la permutation inverse. On note $\mathfrak{S}_E$ l'ensemble des permutations de E et $\mathfrak{S}_n$ l'ensemble des permutations de $[\![1, \dots n]\!]$. Le $i^{\text{ème}}$ élément d'une permutation est noté $\sigma^{-1}(i)$, et dans ce cas i est l'**indice** de $\sigma^{-1}(i)$.

Soient $e = \sigma^{-1}(i)$ et $f = \sigma^{-1}(j)$. Si $i \leqslant j$ (resp. $i < j$, $i \geqslant j$, $i > j$), on note $e \preccurlyeq_\sigma f$ (resp. $e \prec_\sigma f$, $e \succcurlyeq_\sigma f$, $e \succ_\sigma f$) ou, s'il n'y a pas d'ambiguïté, $e \preccurlyeq f$ (resp. $e \prec f$, $e \succcurlyeq f$, $e \succ f$). Si

2. La seule opération autorisée est l'ajout d'un sommet et de tout son voisinage.

$e \prec f$ on dit que e est **à gauche** de f ; sinon il est **à droite**[3].

Un **facteur** est un sous-mot de σ, noté $[e \dots f]$. On peut aussi le voir comme l'image inverse de $[\![a, \dots b]\!]$, $0 \leqslant a \leqslant b \leqslant n$: il contient *tous* les éléments entre deux indices donnés. Un facteur sera noté avec des crochets simples $[e \dots f]$ quand ses bornes sont données dans E, et doubles $[\![3, \dots 8]\!]$ quand elles sont dans $\mathbb{N}$.

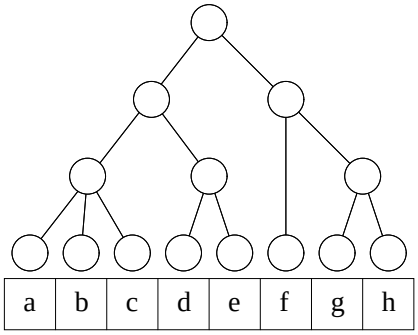
5.2.1 Familles partitives et faiblement partitives

Définition 30
*Soit $\mathcal{P}$ une famille (faiblement) partitive de E et $\mathcal{F}$ l'ensemble des éléments forts de $\mathcal{P}$. Une permutation σ de E est **factorisante** pour $\mathcal{P}$ si pour tout $F \in \mathcal{F}$ F est un* facteur *de σ.*

Si l'on dessine dans le plan l'arbre $\mathcal{T}_{\mathcal{P}}$ des parties fortes de $\mathcal{P}$, la lecture de gauche à droite des feuilles fournit une permutation factorisante. On peut donc affirmer

Proposition 60
Toute famille (faiblement) partitive admet une permutation factorisante.

Mais, réciproquement, à une permutation factorisante donnée correspond un unique dessin de $\mathcal{T}_{\mathcal{P}}$: celui où les fils d'un nœud donné arrivent dans l'ordre des facteurs correspondants de σ.



Dessin d'un arbre d'inclusion au-dessus d'une permutation factorisante.

Tout dessin de $\mathcal{T}_{\mathcal{P}}$ ne fournit pas une permutation factorisante : si les fils d'un nœud *complet* ou *premier* peuvent être dessinés dans un ordre quelconque, ceux d'un nœud *linéaire* doivent suivre l'ordre induit par ce nœud.

Proposition 61
L'ensemble des permutations factorisantes d'une famille faiblement partitive $\mathcal{P}$ est en bijection avec l'ensemble des dessins dans le plan de $\mathcal{T}_{\mathcal{P}}$ où tous les fils d'un nœud linéaire *suivent l'ordre induit par ce nœud.*

3. ces notations viennent de σ vue comme un mot, et peuvent être source de confusion pour des lecteurs de langues orientales, arabe ou hébraïque. Mais dire « plus petit que » entretient une confusion quand les éléments sont canoniquement ordonnés par autre chose que σ, par exemple les entiers selon $\leqslant$.

Cette proposition permet de savoir combien de permutations factorisantes possède une famille. On remarque que les nœuds *premiers* ou *complets* à k fils peuvent être dessinés de $k!$ façons différentes (toute permutation des fils est valide) tandis que ceux d'un nœud *linéaire* de deux façons seulement (de gauche à droite ou de droite à gauche, suivant l'ordre). On dispose donc d'une relation de récurrence, le long de l'arbre d'inclusion, permettant de compter les permutations factorisantes d'une famille : pour un nœud N complet ou premier à k fils on a

$$P(N) = k! \prod_{F \ fils \ de \ N} P(F) \tag{5.1}$$

et pour un nœud linéaire :

$$P(N) = 2 \prod_{F \ fils \ de \ N} P(F) \tag{5.2}$$

Les familles faiblement partitives peuvent donc n'avoir que deux permutations factorisantes. En revanche, pour une famille partitive, il y en a beaucoup.

Proposition 62
Une famille partitive sur n éléments admet au moins 2^{n-1} permutations factorisantes.

Démonstration: On vérifie pour $n = 1$. Supposons la propriété vraie pour les familles à n éléments, et montrons-la pour une famille à $n+1$ éléments. Soit $\mathcal{F}_{n+1}$ une famille partitive d'un ensemble E à $n+1$ éléments. On prend $e \in E$. La famille $\mathcal{F}_n = \{F \setminus \{e\} \mid F \in \mathcal{F}_{n+1}\}$ est partitive sur n éléments et admet donc par hypothèse 2^{n-1} permutations factorisantes. Soit A le plus petit élément fort de $\mathcal{F}_{n+1}$ à plus que un élément contenant e. On considère les fils $A_1 \dots A_k$ de $A \setminus \{e\}$ dans $\mathcal{T}_{\mathcal{F}_n}$. Par hypothèse $\mathcal{T}_{\mathcal{F}_n}$ possède au moins 2^{n-1} dessins différents. Il existe $k+1$ façons de placer $\{e\}$ comme fils de A dans un dessin de $\mathcal{T}_{\mathcal{F}_n}$ pour obtenir un dessin de $\mathcal{T}_{\mathcal{F}_{n+1}}$; les permutations factorisantes correspondantes sont toutes différentes. Or $k \geqslant 1$. $\mathcal{T}_{\mathcal{F}_n}$ a donc au moins 2^n dessins différents.

□

Dans le cas de la décomposition modulaire on remarquera que, comme ce sont exactement les arbres binaires qui atteignent cette borne, et que comme tout nœud `premier` est d'arité au moins 3, seuls les cographes peuvent atteindre la borne.

5.2.2 PQ-trees

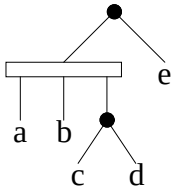
Un autre point de vue sur les familles partitives est celui des PQ-trees [BL76] où l'on définit l'arbre au lieu de la famille. Il est est utilisé pour certaines propriétés des matrices (les 1 consécutifs : on recherche une permutation des colonnes d'une matrice booléenne telle que sur chaque ligne tous les 1 soient consécutifs) et des graphes d'intervalles (il permet de générer tous les modèles d'intervalles, voir §8.8)

Les PQ-trees sont des objets permettant de générer toutes les permutations factorisantes d'une décomposition donnée : les fils d'un nœud P peuvent être arbitrairement ordonnés, tandis que

les fils d'un nœud Q doivent être ordonnés dans un certain ordre, exactement comme dans la proposition 61.

Un PQ-tree sur un ensemble V est un arbre enraciné dont les feuilles sont les éléments de V et dont les nœuds sont étiquetés P ou Q. Les fils d'un nœud Q sont totalement ordonnés. Une permutation σ de V est **compatible** avec le PQ-tree T si elle est obtenue en lisant par parcours en profondeur (ie de gauche à droite dans un certain plongement) les feuilles de T de sorte que les fils d'un nœud Q sont visités soit dans l'ordre de ce nœud, soit dans l'ordre inverse.

Un PQ-tree est **canonique** s'il n'a pas de nœud à un fils, pas de nœud Q à deux fils, et si les fils d'un nœud P ne sont pas des nœuds P. On peut facilement rendre canonique tout PQ-tree. La représentation classique des PQ-trees dessine les nœuds P comme des points et les nœuds Q comme des segments :



Exemple de PQ-tree. Les permutations compatibles sont $(abcde)$, $(abdce)$ $(dcbae)$, $(cdbae)$, $(eabcd)$, $(eabdc)$, $(edcba)$ *et* $(ecdba)$.

$M \subset V$ est un **facteur** d'un PQ-tree T si M est un facteur de toute permutation compatible d'un PQ-tree. L'ensemble de tous les facteurs d'un PQ-tree donné est une **famille PQ**.

Théorème 29
Les facteurs d'un PQ-tree sont une famille faiblement partitive.
L'arbre partitif correspondant à cette famille est exactement le PQ-tree canonique : les nœuds P sont des nœuds `premier`*, les nœuds Q des nœuds* `linéaire`*, et il n'y a pas de nœud* `complet`*.*

Démonstration: Tout d'abord on constate que pour un nœud N, l'ensemble de ses feuilles est un facteur. Ensuite on prouve que ces facteurs sont *forts*. Soit N un nœud et $(x_1 \dots x_a)$ le facteur correspondant à N dans une permutation compatible donnée $(v_1 \dots x_1 \dots x_a \dots v_n)$. S'il n'est pas fort alors un autre facteur $(y_1 \dots y_b x_1 \dots x_c)$, $c < a$, ou bien $(x_d \dots x_a z_1 \dots z_d)$, $1 < d$, le chevauche. La permutation obtenue en *retournant* N, $(v_1 \dots x_a \dots x_1 \dots v_n)$, est aussi compatible, ce qui contredit que $(y_1 \dots y_b x_1 \dots x_c)$ ou $(x_d \dots x_a z_1 \dots z_d)$ soient des facteurs.

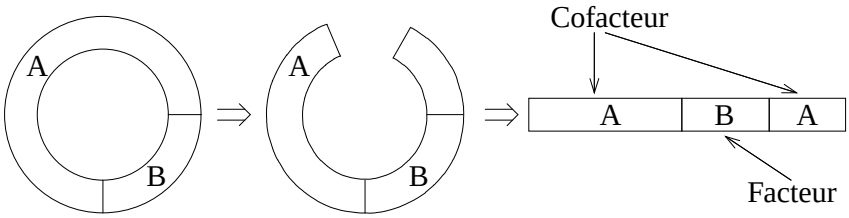
Puis le théorème se démontre par récurrence sur le PQ-tree T. Si la racine de T est un nœud P alors les seuls facteurs sont V ou sont contenus dans les parties correspondant aux fils de ce nœud : la racine de l'arbre partitif est `premier`. Et si c'est un nœud Q alors toute union de fils consécutifs dans cet ordre donne un facteur faible : on obtient un nœud `linéaire` de l'arbre partitif.

□

5.2.3 Familles bipartitives

Le concept de permutation factorisante s'étend aussi aux *familles bipartitives*. Seulement, l'arbre représentant une famille bipartitive n'est pas enraciné. Une lecture de ses feuilles ne donne donc plus une liste, mais seulement une *liste circulaire* : on ne sait lui donner un début ou une fin. Nous aurions pu travailler avec des listes circulaires, mais nous préférons le formalisme des permutations. Il nous faut donc *couper* la liste circulaire à un endroit précis. Mais on n'a pas à se préoccuper du choix de l'endroit. En effet, la notion intéressante dans une permutation factorisante est le *facteur*, notion que l'on retrouve aussi dans les listes circulaires : deux positions dans une liste circulaire définissent deux facteurs. Mais si l'on coupe la liste circulaire pour en faire une permutation, alors l'un des facteurs devient un vrai facteur de la permutation tandis que l'autre devient *deux* facteurs. Nous capturons cela par la notion de *cofacteur* :

Définition 31
*Soit σ une permutation de V et $1 \leqslant i \leqslant j \leqslant |V|$. $\{x \in V \mid i \leqslant \sigma(x) \leqslant j\}$ est le **facteur** $[\![i, j]\!]$ de σ. Le **cofacteur** associé $\overline{[\![i, j]\!]}$ est $V \setminus [\![i, j]\!]$.*



Définition 32
*Soit $\mathcal{P}$ une famille bipartitive de E et $\mathcal{F}$ l'ensemble des bipartitions fortes de $\mathcal{P}$. Une permutation σ de E est **factorisante** pour $\mathcal{P}$ si pour tout $F = \{F^0, F^1\} \in \mathcal{F}$, F^0 est un facteur de σ et F^1 son cofacteur associé, ou l'inverse.*

On se sera alors pas surpris de trouver l'équivalent pour les familles bipartitives des propositions 60 et 61 :

Proposition 63
Toute famille bipartitive admet une permutation factorisante. Toute permutation factorisante pour une famille bipartitive $\mathcal{P}$ correspond à la lecture d'un certain ordonnancement et d'un certain enracinement de $\mathcal{T}_{\mathcal{P}}$.

5.2.4 PC-trees

[HM03] introduit une variante *non-enracinée* du PQ-tree : le PC-tree. C'est un arbre dont les feuilles sont les éléments de V et dont les nœuds sont étiquetés P ou C. Les fils d'un nœud C sont ordonnés. Une permutation σ de V est **compatible** avec le PC-tree T si elle est obtenue en

lisant par parcours en profondeur (ie de gauche à droite dans un certain plongement) les feuilles de T de sorte que les fils d'un nœud C sont visités soit dans l'ordre de ce nœud, soit dans l'ordre inverse. On part d'une feuille quelconque.

Un PC-tree est **canonique** s'il n'a pas de nœud à un fils, pas de nœud C à deux ou trois fils, et si les fils d'un nœud P ne sont pas des nœuds P. On peut facilement rendre canonique tout PC-tree. $\{V^0, V^1\}$ avec $V = V^0 \uplus V^1$ est une **bipartition compatible** d'un PC-tree T si, dans toute permutation factorisante compatible σ de T, V^0 ou V^1 est un facteur. L'ensemble de toutes les bipartitions compatibles d'un PC-tree donné est une **famille PC**.

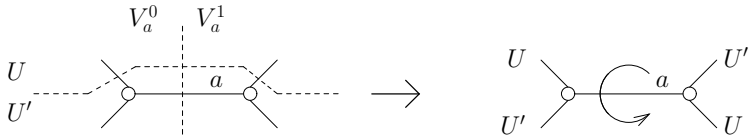
Théorème 30
Les bipartitions compatibles d'un PC-tree sont une famille faiblement bipartitive. L'arbre partitif correspondant à cette famille est exactement le PC-tree canonique : les nœuds P sont des nœuds `premier`*, les nœuds C des nœuds* `linéaire`*, et il n'y a pas de nœud* `complet`*.*

Démonstration: Nous allons montrer

1. une bipartition compatible forte est en bijection avec une arête du PC-tree
2. deux bipartitions compatibles faibles qui se chevauchent correspondent (au sens du lemme 2 page 33) au même nœud du PC-tree.

Soit a une arête du PC-tree T. $T - a$ est coupé en deux composantes : soit V_a^0 l'ensemble des feuilles de la première et V_a^1 l'ensemble des feuilles de la seconde. Soit σ une permutation factorisante compatible de V : elle est obtenue par parcours en profondeur. Un parcours en profondeur parcourt chaque arête d'un arbre deux fois, et entre les deux visite toutes les feuilles du sous-arbre. Selon le côté de a où le parcours commence, V_a^0 ou V_a^1 est un facteur de σ : $\{V_a^0, V_a^1\}$ est une bipartition compatible.

Une bipartition $\{U, U'\}$ obtenue par parcours en largeur et qui chevauche $\{V_a^0, V_a^1\}$ ne peut être compatible : en retourant des fils d'une extrémité de A on produit une permutation qui n'a ni U ni U' comme facteur. Donc $\{V_a^0, V_a^1\}$ est forte.



Un parcours en largeur, produisant une permutation σ, commence à un nœud N et revient à ce même nœud N. Une bipartition compatible énumère tous les fils de N dans un certain ordre. Donc toutes les bipartitions faibles qui se croisent correspondent au même nœud N.

□

5.2.5 Intérêt du concept

La possession d'une permutation factorisante pour une famille partitive ou bipartitive $\mathcal{P}$ apporte moins d'information que la possession de l'arbre $\mathcal{T}_\mathcal{P}$, puisque lire la permutation seule ne permet pas de connaître les facteurs. Une permutation factorisante n'est donc pas un moyen de

représenter les éléments d'une famille (bi)partitive. De plus, il y a unicité de $\mathcal{T}_{\mathcal{P}}$ mais pluralité de $\mathfrak{S}_{\mathcal{P}}$. On peut alors se demander quel intérêt il y a à considérer les permutations factorisantes. Elles ont été introduites pour des raisons *pratiques*.

Au niveau algorithmique, et pour certains cas particuliers de familles (bi)partitives — en l'occurrence des décompositions de graphe, dans le cadre de ce mémoire—, il est plus facile (au niveau de la simplicité de l'algorithme, et —pourquoi pas— au niveau de la complexité) d'obtenir une permutation factorisante que d'obtenir directement l'arbre de décomposition. Ainsi Roland Ducourneau et Michel Habib [DH89] ont remarqué que une extension linéaire, obtenue par parcours en largeur, d'un *graphe d'héritage*[4] fournit une permutation factorisante pour la *décomposition en blocs* de ce graphe. Parallèlement, Hsu et Ma [HM91, Hsu92] établissent que le parcours LexBFS avec heuristique de la cardinalité maximale[5] donne pour la classe des *graphes triangulés* une permutation factorisante pour la décomposition modulaire. Ce résultat se trouve aussi dans [Cap97b]. Michel Habib et Christophe Paul [HP01] travaillent sur un algorithme d'une grande simplicité (mais demandant toutefois plus de travail que LexBFS) pour la décomposition modulaire d'une autre classe, les *cographes*. On remarquera que c'est exactement la classe des graphes dont la décomposition modulaire est *dégénérée*, c'est-à-dire que l'arbre partitif associé ne possède pas de nœud `premier`. Tous ces algorithmes ont un code simple et une complexité $O(n+m)$, ce qui établit que pour certaines décompositions et certaines classes de graphes il est facile d'obtenir une permutation factorisante.

Les deux chapitres suivants présentent des algorithmes de calcul de permutations factorisantes pour le cas des graphes, des tournois et des graphes orientés. Celui des tournois est extrêmement simple, les deux autres un peu moins. Tous sont en $O(n+m)$.

Tout cela est fort bien, mais une nouvelle fois quel est l'intérêt d'obtenir une permutation factorisante, puisqu'elle comporte intrinsèquement moins d'informations qu'un arbre (bi)partitif ? La réponse est double.

Primo, certains problèmes de graphes peuvent être facilement résolus avec l'aide seule d'une permutation factorisante ; l'arbre de décomposition n'est pas nécessaire. Par exemple un algorithme glouton peut colorier les cographes suivant une permutation factorisante.

Secundo, la connaissance d'une permutation factorisante est un outil inestimable pour calculer un arbre de décomposition. Le concept de permutation factorisante a été développé par Christian Capelle et Michel Habib [CH97] dans le but de *simplifier* les algorithmes de décomposition modulaire existants.

Depuis le début des années 1990 il existe des algorithmes de décomposition modulaire en temps linéaire [CH94, MS99, DGM97], aboutissement d'une longue quête de la complexité optimale. Mais ces algorithmes sont un peu frustrants car leur complexité « conceptuelle » est grande, ce qui rend leur compréhension malaisée, leurs preuves lourdes, et leur implémentation rebutante. La permutation factorisante est apparue comme l'outil qui allait permettre la simpli-

4. le graphe orienté acyclique qui représente la relation d'héritage entre les classes d'un langage donné. Il s'agit d'un ordre, possédant un seul plus petit élément : la classe `Object`.

5. C'est une version améliorée du parcours en largeur, possédant des propriétés très intéressante sur les graphes triangulés et les graphes sans triplet astéroïde, voir par exemple [Pau98] ou [Lan01].

fication. La stratégie employée est la suivante : dans un premier temps, obtenir une permutation factorisante. Et dans un deuxième temps construire l'arbre de décomposition en s'appuyant sur la connaissance de cette permutation. Les algorithmes présentés dans cette thèse utilisent cette approche « deux-temps » pour construire la décomposition modulaire de graphes non-orientés, de tournois ou de graphes orientés.

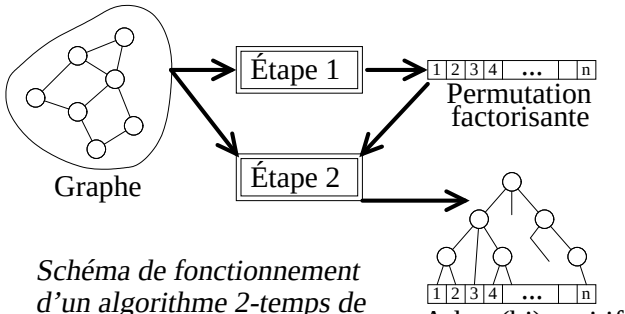


Schéma de fonctionnement d'un algorithme 2-temps de décomposition de graphes

5.3 Ordres factorisants

Une permutation est un ordre *total*. Le caractère *factorisant* des permutations peut aussi être formulé dans le cadre plus vaste des ordres *partiels*.

Un ordre $\mathcal{O} = (V, \prec_{\mathcal{O}})$ est une relation binaire antisymétrique transitive sur V. On a les notations $x \prec_{\mathcal{O}} y$ et $x \preccurlyeq_{\mathcal{O}} y$ pour deux éléments *comparables*, $x||_{\mathcal{O}}y$ pour deux éléments incomparables. On utilisera encore les notations de gauche et droite pour exprimer la préséance. Pour $x \prec_{\mathcal{O}} y$, x est un *prédécesseur* de y, qui est son *successeur*, si aucun sommet ne vient s'intercaler entre eux.

Deux parties disjointes A et B de V se **croisent** si

$$\exists a, a' \in A \; \exists b, b' \in B \;\; a \prec_{\mathcal{O}} b \; et \; a' \succ_{\mathcal{O}} b'$$

cela est noté $A \perp_{\mathcal{O}} B$.

Définition 33
*Étant donné un graphe G, $\mathcal{O}$ est un **ordre factorisant** de $V(G)$ si deux modules forts de G ne se croisent jamais dans $\mathcal{O}$.*

Les permutations factorisantes sont exactement les ordres factorisants totaux. Dans un ordre factorisant, aucun sommet étranger à un module fort M ne vient s'intercaler dans ce module :

$$\forall x \prec_{\mathcal{O}} y \prec_{\mathcal{O}} z, \;\; \{x, z\} \subset M \;\Rightarrow\; y \in M \tag{5.3}$$

Cependant, cette propriété, bien qu'équivalente à la définition dans le cas des ordres totaux, est plus faible qu'elle en général [6].

Théorème 31
Un ordre $\mathcal{O}$ est factorisant si, et seulement si, il existe une extension linéaire de $\mathcal{O}$ qui soit une permutation factorisante.

Démonstration: La partie « seulement si » du théorème étant triviale, démontrons l'autre, par récurrence sur les modules forts de G. Soient $M_1 \dots M_k$ les modules forts maximaux de G, et $\mathcal{O}$ un ordre factorisant G. Prenons deux modules forts M_i et M_j. S'il existe une comparaison entre deux de leurs sommets, par exemple $x_i \in M_i \prec_{\mathcal{O}} x_j \in M_j$, alors $M_i \preccurlyeq_{\mathcal{O}} M_j$, car ces deux modules ne se croisent pas. On est libre de rajouter toutes les comparaisons possibles entre ces deux modules. On prend n'importe quel ordre total sur $\{1, \dots k\}$ compatible avec ces comparaisons pour étendre $\mathcal{O}$. L'ordre obtenu est bien factorisant. Il ne reste des incomparabilité qu'à l'intérieur des modules forts, sur lesquels on repart récursivement. À la fin, l'ordre sur V est total.

□

Extension d'ordres et affinage de partitions

Paige et Tarjan [PT87] ont formalisé le paradigme algorithmique d'*affinage de partitions*. Dans une variante, les classes d'une partition sont ordonnées. Présentons d'abord la classe restreinte d'ordres manipulée par ces algorithmes : les *partitions ordonnées*

Une partition ordonnée est une collection $\mathcal{P} = \{C_1, \dots C_k\}$ de parties de V (non vides, deux à deux disjointes et dont l'union est V) à laquelle on rajoute une relation d'ordre $\mathcal{O}$. Pour $x \in C_i$ et $y \in C_j$, $x \prec_{\mathcal{O}} y$ si et seulement si $i < j$. On notera $\mathcal{P} = C_1 \oplus C_2 \dots \oplus C_k$, où l'opérateur $\oplus$ entre deux classes exprime qu'elle se suivent dans l'ordre total des classes.

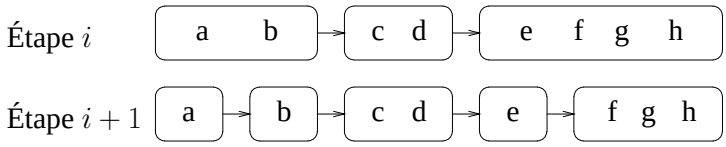
Un petit théorème est que un ordre est une partition orientée si, et seulement si, la relation d'incomparabilité duale de $\mathcal{O}$ est une équivalence. Si $\mathcal{O}$ est un ordre total, la partition n'a que des *singletons* (des classes à un seul sommet) et correspond donc à une permutation.

Une partition ordonnée $\mathcal{P}$ est un ordre partiel bien particulier : les classes sont des ensembles de sommets tous incomparables deux-à-deux, tandis que deux sommets de classes différentes sont ordonnés par les numéros de leurs classes (c'est un ordre total). Les anglo-saxons appellent cela un *weak order*, tandis que les francophones parlent d'*ordre fort* [7].

6. Appelons « faiblement factorisants » les ordres qui vérifient cette deuxième définition. Le $2K_2$ a–b c–d, avec $\mathcal{O}$ défini par $a \prec c$ et $d \prec b$, est faiblement factorisant, mais pas fortement. Dans le cas faible, étant donné *un* module fort, il existe *une* extension linéaire ou ce module est un facteur. Tandis que dans le cas normal il existe *une* extension linéaire ou *tous* les modules sont des facteurs.

7. Le raison de cette loufoquerie est sans doute le sens donné au mot *ordre* : pour nous il s'agit d'une relation antisymétrique transitive, et nous appelons *ordre total* une relation d'ordre comme celle de $\mathbb{N}$. Une partition orientée est alors un ordre qui est juste en-dessous de « total », donc « fort ». Tandis que les anglo-saxons préfèrent parler de *partially ordered sets (posets)* et utilisent en général le mot *order* pour les ordres totaux. Une partition est donc plus « faible » qu'un tel ordre...

En parlant de partitions ordonnées, on jongle entre le monde des familles de parties et celui des ordres. L'affinage de partition consiste en le fait de couper des classes de la partition en plusieurs parties. Les classes issues de la césure de C sont consécutives dans la nouvelle partition, qui devient plus *fine*. En termes d'ordres, on peut voir cela comme l'ajout de comparaisons dans $\mathcal{O}$. On peut donc aussi voir cette opération comme une *extension d'ordres*.



Affinage de partition (extension d'ordre).

Le point de départ d'un algorithme d'affinage de partitions est une partition $\mathcal{P} = \{V\}$, correspondant à l'ordre trivial. Puis la partition est affinée jusqu'à ce que toutes les classes soient des singletons. On peut appliquer ce schéma à la décomposition modulaire : [Pau98] fournit un algorithme de calcul d'une partition modulaire maximale, et [HPV99] donne un algorithme en $O(m \log n)$ de calcul de permutations factorisantes, exposé en §6.1.

5.4 Quatre outils algorithmiques

Nous présentons ici des algorithmes et structures de données qui seront utilisés dans les chapitres suivants comme outils.

5.4.1 Tri des arcs d'un graphe

Soit σ une permutation (quelconque) des sommets d'un graphe G. Ici est présentée une méthode (venant du folklore des algorithmes de graphes) permettant de trier toutes les listes d'adjacence de G selon σ, pour un temps d'exécution total $O(n + m)$.

Il existe des algorithmes *linéaires* de tri (voir [CLR90] par exemple), qui brisent la traditionnelle barrière du $\Theta(n \log n)$, en s'abstenant de faire des comparaisons. Ces algorithmes ont besoin pour fonctionner que les éléments à trier soient des entiers bornés par n (en fait dans un facteur $[-kn, \dots kn]$ où k est une constante indépendante de n). Ce n'est pas le cas si l'on trie chacune des listes d'adjacence indépendemment (les éléments des listes d'adjacence sont compris entre 1 et n alors que les listes peuvent être plus courtes que n), mais si l'on les trie *toutes ensemble* on y arrive. Considérons la liste des m arêtes du graphe, qui sont des couples (x, y) de nombres de $[1, \dots n]$. Cette liste est triée d'abord suivant le *second* élément du couple, puis on re-trie la liste suivant le *premier* élément du couple. Il existe des tris linéaires *stables*, c'est-à-dire qui ne changent pas l'ordre d'éléments ayant la même clef de tri. Donc ces deux tris fournissent la concaténation de toutes les listes d'adjacence triées, qu'on n'a plus qu'à redécouper en n listes.

5.4.2 Calcul des séparateurs de deux sommets

Ici est présenté un algorithme calculant, pour deux sommets x et y, l'ensemble $\mathrm{Sep}(\{x,y\})$ des sommets qui distinguent x de y. Là encore il s'agit d'un algorithme traditionnel.

On suppose que toutes les listes adjacences du graphe sont triées selon une même permutation σ, quitte à utiliser l'algorithme décrit ci-dessus. Étant donné deux listes $L(x)$ et $L(y)$ triées selon σ, calculer les éléments présents dans l'une mais non dans l'autre se fait très simplement, par un parcours simultané des deux listes. On fait avancer un pointeur par liste, et celui qui pointe le plus petit élément avance d'un cran. Ainsi, on détecte en $O(|L(x)|+|L(y)|)$ tout élément présent dans une de ces listes et pas dans l'autre.

Dans le cas des graphes non-orientés, $\mathrm{Sep}(\{x,y\}) = \Gamma(x)\Delta\Gamma(y)$, donc avec cette technique $\mathrm{Sep}(\{x,y\})$ se calcule facilement en $O(|\Gamma(x)| + |\Gamma(y)|)$. Pour les graphes orientés on a par exemple l'identité

$$\mathrm{Sep}(\{x,y\}) = (\Gamma^{\pm}(x)\Delta\Gamma^{\pm}(y)) \cup (\Gamma^{+}(x)\Delta\Gamma^{+}(y)) \cup (\Gamma^{-}(x)\Delta\Gamma^{-}(y))$$

Si le graphe est représenté sous forme des trois listes d'adjacences par sommet $\Gamma^{\pm}(x)$, $\Gamma^{+}(x)$ et $\Gamma^{-}(x)$, on trouve alors $\mathrm{Sep}(\{x,y\})$ en faisant trois parcours simultanés puis en réalisant l'union des trois, et cela se fait en temps $O(|\Gamma(x)| + |\Gamma(y)|)$.

5.4.3 Détection des nœuds modulaire dans un arbre

Cette section décrit un outil qui sera utilisé dans les deux chapitres suivants. Il répond au problème suivant :

Donnée : un graphe $G = (V, E)$ et une famille arborescente de V (sous forme d'un arbre T).

Résultat : marque les nœuds de T qui sont des modules de V.

Pour cela, quatre marques vont être placées sur V. Elles sont relatives à une *permutation factorisante* arbitraire de T, σ.

Définition 34
*Soit N un nœud de T. Le **bord gauche** de N, $bg(N)$, est le plus petit indice dans σ d'un sommet de N, et son **bord droit** $bd(N)$ est le plus grand. Soit $\mathrm{Sep}(N)$ l'ensemble des séparateurs de N (les sommets qui distinguent N). Le **premier séparateur** de N, noté $ps(N)$, est le plus petit indice dans σ d'un sommet de $N \cup Sep(N)$, et son **dernier séparateur** est le plus grand indice dans σ d'un sommet de $N \cup Sep(N)$.*

Proposition 64
Soit N un nœud de T. N est module de G si et seulement si $bg(N) = ps(N)$ et $bd(N) = ds(N)$.

Il est facile de calculer $bg(N)$ et $bd(N)$ pour tous les nœuds de l'arbre : un simple parcours en profondeur suffit. Calculer les premiers et derniers séparateurs de tous les nœuds peut aussi se faire en $O(n+m)$, voici comment.

Soit N un nœud de T ayant comme fils $F_1 \dots F_k$. Un sommet qui distingue N distingue deux sommets de N, donc :

$$ps(N) = \min\left(bg(N),\ \min_{\{x,y\}\subset N} ps(\{x,y\})\right)$$

Si $x \in \mathrm{Sep}(N)$. sépare deux sommets de N, nécessairement il sépare deux sommets de N *consécutifs* selon σ. Donc (on rappelle que $\sigma(x') = 1 + \sigma(x)$) :

$$ps(N) = \min\left(bg(N),\ \min_{\{x,x'\}\subset N} ps(\{x,x'\})\right)$$

Cela ne conduit pas encore à un calcul efficace des séparateurs. Si $\{x, x'\} \subset F_i$ alors le premier séparateur de $\{x, x'\}$ est utilisé à la fois dans le calcul de $ps(F_i)$ et dans celui de $ps(N)$. Pour chaque paire de sommets consécutifs $\{x, x'\}$, il n'est besoin de calculer $ps(\{x, x'\})$ que pour le sommet[8] $N = \mathrm{LCA}(x, x')$. On fait ensuite remonter cette information à tous les nœuds ancêtres de N. On obtient alors les relations suivantes, qui permettent cette fois un calcul efficace :

$$ps(N) = \min\left(bg(N),\ \min_{i=1}^{k-1}\{ps(\{bd(F_i), bg(F_{i+1})\})\}\ ,\ \min_{i=1}^{k}\{ps(F_i)\}\right) \tag{5.4}$$

$$ds(N) = \max\left(bd(N),\ \max_{i=1}^{k-1}\{ds(\{bd(F_i), bg(F_{i+1})\})\}\ ,\ \max_{i=1}^{k}\{ds(F_i)\}\right) \tag{5.5}$$

On vient de voir que $\mathrm{Sep}(\{x, x'\})$ se peut calculer en $O(|\Gamma(x)| + |\Gamma(x')|)$, ce qui permet de trouver $ps(\{x, x'\})$ et $ds(\{x, x'\})$ avec la même complexité. Cette fonction n'est appelée qu'entre sommets *consécutifs* dans σ. x n'est utilisé que dans le calcul de $LCA('x, x)$ et dans celui de $LCA(x, x')$, une fois par chaque équation, donc quatre fois en tout. La complexité totale des appels à Sep est donc $O(n + m)$. Les autres opérations effectuées prennent $O(n)$ en tout, car les valeurs d'un nœud ne sont utilisées que par son père.

5.4.4 Le tarbre, une structure de données pour le calcul des extrema

Nous allons considérer deux problèmes de calcul d'extrema. Pour les deux, la **donnée** est une fonction $f : [\![1, n]\!] \mapsto \mathbb{N}$ injective ($f(x) = f(y) \iff x = y$). Une **requête** est un intervalle $[\![a, b]\!]$ avec $1 \leqslant a \leqslant b \leqslant n$. Le problème du **Minimum d'un Intervalle (MinInt)** consiste à renvoyer comme **résultat** d'une requête le nombre x tel que $a \leqslant x \leqslant b$ et $f(x)$ est le minimum de f sur $[\![a, b]\!]$. Et le problème du **Maximum d'un Intervalle (MaxInt)** est identique mais demande le maximum. Ces opération, appellées *interval range queries*, ont été étudiées par [GBT84] qui proposent une technique que nous présentons ici.

Soit m le maximum de f sur $[\![1, n]\!]$. Comme f est injective, $m \geqslant n$. f peut être vue comme un ensemble de n couples $(x, f(x))$. Un tableau de taille $O(n)$ permet donc de calculer $f(x)$

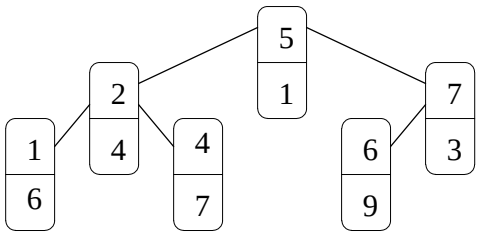
8. où $LCA(x, y)$ est le *plus petit ancêtre commun*, ou *least common ancestor*, de x et y, c'est-à-dire le nœud de profondeur maximale dont descendent x et y.

en temps $O(1)$. Nous allons voir une autre structure de donnée stockant f en espace $O(n)$ et permettant de répondre à une requête MinInt ou MaxInt en temps $O(1)$. Sa construction prend $O(m)$. Le concept est dû à [SA96].

Définition 35
*Soit $f : [\![1, n]\!] \mapsto \mathbb{N}$ une fonction injective. Un **tarbre croissant**[9] de f est un arbre binaire enraciné de sommets $\{1, \dots n\}$ et vérifiant*

- *Pour tout nœud x, tout descendant y de ce nœud vérifie $y < x$ si y descend du fils* gauche *de x et $x < y$ si y descend du fils* droit *de x.*
- *Tout descendant y d'un nœud x vérifie $f(x) < f(y)$.*

*Un **tarbre décroissant** est défini de la même façon mais tout descendant y de x vérifie $f(x) > f(y)$.*



Un tarbre croissant. Le nombre en haut est x et en bas $f(x)$.

La raison pour laquelle nous utilisons le tarbre est la suivante :

Proposition 65
Étant donné un tarbre croissant (resp. décroissant) de f, le résultat d'une requête $[\![a, b]\!]$ du problème MinInt (resp. MaxInt) est le dernier ancêtre commun de a et b dans le tarbre.

Démonstration: Soit m le dernier ancêtre commun de a et b dans un tarbre croissant. a descend du fils gauche de m et b de son fils droit, donc $a < m < b$. Tout x tel que $a < x < m$ ou $m < x < b$ descend de m. Or m minimise f dans son sous-arbre, donc est le résultat recherché.

□

Étant donné un arbre à n nœuds, il existe des algorithmes [HT84, AGKR02] qui calculent $LCA(x, y)$ pour deux nœuds x et y donnés, avec un temps de pré-traitement de $O(n)$ par arbre (il s'agit de mettre des étiquettes sur les nœuds) et un temps de traitement de $O(1)$ par nœud (un nombre constant d'opération entre les étiquettes de x et celles de y). Or le tarbre peut se construire en $O(n)$ par l'algorithme suivant :

Le tarbre décroissant se construit de la même façon en testant $f(y) < f(z)$. Trier au moins n nombres entiers bornés par m peut se faire en temps $O(m)$. La boucle de l'algorithme tourne $n - 1$ fois en temps constant. Donc l'algorithme s'exécute en temps $O(n + m)$.

9. tas+arbre. Copié sur l'anglais *treap*, tree+heap.

Algorithme 2: Construction d'un tarbre croissant de f

Données : Une fonction injective $f : [\![1, n]\!] \mapsto \mathbb{N}$
Résultat : Un tarbre croissant de f
début
  générer tous les couples $(x, f(x))$
  faire une liste doublement chaînée L_1 de ces couples triés selon les x croissants
  faire une deuxième liste chaînée L_2 triée selon les $f(x)$ décroissants
  pour *i de 1 à $n-1$* **faire**
    Soit $X = (x, f(x))$ le $i$eme couple de L_2
    Soit $Y = (y, f(y))$ le couple précédant X dans L_1 et $Z = (z, f(z))$ le couple suivant X dans L_1
    si $f(y) > f(z)$ *(ou si Z n'existe pas)* **alors**
      $Pere(X) = Y$
      $FilsGauche(Y) = X$
    sinon
      $Pere(X) = Z$
      $FilsDroit(Z) = X$
    fin
    Enlever X de L_1 (en joignant Y et Z)
  fin
fin

Montrons que cet algorithme produit bien un tarbre croissant[10]. Dans la preuve, une lettre majuscule comme X représente un couple comme $(x, f(x))$. L'algorithme produit un arbre, car tout sommet X, sauf le dernier de L_1, reçoit un père Y avec $f(x) > f(y)$ (car X est placé avant Y dans L_2). Par récurrence sur la descendance de X, y minimise f dans le sous-arbre dont il est racine.

Une arête entre X et Z passe *au-dessous* de Y si $x < y < z$ et $f(y) = \min(f(x), f(y), f(z))$. Dans L_2 X et Z viennent avant Y. X n'a donc pu être enlevé de L_1 et, dans L_1, est placé entre X et Z. Donc X n'a pas pu être attribué comme père de Z, ni l'inverse : aucune arête ne passe en-dessous d'un sommet. Soient R la racine de l'arbre, G un fils gauche de R, X un descendant de G, D un fils droit de R et Y un descendant de G. Par construction $g < r < d$. Si $x > r$ alors il existe une arête qui passe en-dessous de R, car à cause de la connexité de l'arbre il y a un descendant G' de G et un ascendant X' de X tel que $g' < r < x'$ et $G' = Pere(X')$. Donc $x < r$ ssi X descend d'un fils gauche de R. Par récurrence, on démontre cela pour tout nœud de l'arbre.

10. En fait le tarbre croissant (resp. décroissant) est unique.

Il reste à montrer que l'arbre est binaire. Soit X un nœud ayant deux fils droits Y et Y'. Sans perte de généralité, Y est le premier fils inséré et Y' le second. Dans L_1 (avant suppressions) on a X puis Y puis Y'. Et dans L_2 Y puis Y' puis X. Tout nœud Z entre Y et dans L_1 Y' est situé avant eux dans L_2, donc déjà inséré, sinon Y deviendrait fils de ce nœud. Au moment d'insérer Y, son père est Y', car $f(y') > f(x)$, contradiction. On démontre de même qu'un nœud ne peut avoir deux fils gauches. Finalement :

Théorème 32
Étant donné une fonction injective $f : [\![1, n]\!] \mapsto \mathbb{N}$ bornée par m, un tarbre croissant ou décroissant de f peut être construit en temps $O(m)$ et une requête du problème MinInt ou MaxInt peut être traitée en $O(1)$.

5.5 Algorithme d'intersection de familles partitives

L'algorithme présenté ici a été écrit avec Ross McConnell. Il a surgi comme sous-problème de notre algorithme de décomposition modulaire des graphes orientés (chapitre 7). Les familles partitives à intersecter (donnée du problème) sont représentées sous la forme d'un arbre ; l'algorithme fournit l'arbre représentant l'intersection de la famille. Il est linéaire en la *taille* de la famille, c'est-à-dire $O(n^2)$ dans le cas général[11], mais $O(n+m)$ dans le cas de la décomposition modulaire des graphes (théorème 35).

5.5.1 Définition du problème

Soient $\mathcal{F}_a$ et $\mathcal{F}_b$ deux familles partitives sur le même ensemble V. L'**intersection** de $\mathcal{F}_a$ et $\mathcal{F}_b$ est $\mathcal{F} = \mathcal{F}_a \cap \mathcal{F}_b$, c'est-à-dire la famille des parties de V appartenant aux deux familles.

Proposition 66
L'intersection de deux familles partitives est une famille partitive.

Démonstration: Soient $X, Y \in (\mathcal{F}_a \cap \mathcal{F}_b)$. Si E et F se chevauchent, alors, puisque $\mathcal{F}_a$ est partitive, $X \cup Y$, $X \cap Y$ et $X \Delta Y$ sont membres de $\mathcal{F}_a$. Et puisque $\mathcal{F}_b$ est partitive, ces trois ensembles sont également membres de $\mathcal{F}_b$. Ils sont donc dans $\mathcal{F}_a \cap \mathcal{F}_b$, qui est de ce fait partitive. □

Cela permet de définir un opérateur entre arbres de décompositions de V. Étant donné deux familles partitives $\mathcal{F}_a$ et $\mathcal{F}_b$ munies de leurs arbres de décomposition T_a et T_b, $T_a \wedge T_b$ est l'arbre partitif de $\mathcal{F}_a \cap \mathcal{F}_b$.

Étant donnée une famille (quelconque) $\mathcal{F}$ de parties de V, on pose $Taille(\mathcal{F}) = \sum_{E \in \mathcal{F}} |E|$, et $Forts(\mathcal{F})$ est l'ensemble des parties de $\mathcal{F}$ qui n'en chevauchent aucune autre. Ce chapitre fournit un algorithme de calcul de $T_a \wedge T_b$, linéaire en $Taille(Forts(\mathcal{F}_a)) + Taille(Forts(\mathcal{F}_b))$. Dans

11. Une famille arborescente possède $O(n)$ éléments de taille $O(n)$.

le reste du chapitre, $\mathcal{F}_a$ et $\mathcal{F}_b$ représenteront deux familles partitives données, $\mathcal{F}$ leur intersection, et T_a, T_b et $T = T_a \wedge T_b$ les arbres partitifs de ces trois familles.

5.5.2 Propriétés

Étant donné une partie $F \in \mathcal{F}_a$, $Pere_a(F)$ est la plus petite partie forte de $\mathcal{F}_a$ contenant F. Si F est faible, $Pere_a(F)$ est `complet` et F est l'union de plusieurs fils de F dans T_a, d'après le théorème 1. On définit de même $Pere_b(F)$.

Étant donné une famille (quelconque) $\mathcal{F}$ de parties de V, le **graphe de chevauchement** de $\mathcal{F}$ est $G_{\mathcal{F}}^{\oslash} = (\mathcal{F}, E_{\mathcal{F}}^{\oslash})$ avec $(X, Y) \in E_{\mathcal{F}}^{\oslash}$ ssi $X \oslash Y$. Les composantes connexes de $G_{\mathcal{F}}^{\oslash}$ seront appelées **classes de chevauchement** de $\mathcal{F}$. On pose

$$\mathcal{S} = Forts(\mathcal{F}_a) \cap Forts(\mathcal{F}_b)$$

Lemme 34 *Soit $\mathcal{Z}$ une famille de partie et $\mathcal{C}$ une classe de chevauchement de $\mathcal{Z}$. Si $X \subset V$ chevauche $\bigcup_{C \in \mathcal{C}} E$, alors il existe un élément Y de $\mathcal{C}$ qui chevauche X.*

Démonstration: X n'est contenu dans aucun membre de $\mathcal{C}$, puisqu'il chevauche leur union à tous. Supposons que X ne chevauche aucun membre de $\mathcal{C}$. Alors on peut partitionner $\mathcal{C}$ en $\mathcal{A} \uplus \mathcal{B}$ tels que X n'intersecte aucun membre de $\mathcal{A}$ et contienne tous les membres de $\mathcal{B}$. Puisque X chevauche l'union de $\mathcal{C}$, ces deux parties sont non vides. Aucun membre de $\mathcal{A}$ ne chevauche pourtant de membre de $\mathcal{B}$, ce qui contredit que $\mathcal{C}$ soit une classe de chevauchement.

□

Soit $\mathcal{U}$ l'ensemble des unions de composantes de chevauchement :

$$\mathcal{U} = \{ \bigcup_{X \in \mathcal{C}} X \mid \mathcal{C} \text{ est une classe de chevauchement de } \mathcal{S} \}$$

Clairement $V \in \mathcal{U}$ et $\{v\} \in \mathcal{U}$ pour tout $v \in V$. D'après le lemme 34, deux membres de $\mathcal{U}$ ne peuvent se chevaucher : $\mathcal{U}$ est donc une famille arborescente. Mais de plus aucun membre de $\mathcal{S}$, donc aucune partie forte de $\mathcal{F}_a$ ou $\mathcal{F}_b$, ne chevauche de membre de $\mathcal{U}$. Or, aucune partie de $\mathcal{F}$ ne chevauche de membre de $\mathcal{S}$, donc de membre de $\mathcal{U}$. En notant $T_{\mathcal{U}}$ l'arbre d'inclusion de $\mathcal{U}$:

Lemme 35 *Tout membre de $\mathcal{F}$, de même que tout membre de $\mathcal{S}$, est l'union de nœuds frères de $T_{\mathcal{U}}$.*

$\mathcal{U}$ contient « trop » d'éléments, ce qui nous amène à définir

$$\mathcal{V} = \mathcal{U} \cap \mathcal{F}_a \cap \mathcal{F}_b$$

On munit $\mathcal{V}$ de la relation suivante : $X \mathcal{R} Y$ ssi $Pere_a(X) = Pere_a(Y)$ et $Pere_b(X) = Pere_b(Y)$. $\mathcal{R}$ est clairement une relation d'équivalence. Soit

$$\mathcal{F}' = \mathcal{V} \cup \{ \bigcup_{X \in \mathcal{D}} X \mid \mathcal{D} \text{ est une classe d'équivalence de } \mathcal{R} \}$$

Théorème 33

$$\mathcal{F}' = Forts(\mathcal{F})$$

Démonstration:

1. $\mathcal{F}' \subset Forts(\mathcal{F})$
   Par construction $\mathcal{V} \subset \mathcal{F}$, et d'après le lemme 34 les parties de $\mathcal{V}$ sont fortes dans $\mathcal{F}$. L'union d'une classe de $\mathcal{R}$ contient des sommets qui sont les feuilles de nœuds frères, fils d'un `complet` A de $\mathcal{F}_a$, et également feuilles d'autres nœuds frères, fils d'un `complet` B de $\mathcal{F}_b$, donc une telle union est bien membre de $\mathcal{F}$. De plus toute union partielle de membres $\mathcal{R}$-équivalents est aussi dans $\mathcal{F}$, mais est faible car on peut exhiber une autre union partielle qui la chevauche. Ce n'est pas le cas de l'union de toute la classe, qui est donc forte.
2. $Forts(\mathcal{F}) \subset \mathcal{F}'$
   Soit $X \in Forts(\mathcal{F})$. Si $X \in \mathcal{S}$, alors X est dans une classe de chevauchement triviale, donc $X \in \mathcal{U}$, $X \in \mathcal{V}$ et finalement $X \in \mathcal{F}'$. Sinon, X est faible dans $\mathcal{F}_a$ comme dans $\mathcal{F}_b$. Soient $A = Pere_a(X)$ et $A_1, \ldots A_k$ les fils de A contenant X (un fils de A ne pouvant pas chevaucher X), et $B = Pere_b(X)$ et $B_1, \ldots B_l$ ses fils contenant X. On a vu qu'une classe de chevauchement de $G_{\mathcal{F}}^{\bigcirc\!\!\!\bigcirc}$ ne peut chevaucher X, donc X est une union de classes de chevauchement. Or tout A_i et tout B_i est inclus dans au moins une classe de chevauchement contenue dans X, ce qui n'est pas la cas de A ni de B. Soient $\mathcal{C}_1, \ldots \mathcal{C}_z$ les classe de $G_{\mathcal{F}}^{\bigcirc\!\!\!\bigcirc}$ dont l'union est de père A dans $\mathcal{T}_a$, de père B dans $\mathcal{T}_b$, et qui sont contenues dans X. Ces membres de $\mathcal{V}$ sont tous $\mathcal{R}$-équivalents, mieux, ils forment une classe d'équivalence puisqu'aucun autre ne peut leur être équivalent. Leur union est exactement X.

□

5.5.3 Algorithme

Théorème 34
Étant donnés deux arbres T_a et T_b représentant deux familles partitives $\mathcal{F}_a$ et $\mathcal{F}_b$ de V, $T = T_a \wedge T_b$ peut être calculé en $O(|V| + Taille(Forts(\mathcal{F}_a)) + Taille(Forts(\mathcal{F}_a)))$.

Démonstration: L'algorithme proposé consiste simplement en la construction successive de $\mathcal{S}$, $\mathcal{U}$, $\mathcal{V}$, $\mathcal{F}'$ et T, décrite en détails ci-dessous. La complexité est prouvée à chaque étape. La correction vient du théorème 33.

□

Construction de $\mathcal{S}$

Étant donnés T_a et T_b on construit trivialement $Forts(\mathcal{F}_a)$ et $Forts(\mathcal{F}_b)$, triés par tailles croissante (par un simple parcours en profondeur). Ces listes sont ensuite fusionnées en une liste triée. Il ne reste qu'à détecter les doublons, ce qui se peut faire par affinage de partition : on part

d'une partition où chaque classe est formée des parties de même taille, puis on affine selon le premier sommet de V, puis selon le second, etc. Les classes finales contiennent les ensembles identiques. Cet affinage de partitions peut se faire en temps linéaire en l'entrée. On garde un représentant par doublon et on supprime les occurrences uniques.

Construction de $\mathcal{U}$

Elias Dahlhaus a écrit un algorithme [Dah00] qui, étant donnée une famille (quelconque) $\mathcal{F}$ de parties de V, calcule les composantes de chevauchements de $\mathcal{F}$ en $O(|V| + Taille(\mathcal{F}))$. L'algorithme, relativement facile à implémenter, calcule un graphe qui a les mêmes composantes connexes que $G_{\mathcal{F}}^{\bigcirc\!\!\bigcirc}$, mais dont la taille tient dans l'espace $O(Taille(\mathcal{F}))$. Pour calculer les unions de chaque composante, on peut utiliser un tableau T représentant V, initialisé une fois et remis à zéro après chaque usage, donnant chaque union en $O(Taille(\mathcal{C}))$.

Construction de $\mathcal{V}$

$X \in \mathcal{U}$ appartient à $\mathcal{V}$ ssi il est membre de $\mathcal{F}_a$ et de $\mathcal{F}_b$. l'algorithme 3, emprunté à [Spi92], prend comme données un arbre d'inclusion T de V et un ensemble $X \subset V$, et *marque* les nœuds de profondeur maximale de T intersectant X mais non contenus dans X. En l'utilisant sur T_a, on vérifie si $X \in \mathcal{F}_a$ en regardant si soit un seul nœud est marqué et qu'il est `complet` (X est alors une union de ses fils), soit si un seul nœud est marqué, qu'il est `premier` et que X est égal à un de ses fils. Le même calcul sur T_b permet de construire $\mathcal{V}$ à partir de $\mathcal{U}$. Chaque étape prend un temps $O(X)$, après une initialisation en $O(|V|)$ pour tout l'arbre, donnant à chaque nœud une liste de pointeurs sur ses fils, le nombre de ses fils, un compteur du nombre de fils marqués, un booléen pour la marque, et associant à l'arbre une file des nœuds marqués. La linéarité est évidente en considérant l'usage de la file des nœuds marqués, que l'on n'a d'ailleurs qu'à parcourir pour produire le résultat et réinitialiser l'arbre.

Algorithme 3: Marquage des nœuds de plus grande profondeur d'un arbre d'inclusion T de V, qui intersectent $X \subset V$ sans être contenus dans X [Spi92].

début
  pour chaque *feuille* $N = \{x\}$ *de* T, $x \in X$ **faire**
    marquer N
  fin
  pour chaque *nœud* N *de* T *dont tous les fils* $F_1, \ldots F_k$ *sont marqués* **faire**
    Démarquer $S_1, \ldots S_k$
    marquer N
  fin
fin

Construction de $\mathcal{F}'$

l'algorithme de marquage donne dans la foulée, pour chaque $N \in \mathcal{V}$, les nœuds $Pere_a(N)$ et $Pere_b(N)$. Chacun des $k \leqslant |V|$ nœuds peut être codé par la paire $(Pere_a(N), Pere_b(N))$, un numéro de père étant un entier compris entre 1 et $|V|$. On peut donc utiliser un algorithme de tri *linéaire* et *stable* [CLR90], en triant d'abord selon le premier membre puis suivant le second, pour trouver les couples identiques, donc les classes d'équivalence de $\mathcal{R}$. Cela prend $O(|V|)$. Il n'y a plus ensuite qu'à faire l'union de chaque classe, en temps linéaire en la taille d'une classe, en utilisant un tableau de taille $|V|$ réinitialisé après chaque usage.

Construction de T

L'algorithme présenté ici est plus général, puisqu'il calcule l'arbre d'inclusion d'une famille partitive en général. C'est un classique de l'algorithmique.

Lemme 36 *Étant donnée une famille arborescente $\mathcal{F}$, construire son arbre d'inclusion $T_{\mathcal{F}}$ peut se faire en $O(Taille(\mathcal{F}))$.*

Démonstration: Remarquons que $Taille(\mathcal{F}) = \Omega(|V|)$, puisque $V \in \mathcal{F}$. Les $O(|V|)$ membres de $\mathcal{F}$ sont d'abord triés par taille, en temps $O(|V|)$ grâce à un tri linéaire [CLR90]. Puis pour chaque $x \in V$ est créée une liste des membres de $\mathcal{F}$ contenant ce sommet, triée par tailles croissantes. Cela peut être fait en visitant tous les $X \in \mathcal{F}$ par taille décroissante et, pour chaque $x \in X$, en mettant X en tête de la liste de x. Enfin, pour chaque $x \in V$, sa liste est parcourue du plus petit élément vers le plus grand, et un pointeur est ajouté d'un élément vers son successeur, sauf si ce pointeur existe déjà : cela construit T. Ces opérations se font en temps linéaire.

□

5.6 Un résultat de complexité : la taille des modules forts

Le théorème suivant va être un outil pour démontrer la linéarité des algorithmes des chapitres suivants. Il dit que la *taille* des modules forts d'un graphe (au sens de la section précédente) est proportionelle à la taille d'un graphe, et non en $\Theta(n^2)$ comme on pourrait s'y attendre. Donc l'algorithme précédent est linéaire s'il travaille avec des décompositions modulaires, ce qui sera le cas au chapitre 7. Puis ses deux corollaires établissent que si l'on travaille sur chacun des nœuds de $\mathcal{T}_G$ en un temps linéaire en la taille du graphe quotient ou du graphe caractéristique (avec une définition ad hoc) alors le temps de travail total est également linéaire en la taille du graphe de départ ! Ce résultat renforce l'intérêt que l'on peut trouver à travailler sur l'arbre de décomposition modulaire d'un graphe.

Soit G un graphe (orienté ou non) et $\mathcal{M}(G)$ l'ensemble de ses modules forts. On note $|M|$ la taille d'un module, c'est-à-dire le nombre de sommets de ce module.

Théorème 35

$$\sum_{M \in \mathcal{M}(G)} |M| \leqslant 2m + 3n$$

De plus si G est connexe alors

$$\sum_{M \in \mathcal{M}(G)} |M| \leqslant 2m + 2n$$

Démonstration: Appelons $\rho(G)$ la somme $\sum_{M \in \mathcal{M}(G)} |M|$. Nous allons prouver le théorème par induction sur n. Il est clair que pour un graphe à un sommet $\rho(G) = 1$. Supposons maintenant le théorème vrai pour des graphes de 1 à $n-1$ sommets. Prenons un graphe G à n sommets, et examinons sa décomposition modulaire.

Si G est non connexe (cas 2 du théorème de décomposition) alors G se décompose en $k \geqslant 2$ composantes connexes $G_1 \ldots G_k$. Par hypothèse

$$\forall\ 1 \leqslant i \leqslant k,\ \rho(G_i) \leqslant 2m_i + 2n_i$$

De plus, nous observons que les modules forts de G sont exactement ceux contenus dans chacun des G_i (puisqu'aucun module fort ne chevauche une composante connexe) plus un de taille n (qui est $V(G)$). Tout arc de G est présent dans un G_i donc $\sum_i m_i = m$. On a :

$$\begin{aligned}
\rho(G) &= n + \rho(G_1) + \ldots + \rho(G_k) \\
&\leqslant n + 2m_1 + 2n_1 + \ldots + 2m_k + 2n_k \\
&\leqslant n + 2\sum_i n_i + 2\sum_i m_i \\
&\leqslant 2m + 3n
\end{aligned}$$

Si G est connexe (cas 3 à 5 du théorème de décomposition), soient $G_1 \ldots G_k$ les graphes induits par les modules forts maximaux de G. On ne peut plus supposer les G_i connexes. Par hypothèse d'induction (puisque les G_i ont strictement moins de n sommets) nous avons

$$\forall\ 1 \leqslant i \leqslant k,\ \rho(G_i) \leqslant 2m_i + 3n_i$$

Les arcs de G sont de deux sortes : ceux propres à chaque G_i et ceux reliant des G_i entre eux. Il y en a exactement $\sum m_i$ de la première sorte, quant aux seconds, posons qu'il y en ait $m' = m - \sum m_i$. Remarquons que G est connexe (par hypothèse), mais que si l'on enlève à G les arcs internes à chaque G_i, G reste connexe. En effet pour aller de $x \in G_i$ à $y \in G_j$ on n'a besoin d'emprunter aucun arc interne, et pour aller d'un sommet de G_i à un autre, il suffit de prendre un chemin de longueur 2 passant par un des G_j auquel G_i est adjacent. On a donc $m' \geqslant n - 1$.

Il nous faut maintenant distinguer deux cas.

1. Si chaque G_i a au moins deux sommets, alors de chaque sommet $x \in G_i$ partent au moins deux arcs le reliant à G_j, $i \neq j$. On a donc $m' \geqslant n$.

$$\sum m_i \leqslant m - n, \text{et}$$

$$\begin{aligned} \rho(G) &= n + \rho(G_1) + \ldots + \rho(G_k) \\ &\leqslant n + 2m_1 + 3n_1 + \ldots + 2m_k + 3n_k \\ &\leqslant 4n + 2\sum m_i \\ &\leqslant 2n + 2m \end{aligned}$$

2. Si l'un des G_i est réduit à un seul sommet, en supposant que ce soit G_1, on a

$$\sum m_i \leqslant m - n + 1, \text{et}$$

$$\begin{aligned} \rho(G) &= n + \rho(G_1) + \rho(G_2) + \ldots + \rho(G_k) \\ &\leqslant n + 1 + 2m_2 + 3n_2 + \ldots + 2m_k + 3n_k \\ &\leqslant n + 1 + 2\sum m_i + 3(n-1) \\ &\leqslant n + 1 + 2m - 2n + 2 + 3n - 3 \\ &\leqslant 2n + 2m \end{aligned}$$

ce qui démontre le théorème.

□

Corollaire 1
Si, dans un arbre de décomposition modulaire, on applique au graphe quotient G_N de chaque nœud N un traitement en $O(n_N + m_N)$, alors l'ensemble du processus s'exécute en un temps $O(n+m)$.

Corollaire 2
Si, dans un graphe G, on applique à chaque module fort M un traitement $O(|M| + m_M)$ avec $\sum_M m_M = m$ alors l'ensemble du processus s'exécute en un temps $O(n+m)$.

Démonstration: Chaque arc de G apparaît exactement une fois dans l'ensemble des modules. Si donc le processus prend un temps d'exécution $T \leqslant k(n_M + m_M)$ par module M, alors d'après

le théorème précédant il prend en tout

$$\begin{aligned}\sum_M k(|M| + m_M) &= k\sum_M |M| + k\sum_M m_M \\ &\leqslant k(3n + 2m) + km \\ &\leqslant 3k(n + m)\end{aligned}$$

□

Chapitre 6

Permutation factorisante d'un graphe non-orienté : algorithme en $O(n+m)$

Beware of bugs in the above code ;
I have only proved it correct, not tried it.

Donald KNUTH

IL EXISTE DÉJÀ des algorithmes linéaires de décomposition modulaire des graphes orientés (voir début du chapitre précédent). Pourtant, ce chapitre en présente un nouveau. Quelle peut être la motivation d'un tel travail ? La réponse est la *simplicité*, par-rapport aux autres algorithmes existants. Cet algorithme (ou plutôt la moitié d'algorithme présentée ici, l'autre partie étant décrite dans la chapitre 8), grâce à l'approche *deux-temps*, le premier temps calculant une permutation factorisante et le deuxième temps l'arbre, prétend en effet être plus simple que les algorithmes existants. Ces idées sont le fruit d'un travail réalisé avec Michel Habib et Christophe Paul, dans la continuité d'une série de travaux de ces derniers, avec Laurent Viennot, sur le même sujet [Pau98, HPV99, HP01].

Nous présentons tout d'abord un algorithme écrit par Michel Habib, Christophe Paul et Laurent Viennot, qui est le point de départ de ce travail. Il utilise des méthodes d'*affinage de partition* [PT87], en utilisant les adjacences des sommets comme pivots pour affiner une partition, qui devient une permutation factorisante. L'usage de la *règle de Hopcroft* [Hop71], celle de la minimisation des automates finis, permet d'atteindre une borne de $O(n+m\log n)$. Pour briser la borne, nous avons introduit une nouvelle structure, les partitions ordonnées de chaînes, et une nouvelle *règle* aux deux de [HPV99].

6.1 Partitions ordonnées. Algorithme de Habib, Paul & Viennot

Dans [HPV99] est décrit un algorithme qui calcule une permutation factorisante [1] d'un graphe non-orienté. Il travaille par affinage de partitions ordonnées, en suivant *deux* règles de césure des classes. Nous le décrivons ici, car il constitue la base de l'algorithme décrit ci-après. En effet, la complexité atteinte par Habib, Paul et Viennot est de $O(m \log n)$. L'algorithme linéaire reprend ces deux règles d'affinage et en introduit une troisième, et passe des partitions ordonnées aux partitions ordonnées de chaînes.

Voici les deux règles d'extension de [HPV99]. Tout d'abord, il faut amorcer la pompe. On se base sur le lemme suivant :

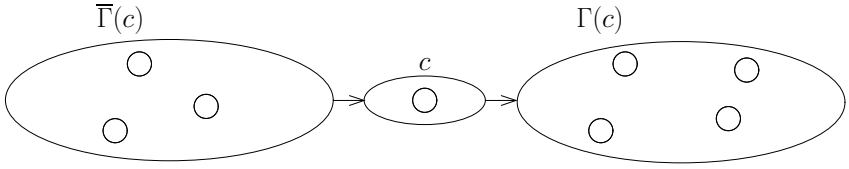
Lemme 37 *[HPV99] Étant donné un sommet c, la partition $\overline{\Gamma}(c) \oplus \{c\} \oplus \Gamma(c)$ est factorisante.*

c est appelé le **centre** de la partition, il joue un rôle particulier dans les preuves. Affiner de cette façon constitue la *règle du centre* que l'on utilise une fois. Les affinages suivants sont faits avec la *règle du pivot*, basée sur l'observation suivante :

Lemme 38 *[HPV99] Soit $\mathcal{O} = C_1 \oplus \ldots \oplus \{c\} \oplus \ldots \oplus C_k$ une partition ordonnée de centre c, et soit $p \in C_i$ un sommet tel que la classe C_j, $i \neq j$, chevauche $\Gamma(p)$. Si $\mathcal{O}$ est factorisant alors*

- *si $C_i \prec_{\mathcal{O}} \{c\} \prec_{\mathcal{O}} C_j$ ou si $C_j \prec_{\mathcal{O}} \{c\} \prec_{\mathcal{O}} C_i$, la partition où C_j est remplacée par $(\overline{\Gamma}(p) \cap C_j) \oplus (\Gamma(p) \cap C_j)$ est factorisante ;*
- *sinon, la partition où C_j est remplacée par $(\Gamma(p) \cap C_j) \oplus (\overline{\Gamma}(p) \cap C_j)$ est factorisante.*

p est appelé le **pivot** de cette étape d'extension. La règle du pivot consiste donc à couper en deux toutes les classes qui chevauchent le voisinage de p (sauf la sienne). Cela peut être fait par un simple parcours de la liste d'adjacence de p. Le plan de cet algorithme est donc :

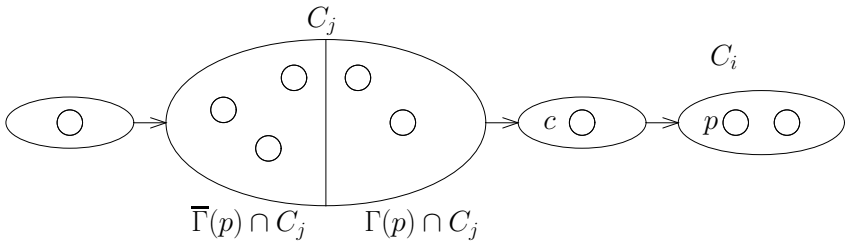


Règle du centre.

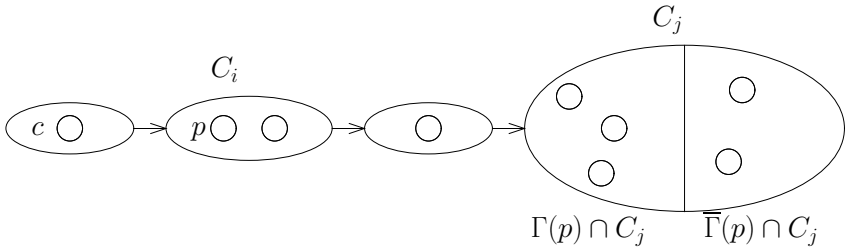
1. on abrègera dans la suite de ce mémoire « permutation factorisante de la famille des modules d'un graphe G » en « permutation factorisante d'un graphe G ».

Algorithme 4: Vue d'ensemble de l'algorithme de [HPV99]

début
  Choisir un centre c
  créer $\mathcal{O}$ selon $\Gamma(c)$ par la règle du centre
  répéter
    Choisir une classe C
    pour chaque $p \in C$ **faire**
      étendre $\mathcal{O}$ selon $\Gamma(p)$ par la règle de pivot
    fin
  jusqu'à *la partition ne puisse plus être étendue*
fin



Règle du pivot (premier cas).



Règle du pivot (second cas).

Quand le processus s'arrête, chaque classe est un module. On peut relancer récursivement l'algorithme sur les modules non-triviaux, jusqu'à obtenir une permutation factorisante.

La complexité temporelle du calcul dépend de la règle de choix des pivots. En utilisant la *règle de Hopcroft* [Hop71], Michel Habib, Christophe Paul et Laurent Viennot parviennent à une complexité $O(n + m \log n)$, car un processus dichotomique assure que chaque sommet servira $\log n$ fois au plus de pivot. Grâce à une implémentation adéquate le temps mis pour appliquer la règle du centre ou du pivot est proportionnel à la longueur de la liste d'adjacence de ce sommet.

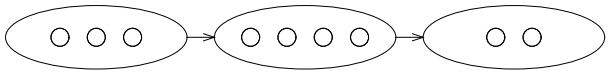
Ainsi l'algorithme de [HPV99] atteint presque une complexité linéaire, mais ce n'est pas le cas. Ce but serait atteint si chaque sommet ne pouvait servir qu'un nombre *constant* de fois de pivot. C'est ce à quoi nous parvenons dans l'algorithme décrit plus bas, qui utilise une classe d'ordres plus large que les partitions : les partitions ordonnées de chaînes.

6.2 Partitions ordonnées de chaînes

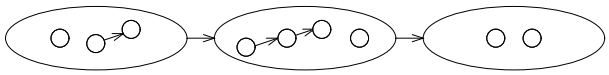
L'algorithme présenté ci-dessous utilise la notion suivante, un hybride entre les permutations factorisantes et les partitions ordonnées :

Définition 36
*Une **partition ordonnée de chaînes** est un ordre sur un ensemble V tel que V est partitionné en **classes** $C_1 \dots C_k$, et chaque classe C_i est à son tour partitionnée en **chaînes** $S_{i,1} \dots S_{i,h}$.*

$\forall x \in C_i \forall y \in C_j,\ x \prec_{\mathcal{O}} y$ si et seulement si $i < j$ ou x et y appartiennent à la même chaîne S (ce qui implique $i = j$) et $x \prec_S y$.

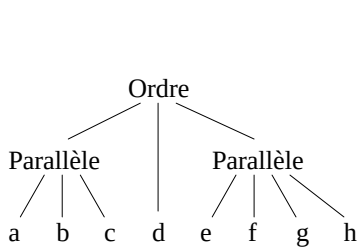


Partition ordonnée classique.

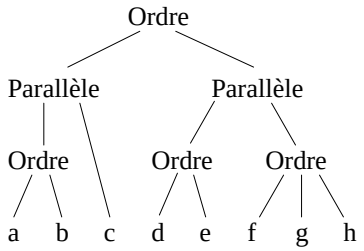


Partition ordonnée de chaînes : des comparaisons supplémentaires apparaissent.

Ainsi, dans une partition ordonnée de chaînes, il existe des relations d'ordre à l'intérieur d'une même classe : les chaînes. Ce sont des sous-ordres totaux. On peut aussi présenter les choses en termes de décomposition modulaire du graphe orienté correspondant. Une partition ordonnée $\mathcal{O}$ possède un arbre de décomposition $\mathcal{T}_{\mathcal{O}}$ dont la racine est un nœud *ordre*, dont les autres nœuds internes sont des *parallèle* (antichaînes) et dont la hauteur est d'au plus 2. Quant à une partition ordonnée de chaînes, sa racine est un nœud *ordre*, ses nœuds internes au premier niveau sous la racine sont tous des *parallèle*, et ses nœuds internes au deuxième niveau sont tous des *ordre*. Sa hauteur est d'au plus 3.



Partition ordonnée classique



Partition ordonnée de chaînes

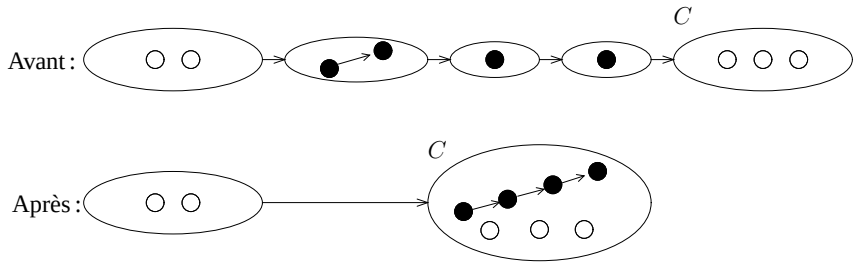
Une classe est **monocaténaire** si elle ne contient qu'une seule chaîne, et **multicaténaire** sinon. Une chaîne est **triviale** si elle ne contient qu'un seul sommet. Les partition ordonnées de chaînes sont une généralisation propre des permutations ordonnées (où toutes les chaînes sont triviales) et des permutations (où toutes les classes sont monocaténaires). La classe d'un sommet x est $\mathcal{C}(x)$, et sa chaîne $\mathcal{S}(x)$.

6.3 L'algorithme linéaire

Cet algorithme prend comme données un graphe non-orienté $G = (V, E)$ et fournit en sortie une permutation $\sigma : V \to [\![1, \dots n]\!]$ factorisante des modules de G. La section suivante présente rapidement les modifications à apporter à l'algorithme pour le linéariser. La section 6.4 décrit l'algorithme en détails et prouve sa correction. La section 6.5 décrit comment l'algorithme peut être implémenté séquentiellement pour tourner en $O(n + m)$ et prouve cette borne ; enfin la section 6.6 montre que l'algorithme peut tourner en $O(n)$ sur une machine parallèle à n processeurs.

Nous travaillons par *extension* de partitions ordonnées de chaînes : l'algorithme démarre avec une partition ordonnée de chaînes n'ayant qu'une seule classe et des chaînes triviales (ordre vide) et s'arrête quand toutes les classes sont monocaténaires (ordre total). Nous utilisons toujours la *règle du centre* et la *règle du pivot*. Elles agissent toujours sur les *classes* uniquement, sans pouvoir couper de chaînes : lorsqu'une classe est coupée, on partitionne ses chaînes entre les nouvelles classes. Chaque chaîne possède exactement un sommet **actif** , les autres sommets étant **inactifs**. Le sommet actif d'une chaîne S est son **représentant** noté $r(S)$. La section 6.5 présente une implémentation des partitions ordonnées de chaînes utilisable par notre algorithme. Les chaînes sont placées dans la classe où leur sommet actif les entraîne, en fonction de leur adjacence au centre (ou au pivot).

Nous rajoutons un troisième règle : la **concaténation**. Contrairement aux deux premières règles qui *étendent* l'ordre, celle-ci *détruit* des comparaisons. La concaténation consiste en le fait de transformer en une seule chaîne plusieurs classes monocaténaires consécutives (qui forment donc un ordre total). La chaîne S ainsi formée est ensuite ajoutée dans une classe C qui lui est adjacente. Au passage, les comparaisons entre S et C sont perdues [2].



Action de la règle de concaténation. Les sommets noirs sont fondus en une seule chaîne mise dans C.

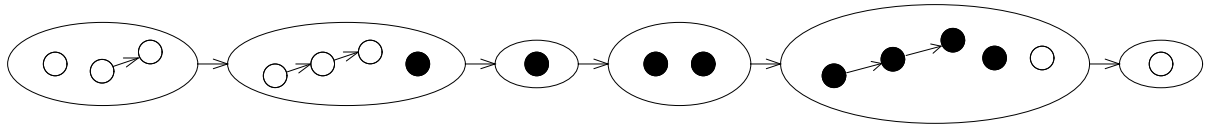
L'invariant principal maintenu par notre algorithme est :

Invariant 1
$\mathcal{O}$ est un ordre factorisant de $V(G)$.

Un invariant plus fort exprime le rôle particulier des chaînes :

Invariant 2
Aucun module fort de G ne chevauche de chaîne de $\mathcal{O}$.

2. Une partie sera retrouvée l'étape suivante, quand une règle du centre coupe cette classe en trois.



Position d'un module fort M (sommets noirs) au sein de $\mathcal{O}$, selon les invariants 1 et 2.

Afin de parvenir à une complexité linéaire, le voisinage d'un sommet donné doit être utilisé $O(1)$ fois. Cela est possible, si nous restreignons l'algorithme à l'utilisation d'*un seul* pivot par classe. Appelons **utilisée** une classe dont un sommet a déjà été utilisé pour la règle du pivot. On s'interdit de re-choisir un pivot au sein d'une classe utilisée : chaque sommet ne servira qu'une fois de pivot. L'idée d'utiliser un seul pivot par classe vient de l'algorithme de Habib & Paul [HP01] pour les cographes, produisant un algorithme linéaire. Voici l'algorithme que nous proposons :

Algorithme 5: Algorithme linéaire de calcul de permutations factorisantes

Données : un centre c et un facteur de travail $[\![i, j]\!]$
début
  Couper $[\![i, j]\!]$ selon la règle du centre appliquée à c
  répéter (jusqu'au retour de procédure)
    tant que *il existe une classe inutilisée dans* $[\![i, j]\!]$ **faire**
      Choisir une classe inutilisée $C \subset [\![i, j]\!]$ et un pivot $p \in C$
      Étendre $\mathcal{O}$ selon la règle du pivot appliquée à p
    fin
    si $[\![i, j]\!]$ *n'a plus de classe multicaténaire* **alors retourner**
    Choisir la classe multicaténaire C' du nouveau centre c_n
    Ajouter à C' une chaîne contenant c par la règle de concaténation
    Relancer récursivement avec le centre c_n et le facteur de travail C'
  fin
fin

Cette ébauche d'algorithme peut s'arrêter avant de parvenir à une permutation factorisante ; pire, les classes non-triviales (multicaténaires) quand l'algorithme termine ne sont pas des modules, contrairement au cas de [HPV99]. On décoince les choses en choisissant un nouveau centre : l'algorithme 5 est *récursif*. Mais on ne peut pas se permettre d'utiliser la seule classe du centre comme champ de l'appel récursif (puisqu'elle n'est pas un module) : c'est là qu'intervient la troisième règle (concaténation) qui va lui rajouter des sommets afin de rendre possible un appel récursif. Présentons les deux problèmes que causent la récursivation : le choix du centre, et l'existence des mauvais pivots.

Choix d'un nouveau centre

La preuve de [HPV99] fait apparaître le rôle crucial du centre. La raison pour laquelle la règle de pivot distingue deux cas selon la position relative du centre apparaîtra dans la preuve, §6.4.3. Cette preuve est basée sur l'invariant suivant :

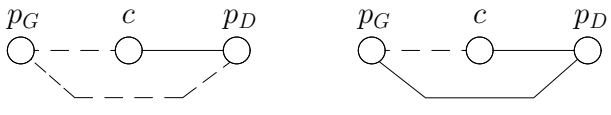
Invariant 3
Soit M un module fort et c le centre. Au moins une des propositions suivantes est vraie :

1. $c \in M$
2. *toutes les classes intersectant M sont monocaténaires*
3. M *est inclus dans une unique classe C*

Cet invariant permet de classer les modules en trois groupes : ceux en cours de traitement, correspondant au dessin précédent et dont l'algorithme est en train de calculer une permutation ; ceux déjà calculés, qui ne sont plus que dans des classes monocaténaires puis, après concaténation, dans une seule chaîne ; et enfin ceux non encore traités qui sont inclus dans une classe ne contenant pas le centre.

Lors d'un appel récursif, le nouveau centre c_n doit vérifier cet invariant (que l'ancien centre c vérifiait). Si tous les modules forts contenant c mais pas c_n étaient inclus dans la classe $\mathcal{C}(c)$ l'invariant resterait vrai. C'est justement ce que nous allons essayer de faire. Plus c et c_n sont proches (au sens de l'ordre $\mathcal{O}$), plus les modules contenant c mais non c_n seront petits. c_n sera le pivot de l'une des deux classes multicaténaires les plus proches de c, soit celle de gauche, soit celle de droite.

Donc, avant de lancer un appel récursif, il faut décider par une certaine règle de choix quel sera le nouveau pivot. Soit p_G le pivot de la classe multicaténaire la plus proche de c située à gauche de c, et p_D celui de la classe située à droite de c. On suppose (cela sera établi plus tard, quand on montrera que p_G et p_D sont de *bons sommets*) que p_G n'est pas voisin de c et que l'autre sommet l'est. Comme le montre le dessin suivant, un et un seul des deux pivots distingue toujours la paire formée par le centre et l'autre pivot. Si on prend comme nouveau centre celui qui ne distingue pas, on court le risque qu'il existe un module fort contenant les deux autres sommets en violation de l'invariant 3. Ce risque est éliminé si on prend l'autre sommet (qui distingue) : c'est donc lui le nouveau centre. Donc un simple test d'adjacence entre p_G et p_D permet de déterminer le nouveau centre. Le lemme 43 page 174 et sa preuve détaillent davantage ce point.



Premier cas : on choisit p_D Second cas : on choisit p_G

La règle de choix du nouveau centre : si $(p_G, p_D) \notin E$ alors p_D est choisi ; sinon c'est p_G le nouveau centre.

L'ancien centre c ainsi que tous les sommets entre c et c_n et d'autres sommets (c'est le rôle de la fonction `TrouverChaine` de dire lesquels, et cela est le point technique de l'algorithme) sont alors rajoutées à la classe du nouveau centre c_n, par la règle de concaténation. Le Lemme 43 (§6.4.4) assure qu'avec cette règle du choix du centre puis cette concaténation tous les modules forts contenant c mais pas c_n sont maintenant dans $\mathcal{C}(c_n)$, et qu'ainsi les invariants de l'algorithme resteront valides.

On remarquera au passage que $\mathcal{C}(c_n)$ contient après cette concaténation *deux* pivots utilisés, c et c_n. Heureusement la règle du centre, appliquée immédiatement après, les séparera de nouveau. On a donc l'invariant suivant, fondamental pour la preuve de complexité :

Invariant 4
Une classe contient au plus deux pivots utilisés. Si elle en contient exactement deux, alors l'un d'eux est le centre.

Ce deuxième cas est bien sûr vérifié par au plus une classe.

Le problème des mauvais pivots et le facteur de travail

La preuve que la règle du pivot préserve le caractère factorisant de l'ordre (établie en §6.4.3) dépend d'un second point important : le fait qu'un pivot est voisin du centre si et seulement si il est à sa droite. C'est précisément le rôle de la règle du centre que de créer une partition ordonnée de chaînes qui vérifie cette propriété. Mais lors d'un changement de centre, elle n'a plus aucune raison d'être vraie ! Un sommet x est dit **mauvais** si

- soit $x \prec_{\mathcal{O}} c$ et $x \in \Gamma(c)$;
- soit $x \succ_{\mathcal{O}} c$ et $x \in \overline{\Gamma}(c)$.

Une chaîne est mauvaise si son représentant (c'est-à-dire son unique sommet actif) est mauvais, et une classe est mauvaise si elle ne contient **que** des mauvaises chaînes.

Le **facteur de travail** est l'ensemble des chaînes qui forment $\mathcal{C}(c)$ au moment où l'appel récursif commence. Ce sont exactement eux qui sont concernés par la règle du centre, et placés à gauche ou à droite de c selon leur voisinage : ils ne peuvent être mauvais. On note $[\![i, j]\!]$ le facteur de travail (les deux nombres i et j sont tels que dans toute extension linéaire σ de $\mathcal{O}$, un sommet x est dans l'intervalle de travail ssi $i \leqslant \sigma(x) \leqslant j$).

Les invariants et remarques suivants précisent le rôle du facteur de travail et l'innocuité des mauvaises classes.

Invariant 5
Si un sommet actif x est mauvais alors $\mathcal{C}(x)$ est une mauvaise classe. Aucun module fort ne chevauche alors $\mathcal{C}(x)$, et $\mathcal{C}(x) \cap [\![i, j]\!] = \emptyset$.

Invariant 6
Soit $[\![i, j]\!]$ le facteur de travail et c le centre. Aucune classe ne chevauche $[\![i, j]\!]$. Si un module fort chevauche $[\![i, j]\!]$ alors il contient c.

Remarque 1
Les facteurs de travail des différents appels récursifs sont imbriqués les uns dans les autres.

Remarque 2
Lorsqu'un appel récursif termine, le facteur de travail qui lui est associé ne contient plus que des classes monocaténaires utilisées.

On dit qu'on a *résolu* le facteur : un ordre total lui est donné. Quand l'appel initial, donc le facteur est V tout entier, termine, on possède donc une permutation. D'après l'invariant 1 elle est factorisante. Cela constitue la preuve de validité de l'algorithme (théorème 36).

Grandes lignes de l'algorithme

La section suivante décrit chaque étape de l'algorithme linéaire en détails, et donne la preuve que les opérations effectuées préservent les invariants. En §6.5 est proposée une implémentation et du code plus « réalistes », prouvant qu'une complexité séquentielle $O(n + m)$ peut être atteinte ; §6.6 parallélise l'algorithme.

6.4 Description détaillée et preuve de l'algorithme

L'algorithme est découpé en quatre procédures et une fonction. La procédure principale, récursive, est `POC-ext`. Elle ne fait aucune action. Les procédures `Ext-Centre`, `Ext-Pivot` et `Concat` implémentent respectivement les règles du centre, du pivot et de concaténation. Enfin la fonction `TrouverChaine` définit l'extension de la chaîne qui sera utilisée par la règle de concaténation.

Les sections suivantes les décrivent en détails et fournissent les éléments de la preuve par invariants de l'algorithme.

Invariant 1
$\mathcal{O}$ est un ordre factorisant de $V(G)$.

Invariant 2
Aucun module fort de G ne chevauche de chaîne de $\mathcal{O}$.

Invariant 3
Soit M un module fort et c le centre. Au moins une des propositions suivantes est vraie :

1. *$c \in M$*
2. *toutes les classes intersectant M sont monocaténaires*
3. *M est inclus dans une unique classe C*

Invariant 4
Une classe contient au plus deux pivots utilisés. Si elle en contient exactement deux, alors l'un d'eux est le centre.

Invariant 5
Si un sommet actif x est mauvais alors $\mathcal{C}(x)$ est une mauvaise classe. Aucun module fort ne chevauche alors $\mathcal{C}(x)$, et $\mathcal{C}(x) \cap [\![i, j]\!] = \emptyset$.

Invariant 6
Soit $[\![i, j]\!]$ le facteur de travail et c le centre. Aucune classe ne chevauche $[\![i, j]\!]$. Si un module fort chevauche $[\![i, j]\!]$ alors il contient c.

Récapitulatif des invariants

6.4.1 La procédure principale, POC-ext

Description

L'algorithme 6 donne le pseudo-code détaillé de la procédure principale de l'algorithme. Ses paramètres sont la partition ordonnée de chaînes à étendre, les bornes i et j du facteur de travail, ainsi que le centre c et le centre c_p de l'appel récursif parent (qui est utilisé par Ext-Centre). L'appel initial est POC-ext(OrdreVide$(V), 1, n, c_0, c_0)$; le premier centre c_0 étant arbitrairement choisi. $\mathcal{O}$ est la partition ordonnée de chaînes courante. Elle est initialisée à $\{V\}$.

Preuve de validité

Théorème 36
L'algorithme 6 calcule une permutation factorisante d'un graphe G.

Démonstration: Afin de prouver ce théorème, il nous faut montrer que les invariants 1 à 6 sont vérifiés tout au long de l'algorithme. Puisque le facteur de travail de l'appel initial est V, d'après la remarque 2 (très facile à prouver au vu de la condition d'arrêt de l'algorithme), ce facteur

Algorithme 6: `POC-ext`$(\mathcal{O}, i, j, c, c_p)$: calcul d'une permutation factorisante

début
  `Ext-Centre`$(\mathcal{O}, c, c_p)$
  répéter (jusqu'au retour de procédure)
    tant que $[\![i, j]\!]$ *contient des classes inutilisées* **faire**
      Choisir une classe inutilisée $C \subset [\![i, j]\!]$ et un pivot actif $p \in C$
      $Pivot[C] \leftarrow p$
      `Ext-Pivot`$(\mathcal{O}, p, c)$
    fin
    si $[\![i, j]\!]$ *n'a plus de classe multicaténaire* **alors retourner**
    Soit C_G (resp. C_D) la classe multicaténaire de $[\![i, j]\!]$ la plus proche de c située à sa gauche (resp. droite) si une telle classe existe
    si C_D ***n'existe pas ou si*** $(Pivot[C_G], Pivot[C_D]) \in E$ **alors**
      $C' \leftarrow C_G$
    sinon
      $C' \leftarrow C_D$
    fin
    $a \leftarrow$ `TrouverChaine`$(\mathcal{O}, Pivot[C'], c, i, j)$
    `Concat`$(\mathcal{O}, C', a, c)$
    $c_n \leftarrow Pivot[C']$
    $i_n \leftarrow Debut[C']$
    $j_n \leftarrow Fin[C']$
    `Ext-Pivot`$(\mathcal{O}, c_n, c)$
    `POC-ext`$(\mathcal{O}, i_n, j_n, c_n, c)$
  fin
fin

est une permutation quand l'algorithme s'arrête. L'invariant 1 assure que cette permutation est factorisante.

La terminaison de l'algorithme n'est pas triviale, puisque les classes peuvent être coupées mais aussi fusionnées ; elle est établie par le théorème 37 de complexité temporelle.

On remarquera que, outre le graphe et ses modules, les invariants mentionnent trois variables : la partition ordonnée de chaînes $\mathcal{O}$, le centre c et l'intervalle de travail $[\![i, j]\!]$. $\mathcal{O}$ n'est modifié que par les trois procédures `Ext-Centre`, `Ext-Pivot` et `Concat`. La preuve que chacune maintient les invariants relatifs à $\mathcal{O}$ est donnée en §6.4.2, 6.4.3 et 6.4.5, respectivement. c et $[\![i, j]\!]$ sont définis par `POC-ext` avant un appel récursif. La seule modification ultérieure d'une de ces variable est celle faite par `Ext-Centre` dans un cas bien particulier. Le lemme 39 établi que si les invariants sont vérifiés avant un appel récursif, ils le restent pour les nouvelles valeurs du

centre et de l'intervalle de travail. Le lemme 40 donne la même preuve au retour d'un appel récursif. Et le lemme 41 concerne la modification faite par `Ext-Centre`.

Enfin, puisque l'ordre vide, le centre initial c_0 et l'intervalle V vérifient trivialement les invariants avant que l'algorithme ne démarre, la preuve est valide.

□

Validité des invariants lors d'un appel récursif

Lemme 39 *Si les invariants sont vérifiés avant de faire un appel récursif à* `POC-ext` *alors ils le sont lorsque cet appel commence.*

Démonstration: La partition ordonnée de chaînes $\mathcal{O}$ n'est pas modifiée par le fait qu'un appel récursif commence : seuls changent le centre et le facteur de travail. Les invariants 1 et 2 ne les citant pas, ils restent vrais. Montrons la validité des quatre derniers invariants. i, j et c représentent les valeurs des variables lors de l'appel présent de `POC-ext` et i_n, j_n et c_n les valeurs prises par l'appel récursif qui va être fait.

Invariant 3

Soit M un module fort. Si $c \notin M$, M vérifie déjà le point 2 ou le point 3 de l'invariant. Sinon, d'après le lemme 43 page 174, tout module contenant c mais non c_n est inclus dans la classe $\mathcal{C}(c)$, et vérifie donc le point 3. Tout autre module qui vérifiait avant le point 1 (en contenant c) contient aussi c_n et donc vérifiera toujours le point 1 avec le nouveau centre.

Invariant 4

Le nouveau centre c_n est pris dans la classe $\mathcal{C}(c)$ (après concaténation) donc la seule classe à avoir deux pivots utilisés contiendra le nouveau centre.

Invariant 5

Soit x un sommet actif qui est mauvais vis-à-vis de c_n. Puisque $\mathcal{C}(x) \neq \mathcal{C}(c_n)$, $x \notin [\![i_n, j_n]\!]$. Juste avant l'appel récursif, la règle du pivot est utilisée (ligne `Ext-Pivot`$(\mathcal{O}, c_n, c)$) donc $\mathcal{C}(x)$ ne chevauche pas $\Gamma(c_n)$. Tous ses sommets actifs sont mauvais : c'est une mauvaise classe.

Supposons l'existence d'un module fort M chevauchant $\mathcal{C}(x)$. D'après l'invariant 3, $c \in M$. D'après le lemme 43, puisque $\mathcal{C}(x) \cap [\![i_n, j_n]\!] = \emptyset$, $c_n \in M$. Supposons sans perte de généralité que x soit un voisin de c_n, situé à sa gauche. Comme $\mathcal{C}(x) \cap M \neq \emptyset$, il existe un sommet $z \notin M$, actif d'après l'invariant 2, tel que $z \in \mathcal{C}(x)$. z est aussi à gauche de c et, comme M est adjacent à z, $(z, c) \in E$. z est donc mauvais vis-à-vis du centre c. La classe $\mathcal{C}(x)$ viole donc l'invariant 5 pris avec le centre c, contradiction.

Invariant 6

Comme $[\![i_n, j_n]\!] = \mathcal{C}(c_n)$ aucune classe ne peut chevaucher le nouveau facteur de travail. Soit

M un module fort qui le chevauche. La classe $\mathcal{C}(c_n)$ étant multicaténaire, d'après l'invariant 3 $c \in M$. Le lemme 43 garanti que $c_n \in M$ sans quoi M serait inclus dans $[\![i_n, j_n]\!]$.

□

Validité des invariants au retour d'un appel récursif

Lemme 40 *Si les invariants sont vrais lorsqu'un appel récursif de* `POC-ext` *s'achève alors ils restent vrais au retour dans l'appel parent.*

Démonstration: Là encore les invariants 1 et 2 ne sont pas concernés. On pose que i_n, j_n et c_n sont les valeurs prises par les variables dans l'appel récursif qui se termine, et i, j et c celles de l'appel parent.

Invariant 3
Si $c \in M$ alors cet invariant est vrai. Tout module qui vérifiait déjà le point 2 avant l'appel récursif le vérifie toujours ; et d'après la remarque 2 un module inclus dans $[\![i, j]\!]$ le vérifie aussi. reste le cas des modules qui vérifiaient le point 3 avant l'appel. Ils étaient hors de $[\![i, j]\!]$. Les règles du centre et de concaténation n'ont pu les affecter. Quand aux extensions faites par la règle du pivot, puisque le pivot est pris dans $[\![i, j]\!]$ il ne contient ces modules et donc elles n'ont pu les séparer en plusieurs classes : ils vérifient toujours le point 3.

Invariant 4
La classe du centre est monocaténaire : toute classe ne peut contenir qu'un seul pivot utilisé et cet invariant restera vrai.

Invariant 5
Comme une classe mauvaise vis-à-vis de c est hors de $[\![i, j]\!]$, elle est hors de $[\![i_n, j_n]\!]$. La règle du centre ou les concaténations faites par les appels récursifs n'ont pu l'affecter, quant aux extensions faites par la règle du pivot, elles n'ont pu créer de chevauchement de module.

Invariant 6
Comme i, j et c reprennent leurs valeurs précédentes cet invariant est vrai, quelles que soit les actions faites par l'appel récursif.

□

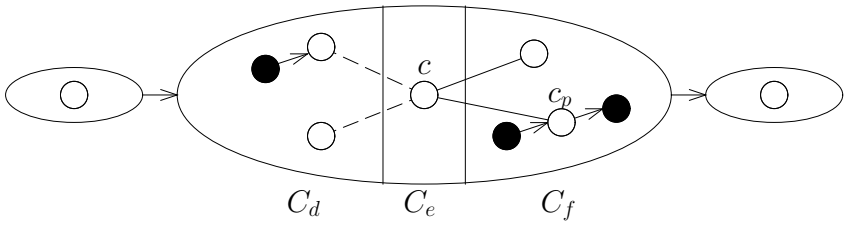
6.4.2 La règle du centre : procédure `Ext-Centre`

Description

La règle du centre est programmée en la procédure `Ext-Centre`$(\mathcal{O}, c, c_p)$. Trois nouvelles classes sont créées à partir d'une, C_c, contenant c et c_p (le centre et le centre précédent). D'après

l'invariant 4, il s'agit de la seule classe ayant deux pivots utilisés (qui vont être séparés). Les autres classes ne sont pas affectées.

La procédure déplace des chaînes (et non des sommets), en fonction de l'adjacence de leur sommet actif vis-à-vis du centre. Trois nouvelles classes sont crées : C_d, C_e et C_f.



Action de `Ext-Centre` *sur* C_c*. Les sommets actifs sont blancs.*

Afin de garantir la terminaison de l'algorithme, nous utilisons la version modifiée de cette règle, comme dans [HP01] : il est impératif que la classe soit coupée en *trois*. Donc, si $C_c \cap \overline{\Gamma}(c)$ ou $C_c \cap \Gamma(c)$ était vide, $\mathcal{S}(c_p)$ est placée dans cette classe. Remarquons que C_c contient au moins trois chaînes, car elle est le produit de la concaténation d'une classe multicaténaire avec $\mathcal{S}(c_p)$. La trissection est donc toujours possible. Toutefois, dans ce cas, c_p serait un mauvais sommet à l'intérieur du facteur de travail. Pour éviter cela (et bien que c ne soit pas strictement nécessaire, car c_p ne servira plus jamais de pivot) cette classe est exclue du facteur de travail, dont les bords sont modifiés en conséquence.

Algorithme 7: Procédure `Ext-Centre`$(\mathcal{O}, c, c_p)$

début
  $C_d \leftarrow \emptyset; C_e \leftarrow \emptyset; C_f \leftarrow \emptyset$
  pour chaque *chaîne* $S \subset C_c$ **faire**
    si $r(S) \in \Gamma(c)$ **alors** $C_f \leftarrow C_f \uplus S$
    sinon si $r(S) = c$ **alors** $C_e \leftarrow C_e \uplus S$
    sinon $C_d \leftarrow C_d \uplus S$
  fin
  Remplacer C_c par $C_d \oplus C_e \oplus C_f$ dans $\mathcal{O}$
  si C_d *est vide* **alors**
    $\mathcal{C}(\mathcal{S}(c_p)) \leftarrow C_d$
    augmenter i de sorte que $[\![i, j]\!] = C_e \oplus C_f$
  sinon si C_f *est vide* **alors**
    $\mathcal{C}(\mathcal{S}(c_p)) \leftarrow C_f$
    diminuer j de sorte que $[\![i, j]\!] = C_d \oplus C_e$
  fin
fin

Preuve de validité

Lemme 41 *Si les invariants sont vérifiés avant un appel à la procédure* `Ext-Centre` *alors ils le sont après cet appel.*

Démonstration:

Invariant 1

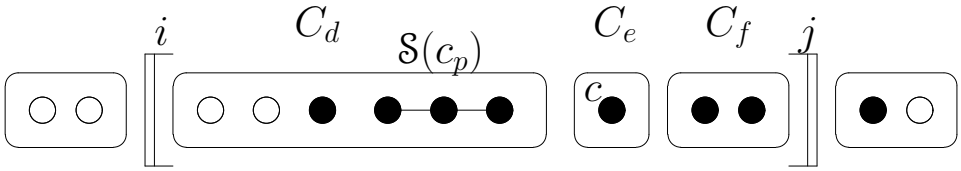
Soit M un module fort. Aucun module fort ne croise M avant l'appel de `Ext-Centre`. Supposons qu'après cet appel un module fort N croise M. Cela veut dire que se sont les comparaisons rajoutées par la trissection de C_c qui ont créé ce croisement. Donc M et N intersectent tous deux C_c.

C_c est une classe multicaténaire : d'après l'invariant 3 tout module qui la chevauche contient le centre c. Or M et N sont d'intersection disjointe : un seul, au plus, peut donc chevaucher C.

Si les deux sont inclus dans C, et puisqu'ils se croisent, au moins l'un des deux intersecte C_d et C_f (car un seul peut intersecter C_e, monocaténaire) sans contenir c. c distingue donc un module : contradiction.

Supposons donc que M chevauche C_c, et donc $c \in M$, tandis que $N \subset C_c$. Ce n'est pas le premier appel (sans quoi $C_c = V$) donc c_p est le centre précédent. D'après le lemme 43 tout module contenant c_p mais non c est inclus dans C_c, ce que ne fait pas M : $c_p \in M$.

Nous devons maintenant raisonner avec c_p. Il y a deux cas : $(c_p, c) \notin E$ ou $(c_p, c) \in E$. Nous n'étudierons que le premier ; l'autre étant semblable quitte à complémenter G et à retourner $\mathcal{O}$. D'après l'invariant 2, $\mathcal{S}(c_p) \subset M$. La classe C_c est le résultat de la concaténation d'une classe antérieure, C_G dans l'algorithme 6, avec une chaîne dont c_p est le représentant. D'après l'invariant 5 cette classe, située dans le facteur de travail de c_p, était bonne vis-à-vis de c_p, donc non-adjacente à c_p. Prenons $x \in C_f$. x est voisin de c mais non de c_p, donc distingue M. Donc $C_f \subset M$. On a aussi $C_e \subset M$. Donc N ne peut appartenir qu'à C_d. Mais dans ce cas, il ne peut croiser M, car $N \preccurlyeq_{\mathcal{O}} M$.



Position de M (en noir) au sein de $\mathcal{O}$.

Si C_f était vide et que $\mathcal{S}(c_p)$ y a été placé, $\mathcal{C}(c_p) \subset M$ et on a exactement la même configuration que le cas précédent.

Invariant 2

Comme `Ext-Centre` déplace les chaînes sans les couper, cet invariant est préservé.

Invariant 3

Soit M un module fort. Si $c \in M$ cet invariant reste vrai (point 1). Sinon seul un module intersectant C_c peut être affecté par la règle du centre. Un tel module M ne peut vérifier que le point 3. Puisque $c \notin M$, M sera placé soit dans C_d soit dans C_f selon son adjacence à c.

Il y a la cas particulier où `Ext-Centre` met c_p du « mauvais coté » de c. Supposons par exemple que c soit non-voisin de tous les sommets actifs de C_c. C'est aussi le cas de c_p, puisqu'il est l'ancien centre. Si un module M contient uniquement des sommets actifs de C_c, que $c \notin M$ et $c_p \in M$, alors M' avec les mêmes sommets mais en ajoutant c_p et en retranchant c est aussi un module : M n'est pas fort. Cela termine la preuve.

Invariant 4

Après appel à `Ext-Centre`, c et c_p sont placé dans deux classes différentes. c est l'unique pivot actif utilisé de C_e, c_p celui de C_d ou C_f selon le cas, et la troisième classe est inutilisée.

Invariant 5

Soit x un mauvais sommet actif. Si x n'est pas dans la classe de c avant l'appel à `Ext-Centre`, alors la classe de x n'est pas affecté par cette procédure (qui ne touche que C_c) et l'invariant restera vrai. Sinon on a $x \in C_c$. Si $x = c$ l'invariant est vrai. Si $x = c_p$ est placé du « mauvais côté » de c car sinon sa classe serait vide, l'intervalle $[\![i, j]\!]$ est modifié de sorte que c_p en soit exclu, ce qui préserve cet invariant. $\mathcal{C}(c_p)$ est de plus monocaténaire, donc mauvaise et non-chevauchable par un module, d'après l'invariant 2. Et sinon, x sera placé dans C_g ou C_d selon qu'il est adjacent à c ou non. Il se retrouve du « bon côté » par-rapport au centre : ce n'est pas un mauvais sommet.

Invariant 6

Si C_d et C_f sont non vides après la trissection de C_c, le facteur de travail n'est pas modifié et l'invariant reste vrai. Sinon, $[\![i, j]\!]$ contient exactement deux classes, sans chevauchement. Supposons qu'un module fort M chevauche maintenant ce facteur sans le chevaucher avant. D'après l'invariant 1 M intersecte $\mathcal{C}(c_p)$ et C_e, donc $c \in M$.

□

6.4.3 La règle du pivot : procédure `Ext-Pivot`

Description

La procédure `Ext-Pivot` (algorithme 9) étend $\mathcal{O}$ en *coupant* chaque classe X (sauf celle contenant le pivot p) selon le voisinage de p. Les chaînes sont déplacées selon l'adjacence de leur représentant avec le pivot.

Si X chevauche $\Gamma(p)$, les chaînes de X dont le représentant est non-voisin de p sont mises dans une classe X_a, et les autres chaînes dans une classe X_b. On utilise le test du lemme 38 pour

placer X_a relativement à X_b dans $\mathcal{O}$. Si on crée deux nouvelles classes à partir d'une classe non-utilisée, ces deux classes sont non-utilisées. Sinon, la classe contenant le pivot de X est utilisée et son pivot prend la valeur du pivot de X, tandis que l'autre classe est inutilisée. Ainsi l'application de la règle du pivot utilise une classe (celle de p) mais peut créer des classes non-utilisées.

Algorithme 8: Procédure `Ext-Pivot`$(\mathcal{O}, p, c)$

début
  pour chaque *classe* X *telle que* $\mathcal{C}(p) \neq X$ **faire**
    pour chaque *chaîne* $S \subset X$ **faire**
      si $r(S) \notin \Gamma(p)$ **alors** $\mathcal{C}(S) \leftarrow X_a$
      sinon $\mathcal{C}(S) \leftarrow X_b$
    fin
    si X_a *et* X_b *sont non-vides* **alors**
      si $c \prec_{\mathcal{O}} X \prec_{\mathcal{O}} p$ *ou* $p \prec_{\mathcal{O}} X \prec_{\mathcal{O}} x$ **alors**
        Remplacer X par $X_b \oplus X_a$ au sein de $\mathcal{O}$
      sinon
        Remplacer X par $X_a \oplus X_b$ au sein de $\mathcal{O}$
      fin
    fin
  fin
fin

Preuve de validité

Lemme 42 *Si les invariants sont vérifiés avant appel à* `Ext-Pivot` *alors ils le restent après cet appel.*

Démonstration:

Invariant 1

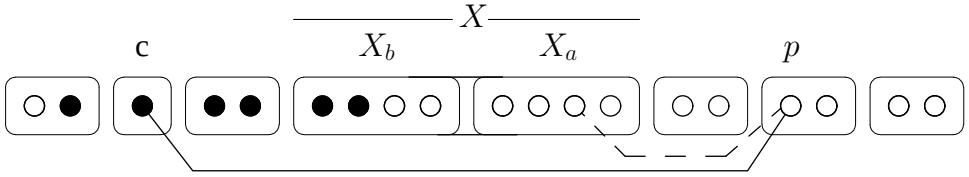
Supposons cet invariant vrai avant d'appeler `Ext-Pivot`, mais faux après l'appel. Les classes sont coupées une par une. Soit X la classe telle que l'invariant devienne faux après qu'elle soit coupée. Si $\mathcal{O}$ n'est pas un ordre factorisant, alors il existe deux modules M et N tels que $M \perp_{\mathcal{O}} N$. Puisque ce sont les comparaisons créées par la coupure de X qui ont créé ce croisement, M et N intersectent tous deux X.

Si $M \subset X$ alors $p \notin M$. Après coupure $M \subset X_a$ ou $M \subset X_b$, selon son adjacence à p. Et sinon, M chevauche X. D'après l'invariant 3, $c \in M$. Dans ce cas N ne peut pas chevaucher aussi X : cela entraînerait $c \in N$, or, les deux modules sont d'intersection vide. Si M et N sont

tous deux inclus dans une seule classe, ils ne peuvent se croiser. Dans la suite, on va supposer que M chevauche X, tandis que N est inclus dans X_a ou X_b.

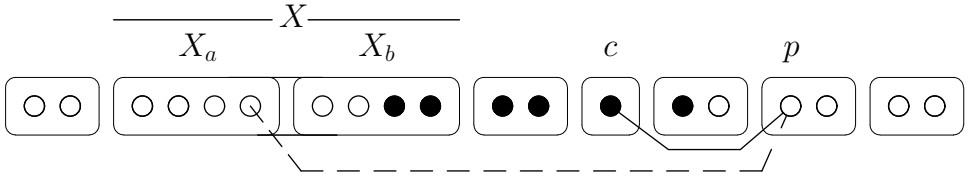
On a donc $c \in M$. On se restreint au cas $c \prec_{\mathcal{O}} p$. Comme p n'est pas mauvais (invariant 5) $(p, c) \notin E$. Le cas où p est non-voisin et à gauche de c se prouve de la même façon. Il faut par contre distinguer deux cas selon que $c \prec_{\mathcal{O}} X$ ou $c \succ_{\mathcal{O}} X$.

1. Si $p \notin M$
   - Si $c \prec_{\mathcal{O}} X$, d'après l'invariant 1, X se trouve placée *entre* c et p. Donc M est adjacent à p. Donc $M \cap X_a = \emptyset$. La procédure place X_b à gauche dans ce cas. Qu'on prenne $N \subset X_a$ ou $N \subset X_b$, N ne peut croiser M : l'invariant est préservé.



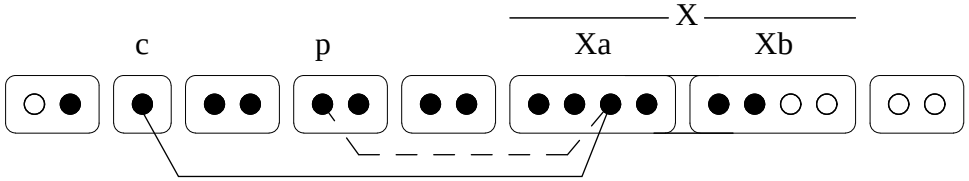
Cas où X est à droite de c et $p \notin M$. Les sommets de M sont en noir.

   - Si $p \notin M$ et $X \prec_{\mathcal{O}} c$. M est adjacent à p : $M \cap X_a = \emptyset$. Or X n'est pas placée entre c et p, donc la procédure place X_b à droite. Qu'on prenne $N \subset X_a$ ou $N \subset X_b$, N ne peut croiser M : l'invariant est préservé.



Cas où X est à gauche de c. Les sommets de M sont en noir.

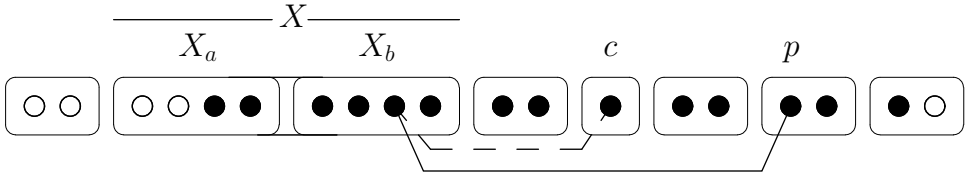
2. Si $p \in M$. Si il existe dans X un *mauvais sommet* adjacent à c, alors d'après l'invariant 5 aucun module fort ne peut chevaucher X, ce que viole M. X est donc une *bonne classe*.
   - Si $c \prec_{\mathcal{O}} X$, X est adjacente à c. D'après l'invariant 1, X se trouve placée *à droite* de p. $\forall x \in X_a$, on a $p \notin \Gamma(x) \ni c$. M chevauche le voisinage de x et M est un module, donc $x \in M$. Donc $X_a \subset M$. N est donc inclus dans X_b, mais ne peut croiser M, car X_b est à droite : l'invariant est préservé.



Cas où X est à droite de c et $p \in M$. Les sommets de M sont en noir.

   - Si $X \prec_{\mathcal{O}} c$, X est non-adjacente à c. Elle est de plus à gauche de p. Un sommet de X_b est non-adjacent à c et adjacent à p : il distingue deux sommets de M. Donc $X_b \subset M$.

Dans ce cas X_b est placé à droite. N est donc inclus dans X_a, mais ne peut croiser M : l'invariant est préservé.



Cas où X est à gauche de c. Les sommets de M sont en noir.

Invariant 2
Comme `Ext-Pivot` déplace les chaînes sans les couper, cet invariant est préservé.

invariant 3 Soit M un module fort avant extension.

1. Si $c \in M$ l'invariant reste vrai après extension.
2. Si toutes les classes intersectant M sont monocaténaires, alors tel est encore le cas après extension, car seules les classes multicaténaires peuvent être coupées.
3. Si M est inclus dans une classe C, alors
   - si M est inclus dans une seule chaîne avant extension alors c'est aussi le cas après extension ;
   - si $p \in C$ cette classe n'est pas affectée par l'extension ;
   - sinon : M n'est pas inclus dans une seule chaîne donc, d'après l'invariant 2, M est une union de chaînes. Et $p \notin M$: si C est coupée par p alors les sommets actifs de M se retrouvent dans X_a ou X_b et leurs chaînes aussi : M reste inclus dans une seule classe.

Invariant 4
p est pris uniquement dans une classe inutilisée, qui contient ensuite exactement un pivot actif utilisé. Cet invariant reste vrai.

Invariant 5
Soit x un mauvais sommet actif. Avant extension, x est hors de $[\![i, j]\!]$ et la mauvaise classe X qui le contient n'est chevauchée par aucun module. Soit M un module fort. Si $X \subset M$ ou si $M \cap C = \emptyset$ alors M ne peut chevaucher X_a ni X_b. Et si $M \subset X$ alors, selon que M est adjacent ou non à p, M se retrouve inclus dans X_a ou dans X_b et ne chevauche aucune classe.

Invariant 6
Une extension ne modifie pas les valeurs de i et j, cet invariant reste donc inchangé.

□

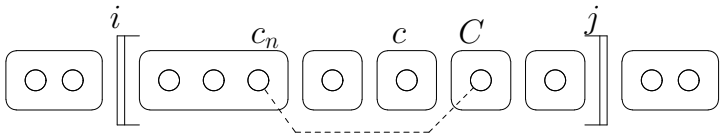
6.4.4 La fonction TrouverChaine

Description

Cette fonction précise l'extension de la chaîne à concaténer. Comme on l'a vu en §6.2, cette chaîne contient le centre c et au moins tous les sommets situés entre le centre c et le nouveau centre c_n. En fait il est parfois nécessaire de lui ajouter un peu *plus* de sommets, afin que le lemme 43 soit vérifié. La fonction TrouverChaine$(\mathcal{O}, c_n, c, i, j)$ précise où s'arrêter dans la concaténation.

Soit C_c la classe monocaténaire dont le centre c est le pivot. Supposons que le nouveau centre c_n soit *à gauche* de c. On définit la propriété (P) que possède une classe qui

- est monocaténaire, et
- est à droite de C_c ou lui est égale, et
- est situé dans $[\![i, j]\!]$, et
- son pivot p (son unique sommet actif) est non-voisin de c_n



La classe C vérifie (P).

TrouverChaine$(\mathcal{O}, c_n, c, i, j)$ est le pivot de l'unique classe D qui vérifie (P), telle que toutes les classes entre C_c et D vérifient (P), mais telle que la classe à droite de D viole (P) ou n'existe pas. On note que si c_n est à gauche de c alors $(c_n, c) \notin E$ (invariant 5), donc C_c vérifie (P). TrouverChaine$(\mathcal{O}, c_n, c, i, j)$ est toujours défini car il est au moins égal à c.

Dans le cas où c_n est *à droite* de c, on définit une propriété (P') que possède une classe qui

- est monocaténaire, et
- est à **gauche** de C_c ou lui est égale, et
- est situé dans $[\![i, j]\!]$, et
- son pivot p (son unique sommet actif) est **voisin** de c_n

TrouverChaine$(\mathcal{O}, c_n, c, i, j)$ est alors le pivot de l'unique classe D qui vérifie (P'), telle que toutes les classes entre C_c et D vérifient (P'), mais telle que la classe à **gauche** de D viole (P') ou n'existe pas. Là encore TrouverChaine$(\mathcal{O}, c_n, c, i, j)$ est toujours défini, car c_n est à droite de c alors $(c_n, c) \in E$ (invariant 5), donc C_c vérifie (P').

Lemme 43 *Considérons l'état des variables juste avant un appel récursif de* POC-ext. *Soit M un module fort tel que $c \in M$ et $c_n \notin M$. $M \subset [\![i_n, j_n]\!]$*

Démonstration: Soit M_1 le plus grand module fort tel que $c \in M_1$ et $c_n \notin M_1$. Supposons que $(c_n, c) \notin E$; le nouveau centre est à gauche de l'ancien. La preuve serait identique pour l'autre cas. c_n est dans la classe C_G, et c est à sa droite. Le facteur $[\![i_n, j_n]\!]$ contient avant l'appel Concat$(\mathcal{O}, C_G, a, c)$ la classe C_G et toute une série de classes monocaténaires, incluant c et

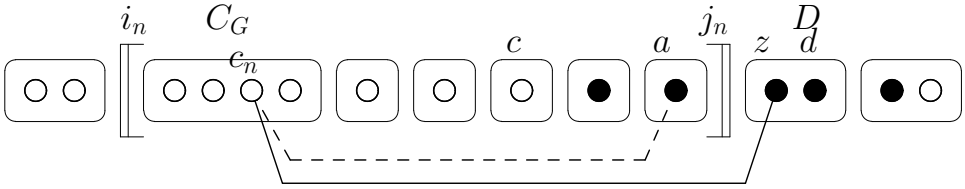
allant jusqu'à la classe dont le seul sommet actif est $a =$ `TrouverChaine`$(\mathcal{O}, c_n, c, i, j)$. Après l'appel à `Concat` ce facteur ne contient plus qu'une seule classe, que nous appellerons C'. Les sommets présents sont exactement les mêmes, donc cette preuve vaut avant et après concaténation. Nous utilisons les classes *avant* concaténation, car on a alors plus de comparaisons dans $\mathcal{O}$.

D'après l'invariant 1, puisque $c_n \notin M_1$, M_1 est soit inclus dans $[\![i_n, j_n]\!]$ — et le lemme est vrai — soit il le chevauche — et le lemme est faux. Supposons pour contradiction ce deuxième cas. Puisque $c_n \notin M$, on a

$$i_n < Debut(M) < j_n < Fin(M)$$

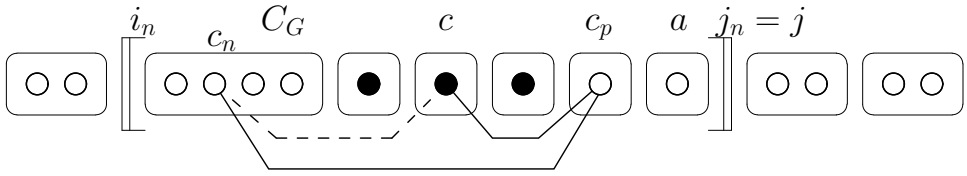
M intersecte alors une classe située à droite de a, et $a \in M$. Voyons comment `TrouverChaine` a calculé a, c'est-à-dire comment (P) a été violée.

- Si $Fin(\mathcal{C}(a)) \leqslant j$, soit D la classe située à droite de $\mathcal{C}(a)$. D viole (P), mais vérifie les trois premiers points de (P), donc son pivot d est un voisin de c_n. Or, M_1 chevauche $[\![i_n, j_n]\!]$. Donc $a \in M_1$ (car sa classe est monocaténaire, et invariant 2). De plus, comme c_n est un pivot utilisé, D ne chevauche pas $\Gamma(c_n)$ mais $D \subset \Gamma(c_n)$. Il existe donc un sommet $z \in (D \cap M)$ qui est voisin de c_n. c_n sépare M_1 : contradiction.



M_1 *en noir.*

- Sinon $Fin(\mathcal{C}(a)) = j$. Considérons l'ancien centre c_p. Lorsque c devint centre, sa classe d'alors, C_0, était bonne (invariant 5) donc tous les sommets actifs de $[\![i, j]\!]$ sont voisins de c_p si, et seulement si, c est voisin de c_p.
  - si $(c_p, c) \in E$ alors $(c_n, c_p) \in E$. $\mathcal{S}(c_p)$ est placé dans C_f, à droite de c. Or M_1 est non-adjacent à c_n, car $(c_n, c) \notin E$. D'après l'invariant 1, M_1 commence à droite de c_n et finit à gauche de c_p. Or $c_p \in [\![i, j]\!]$ et $j_n = j$, donc $c_p \in [\![i_n, j_n]\!]$, et finalement $M_1 \subset [\![i_n, j_n]\!]$.



M_1 *en noir.*

  - Si $(c_p, c) \notin E$ alors posons que Z est l'ensemble des sommets actifs de $[\![i, j]\!]$ qui n'appartiennent pas à M_1, plus le sommet c_p. `Ext-Centre`$(\mathcal{O}, c, c_p)$ place Z dans C_d, car $(c_p, c) \notin E$ et $C_f \subset M_1$ (si M_1 chevauche C_d *et* C_f il est inclus dans $[\![i, j]\!]$,

invariant 1). Puisque $c \in M_1$, tout sommet de Z est non-adjacent à M. Donc aucun pivot pris dans M ne peut couper C_d ou les classes qui en sont issues en plaçant Z dans plusieurs classes différentes : cet ensemble reste toujours dans une seule classe utilisée, de pivot c_p. Or, tous les pivot pris dans $[\![i, j]\!]$ appartiennent à M. La classe contenant Z est maintenant C_G (sinon cette classe multicaténaire serait entre C_G et c, impossible par définition de C_G), et son pivot est $c_n = c_p$. D'après le lemme 46 c_p ne peut servir une seconde fois de centre : contradiction.

□

Lemme 44

- *Si* $c_n \prec_{\mathcal{O}} c$ *alors* $C_G \subset \overline{\Gamma}($ `TrouverChaine`$(\mathcal{O}, c_n, c, i, j))$.
- *Si* $c \prec_{\mathcal{O}} c_n$ *alors* $C_D \subset \Gamma($ `TrouverChaine`$(\mathcal{O}, c_n, c, i, j))$.

Démonstration: Prouvons seulement le premier cas. Soit $a =$ `TrouverChaine`$(\mathcal{O}, c_n, c, i, j)$. $\mathcal{C}(c_n) = C_G$ et $\mathcal{C}(a)$ sont tous deux inclus dans $[\![i, j]\!]$. Mais ces deux classes sont situés des deux côtés du centre c, et ont été séparées lors de la trissection de $[\![i, j]\!]$ par `Ext-Centre` (quand c fut choisi comme centre). Le lemme 46 prouve que c_n n'est pas un ancien centre. Il est par contre possible que $a = c_p$: voyons les deux cas.

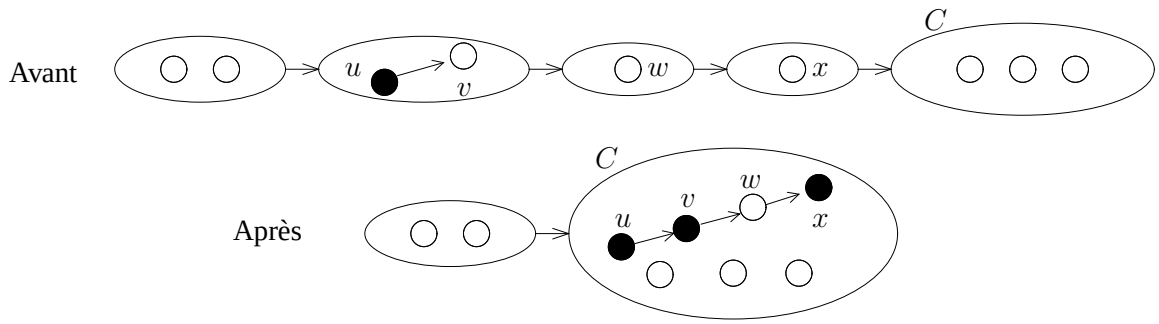
- Si $a \neq c_p$ alors a était inutilisé quand les classes C_G et $\mathcal{C}(a)$ furent séparées, mais a été utilisé depuis. Comme $c_n \prec_{\mathcal{O}} c$, a est choisi selon la propriété (P) donc $(a, c_n) \notin E$. Comme $\Gamma(a)$ a été utilisé par `Ext-Pivot`, on a $C_G \subset \overline{\Gamma}(a)$.
- Si $a = c_p$ alors, comme $(c_n, a) \notin E$, tous les sommets actifs de $[\![i, j]\!]$ sont non-voisins de cet ancien centre, et en particulier $C_G \subset \overline{\Gamma}(a)$.

□

6.4.5 La règle de concaténation : procédure `Concat`

Description

`Concat`$(\mathcal{O}, C, s, r)$ est la procédure implémentant la règle de concaténation. Elle crée une nouvelle chaîne et détruit des classes. Cette opération fait perdre des relations de comparabilité dans l'ordre partiel $\mathcal{O}$, mais n'en crée pas (en effet, une série de chaînes consécutives est un sous-ordre total, donc peut être changée en une seule chaîne). Toutes les classes comprises entre C (exclue) et $\mathcal{C}(s)$ (inclue) sont supposées être monocaténaire. Elle sont fusionnées (sans modifier $\prec_{\mathcal{O}}$) en une nouvelle chaîne S dont le sommet actif r est le représentant. Cette chaîne est alors ajoutée à la classe C.



Action de la procédure `Concat`$(\mathcal{O}, C, v, w)$*. Les sommets actifs sont en blanc.*

Preuve de validité

Lemme 45 *Si les invariants sont vérifiés avant appel à* `Concat` *alors ils le restent après cet appel.*

Démonstration: On se place dans les conditions où `Concat` est appelée. Il y a deux cas possibles : soit C est la classe C_G et $a =$ `TrouverChaine`$(\mathcal{O}, c_n, c, i, j)$ est à droite de c, soit C est la classe C_D et a est à gauche de c. On supposera le premier cas. La preuve pour le deuxième cas est similaire, en complémentant G et en retournant $\mathcal{O}$. Soient $S_1 \dots S_k$ les chaînes situées dans des classes monocaténaires qui sont concaténées en la nouvelle chaîne S.

Invariant 1

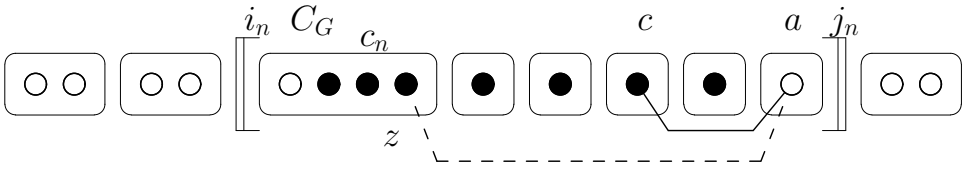
`Concat` ne crée pas de comparaisons dans $\mathcal{O}$ (bien au contraire elle en détruit) donc cet invariant reste vrai.

Invariant 2

Cet invariant ne peut être violé que s'il existe un module fort M qui chevauche S. Nous allons montrer que cela conduit à une contradiction. Supposons que $(c, c_n) \notin E$, la preuve dans l'autre cas étant semblable (quitte à complémenter G et à retourner $\mathcal{O}$).

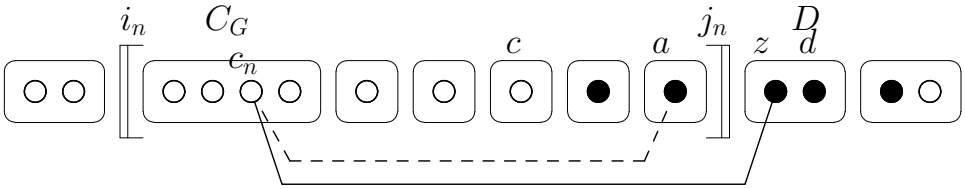
Par définition de la propriété (P) dans `TrouverChaine`, on a $(a, c_n) \notin E$. $a \in [\![i, j]\!]$ n'est pas mauvais (invariant 3) donc $(a, c) \in E$. D'après l'invariant 2 soit $S_i \subset M$ soit $S_i \cap M = \emptyset$. D'après l'invariant 1 M contient des chaînes consécutives, et peut donc chevaucher S de deux façons :

– Soit M contient $S_1 \dots S_i$ et n'intersecte pas $S_{i+1} \dots S_k$. On a alors $a \notin M$. Puisqu'il y a chevauchement, M intersecte C_G. D'après l'invariant 3, cette classe étant multicaténaire, $c \in M$. D'après le lemme 44 $C_G \subset \overline{\Gamma}(a)$. Il existe donc un non-voisin z de a dans M, mais aussi un un voisin, c. a sépare un module, contradiction.



M (sommets noirs) chevauchant S en venant de sa gauche : contradiction.

– Soit M contient $S_i \dots S_k$ et n'intersecte pas $S_1 \dots S_{i-1}$. On a alors $c_n \notin M$. Voyons comment a a été calculé.
  – Soit $Fin[\mathcal{C}(a)] = j$ (toutes les classes de $[\![i, j]\!]$ à droite du centre vérifient (P)). Comme M chevauche alors $[\![i, j]\!]$, on a $c \in M$. D'après le lemme 43 tout module fort contenant c mais pas c_n est inclus dans $[\![i_n, j_n]\!]$, ce que viole M : contradiction.
  – Sinon, soit D la classe à droite de $\mathcal{C}(a)$. D est dans $[\![i, j]\!]$ et viole (P) : sont pivot d est voisin de c_n. Puisque D et C_G sont issues l'une de C_d et l'autre de C_f, le pivot c_n de C_G ne coupe pas D mais on a $D \subset \Gamma(c_n)$. Puisqu'il y a chevauchement, il existe un sommet $z \in M$ voisin de c_n, mais aussi un non-voisin, a. c_n sépare un module : contradiction.



M (sommets noirs) chevauchant S en venant de sa droite : contradiction.

Invariant 3

Soit M un module fort. Si avant concaténation il vérifie les points 1 ou 3 de cet invariant, alors il continue de les vérifier après concaténation. Sinon, supposons que toutes les classes intersectant M soit monocaténaires, mais que $c \notin M$. Soit S la chaîne produite par la concaténation. Nous venons de prouver que M ne chevauche pas S (preuve de l'invariant 2 ci-dessus). Si $M \subset S$ alors après concaténation M est inclus dans une seule classe, celle de S. Si $M \cap S = \emptyset$ le module n'est pas affecté par la concaténation. Et si $S \subset M$, vu que $c_n \notin M$, la preuve de l'invariant 2 ci-dessus a montré que $S = M$.

Invariant 4

Avant l'appel $\texttt{Concat}(\mathcal{O}, C, c, a)$, $\mathcal{C}(c)$ ne contient que c comme sommet utilisé actif car elle est monocaténaire, et d'après l'invariant 2 $\mathcal{C}(c_n)$ contient aussi sommet seul utilisé actif, c_n. Après concaténation la classe produite contient ces deux sommet utilisés, mais aussi le centre, et l'invariant est vrai.

Invariant 5

Il n'y a pas de mauvais sommets dans $[\![i_n, j_n]\!]$: cet invariant reste vrai.

Invariant 6

La procédure `Concat` travaillant uniquement avec des sommets situés à l'intérieur de $[\![i, j]\!]$, cet invariant reste vrai.

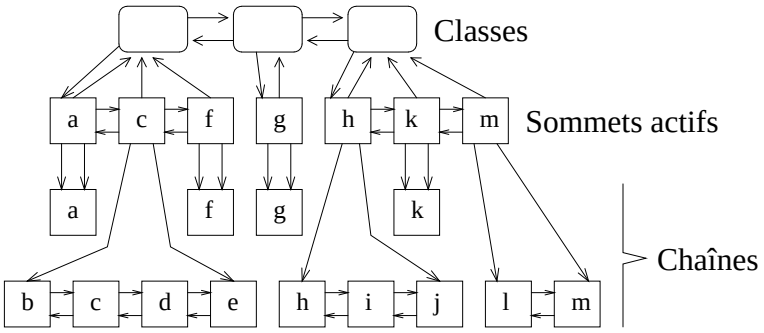
□

6.5 Implémentation séquentielle

L'algorithme qui est décrit dans la section précédente peut être implémenté afin de s'exécuter en temps $O(n + m)$. Pour ce faire nous utilisons une implémentation simple des partitions ordonnées de chaînes (proche d'une structure d'arbre), puis nous réécrivons de façon plus technique les procédures afin de les adapter à cette structure et de mettre en lumière tous les détails techniques. Après cela nous prouvons la complexité.

6.5.1 Implémentation des partitions ordonnées de chaînes

Nous proposons une implémentation à trois étages, basée sur des listes doublement chaînées. Au premier étage on trouve une liste doublement chaînée des classes. Au deuxième étage, pour chaque classe, une liste doublement chaînées des sommets actifs de cette classe. Et au troisième étage, pour chaque sommet actif, une liste doublement chaînée des sommets de la chaîne qu'il représente.



Implémentation d'une partition ordonnées de chaînes. Les flèches représentent les pointeurs.

Enfin, un certains nombre de pointeurs permettent de faire en $O(1)$ toutes les opérations souhaitées. Chaque sommet actif pointe sa propre classe : ainsi, changer un sommet de classe se fait en $O(1)$. Comme l'ordre des sommets d'une classe est quelconque, le nouveau venu pourra être inséré en tête. Cette liste des sommets d'une classe ne sert qu'à une chose : choisir en $O(1)$ un nouveau pivot. Un pointeur sur la tête suffit. La liste d'une chaîne, associée à un sommet actif, n'est pas utile au déroulement de l'algorithme mais seulement à la production du résultat. Afin de réaliser en $O(1)$ la concaténation de deux chaînes, chaque sommet actif maintient un pointeur sur la tête et un sur la queue.

Il y a en plus un certain nombre de champs associés à chaque structure. Pour une classe on a besoin de

- indices numériques $Debut$ et Fin des bords (afin de comparer la position des classes),
- nombre de sommets actifs,
- drapeau indiquant si la classe est utilisée, et si oui nom du pivot,
- pointeur sur la tête de la liste des sommets,
- pointeurs sur les classes précédentes et suivantes.

Et pour un sommet :

- numéro du sommet dans le graphe de départ,
- liste d'adjacence,
- pointeurs sur les sommets précédents et suivants au sein de la chaîne,
- drapeau indiquant si le sommet est actif. Si inactif, tous les, champs qui suivent sont sans valeur définie
- pointeurs sur la tête et la queue de sa chaîne,
- taille de la chaîne (pour calculer les bords des classes),
- pointeur sur sa classe,
- pointeurs sur les sommets précédents et suivants au sein de la classe.

Les sommets peuvent être placés au sein d'un tableau, car leur nombre ne varie pas. Afin de soulager le gestionnaire de mémoire on peut aussi placer les classes, dont le nombre est borné par n, dans un tableau.

6.5.2 Implémentation de `POC-ext` en temps (total) $O(n)$

La procédure `POC-ext` peut être programmée de sorte que le temps mis par l'appel initial et tous les appels récursifs (mais sans compter le temps mis pas les quatre procédures et fonctions auxiliaires) soit $O(n)$. Pour cela, il y a trois points à expliciter dans l'algorithme 6.

Choix d'une classe inutilisée

La procédure `POC-ext` a besoin de pouvoir choisir en $O(1)$ une classe inutilisée dans le facteur de travail, ou de savoir qu'il n'y en a pas. Voici un mécanisme utilisable.

D'après la remarque 1, les facteurs de travail sont imbriqués. On peut leur donner à chacun une *profondeur*, qui est le niveau de l'appel récursif correspondant au facteur. On va également attribuer aux classes une profondeur, de la façon suivante : la première classe $\{V\}$ est à la profondeur 0. Lorsque la règle du centre coupe une classe de profondeur p, elle crée trois classes de profondeur $p+1$. Mais quand la règle du pivot coupe une classe, les classes obtenues ont même profondeur que l'originale.

La profondeur est bornée par n (voir lemme 46). On crée n piles $I_1 \dots I_n$ (l'initialisation prend $O(n)$ en tout ; ce sont des variables globales). Lorsqu'une classe inutilisée est créée à la profondeur p, on l'inscrit dans la pile U_p.

Soit P la profondeur de l'appel récursif courant. Il est facile de voir que les classes de l'intervalle de travail sont exactement celles à la profondeur P ; la pile U_P contient donc toutes les classes inutilisées du facteur de travail. La boucle « **tant que** $[\![i, j]\!]$ contient des classes inutilisées » n'a qu'à puiser dans cette pile.

Choix d'un pivot

Puisque chaque classe possède une liste de sommets actifs, le pivot d'une classe inutilisée est juste la tête de cette liste.

Calcul de C_G et C_D

Il est nécessaire de savoir trouver C_G et C_D « assez vite », ou de pouvoir affirmer que le côté gauche (resp. droit) de l'intervalle de travail ne contient pas de classe multicaténaire. Pour cela on utilise deux variables locales. Elles sont initialisées a $\mathcal{C}(c)$, puis suivent la liste chaînée des classes par incréments, l'une vers la gauche, l'autre vers la droite, en s'arrêtant sur les classes multicaténaires. Le temps mis pour mettre ces variables à jour, pris sur tous les appels de `POC-ext`, est égal au nombre de classes créées, borné par n.

Complexité temporelle

D'après le lemme 46, un sommet donné ne peut servir qu'une fois de centre. Chaque itération dans la boucle « **répéter** jusqu'au retour de procédure » soit fait un appel récursif (et donc consomme un nouveau centre) soit retourne de l'appel en cours : il y a en tout moins de $2n$ tous dans cette boucle. Et chaque itération de la boucle « **tant que** $[\![i, j]\!]$ contient des classes inutilisées » utilise un nouveau pivot : d'après le lemme 46 il y en a au plus n en tout. Donc, la complexité temporelle de tous les appels récursifs de `POC-ext` est $O(n)$.

6.5.3 Programmation de `Ext-Pivot` et de `Ext-Centre` en temps $O(|\Gamma(p)|)$

L'algorithme 9 est une implémentation efficace de la procédure `Ext-Pivot`. Pour chaque classe X qui est coupée par `Ext-Pivot`$(\mathcal{O}, p, c)$, la classe X_a est une nouvelle classe vers laquelle sont déplacée des chaînes, tandis que X_b est formée de l'ancienne X avec les chaînes qui n'ont pas bougé. Ainsi, couper une classe X ne se fait pas en $O(|X|)$, mais bien en $O(|\Gamma(p)|)$ pour *toutes* les classes en même temps !

Chaque classe possède un champ $EnCoupe$ qui vaut vrai si la classe est en train d'être coupée par l'appel courant, ainsi qu'un champ Xa qui pointe sur la classe X_a dont cette classe est le X_b. Les classes coupées sont stockées dans une pile P, afin qu'une deuxième passe, le long de cette pile, supprime les classes vides (pour le cas $X = X_a$) et réinitialise les $EnCoupe$. Une première passe optionnelle peut détecter les classes qui ne seront pas réellement coupées.

Avec la structure de donnée décrite ci-dessus, déplacer une chaîne se fait en $O(1)$, en déplaçant son sommet représentatif. Il est clair que la procédure peut donc s'exécuter en $O(|\Gamma(p)|)$.

Algorithme 9: Procédure `Ext-Pivot`$(\mathcal{O}, p, c)$

début
  pour chaque *sommet actif* $x \in \Gamma(p)$ *tel que* $Classe[x] \neq Classe[p]$ **faire**
    $X = Classe[x]$
    si $EnCoupe[X] =$ `faux` **alors**
      si X *est située entre* $Classe[p]$ *et* $Classe[c]$ **alors**
        $Xa[X] \leftarrow NouvelleClasseAGaucheDe(X)$
      sinon
        $Xa[X] \leftarrow NouvelleClasseADroiteDe(X)$
      fin
      $Empiler(P, X)$
      $EnCoupe[X] \leftarrow$ `vrai`
    fin
    $Classe[x] \leftarrow Xa[X]$
    Modifier les bords et liste de la classe en conséquence
  fin
  pour chaque *classe* $X \in P$ **faire**
    $EnCoupe[X] \leftarrow$ `faux`
    si $X = \emptyset$ **alors** $DetruireClasse(X)$
  fin
fin

De même, `Ext-Centre`$(\mathcal{O}, c, c_p)$ peut être implémentée avec un balayage de $\Gamma(c)$. Le code est encore plus simple, car seules trois classes sont créées.

6.5.4 Programmation de `TrouverChaine` et `Concat` en $O(|\Gamma(c)| + |S|) + 1$

En §6.4.4 on a vu que `TrouverChaine`$(\mathcal{O}, c_n, c, i, j)$ retournait l'unique sommet actif a d'une classe vérifiant une certaine propriété (P) (resp. (P')), et tel que toutes les classes entre lui et $\mathcal{C}(c)$ vérifient aussi cette propriété. Un parcours de la liste chaînée des classes, partant de $\mathcal{C}(c)$ et allant dans la direction opposée à c_n, peut trouver a en vérifiant si chaque classe vérifie P (resp. P').

(P) est une conjonction de quatre propriété dont trois, avec notre structure de donnée, peuvent être vérifié en $O(1)$. Le quatrième est un test d'adjacence. Afin de le réaliser en $O(1)$, un tableau $N[n]$ est utilisé. Il reflète la liste d'adjacence de c_n. Son initialisation est faite une fois pour toutes : c'est une variable statique. Une première boucle copie la liste d'adjacence, une deuxième boucle parcourt les classes pour trouver le résultat, une troisième réinitialise N. Le code présenté ici est dans le cas où c est à droite de c_n : (P) est testée. L'autre cas est semblable.

Les premières et troisièmes boucles tournent en $O(|\Gamma(c_n)|)$. Or le lemme 46 garantit qu'un sommet ne sera choisi qu'une fois comme centre : la complexité totale de toutes ces boucles est $O(m)$.

La deuxième boucle tourne en $O(|S|)+1$, où S est l'ensemble des classes monocaténaires testées. Mais toutes ces classes seront juste après fusionnées en une seule chaîne, par la règle de concaténation. $|S|-1$ sommet actifs deviennent inactifs ; or un sommet ne devient inactif qu'une fois. Donc la complexité totale de ces boucles est $O(n)$.

Algorithme 10: Fonction `TrouverChaine`$(\mathcal{O}, c_n, c, i, j)$, cas $c_n \prec_{\mathcal{O}} c$

début
  pour chaque $x \in \Gamma(c_n)$ **faire** $N[x] \leftarrow$ `vrai`
  $C \leftarrow Classe[c]$
  répéter
    $D \leftarrow ClasseADroiteDe(C)$
    $p \leftarrow Pivot[D]$
    si *D est multicaténaire* ***ou*** $N[p] =$ `vrai` ***ou*** $Fin[C] = j$ **alors** fin de boucle
    sinon $C \leftarrow D$
  fin
  pour chaque $x \in \Gamma(c_n)$ **faire** $N[x] \leftarrow$ `faux`
  retourner $Pivot[C]$
fin

`Concat`$(\mathcal{O}, C, a, c)$ peut être programmée $O(|S|)+1$, par un simple parcours de la liste chaînée des classes, puisque dans notre structure concaténer deux chaînes se fait en $O(1)$. Le temps total mis par les appels est là encore $O(n)$. En fait, pour des raisons d'efficacité on peut combiner `TrouverChaine` et `Concat` en une seule procédure.

6.5.5 Complexité temporelle

Lemme 46 *Un sommet donné peut être utilisé au plus une fois comme centre et deux fois comme pivot.*

Démonstration: D'après l'invariant 4 si un sommet est utilisé une première fois comme pivot, sa classe devient utilisé, et il ne peut plus servir de pivot, à moins qu'il ne devienne centre. Reste donc à prouver qu'un sommet ne devient centre qu'une fois.

Le nouveau centre c_n est le pivot d'une des deux classes multicaténaires entourant le centre c. Le facteur de travail ne contient qu'un seul ancien centre actif : c_p (invariant 4, pris au moment où c devint centre). Montrons que c_p ne peut être repris comme nouveau centre.

Tout d'abord c_p est un bon sommet : dans le cas contraire il est exclu par `Ext-Centre` du facteur de travail. Supposons sans perte de généralité $(c, c_p) \notin E$. c_p est donc à gauche de c ; il

peut être p_G. Si tel est le cas, considérons l'instant où `Concat`$(\mathcal{O}, C_z, c_p, a)$ fut appelé. devint centre. On avait $C_z \subset \overline{\Gamma(c_p)}$. Tous les sommet actifs de $[\![i, j]\!]$, sauf c_p, sont issus de C_z, donc $(p_G, p_D) \notin E$. Le test de choix de nouveau centre prendra donc $c_n = p_D$.

□

Théorème 37
S'il prend en entrée un graphe $G = (V, E)$, l'algorithme 6 termine en $O(n + m)$.

Démonstration: Le lemme précédent garantie que l'algorithme termine effectivement. Comme on a vu :

- Le temps cumulé mis pas tous les appels de `POC-ext` est $O(n)$
- `Ext-Centre`$(\mathcal{O}, c, c_p)$ peut être effectué en $O(|\Gamma(c)|)$, et comme un sommet sert de centre au plus une fois, le temps cumulé est $O(n + m)$.
- `Ext-Pivot`$(\mathcal{O}, p, c)$ peut être effectué en $O(|\Gamma(p)|)$, et comme un sommet sert de pivot au plus deux fois, le temps cumulé est $O(n + m)$.
- `TrouverChaine`$(\mathcal{O}, c_n, c, i, j)$ peut être calculé en $O(|\Gamma(c)|+|S|)+1$. Comme $\sum |S| = n$ et qu'un sommet sert de centre au plus une fois, le temps cumulé est $O(m + n)$.
- `Concat`$(\mathcal{O}, C, a, c)$ peut être effectué en $O(|S|) + 1$: le temps cumulé est $O(n)$.

□

6.6 Implémentation parallèle

L'analyse de complexité de la section précédente montre que l'essentiel des calculs se fait en $O(n)$; le m dans la borne vient uniquement des parcours de listes d'adjacence faits par `Ext-Centre`, `Ext-Pivot` et `TrouverChaine`. Or, on peut les paralléliser, de sorte que les temps de calcul de ces fonctions deviennent respectivement $O(1)$, $O(1)$ et $O(|S|) + 1$ (avec les notations du théorème 37). Le reste de l'algorithme demeure séquentiel. En relisant la démonstration de ce même théorème, on voit que l'on aura un algorithme en $O(n)$ de calcul d'une permutation factorisante.

Nous utilisons une machine parallèle à n processeurs, avec une mémoire partagée de $O(n)$, plus $O(n)$ de mémoire par processeur. Les phases *séquentielles* (comme dans la version précédente) et *parallèles* alternent. En phase séquentielle, chaque processeur correspond à un sommet. Il y a quatre sortes de phases parallèles :

- **Prétraitements**
  Le processeur P_i commence par construire, en $O(n)$, le *vecteur d'adjacence* du sommet i, d'après sa liste d'adjacence : les tests d'adjacence se feront on $O(1)$.
- **choix d'un pivot**
  La structure de donnée d'une partition de chaînes ne contient plus la liste chaînée des sommets (la raison apparaîtra au point suivant). Pour trouver un pivot à la classe C, et

comme la classe d'un sommet se calcule toujours en $O(1)$, on met en concurrence les n processeurs : le premier qui trouve un sommet de C gagne la course. Cela prend $O(1)$.

– **Calcul de** `Ext-Pivot`$(\mathcal{O}, p, c)$
Le processeur P_i détermine, en $O(1)$, si $\mathcal{C}(i) \neq \mathcal{C}(p)$ et si $i \in \Gamma(p)$ (grâce au vecteur d'adjacence). Si tel est le cas, il y a lieu de changer la classe de i. Comme il n'y a plus de liste chaînée des sommets d'une classe, la classe peut être coupée en parallèle. Cela soulève les problèmes suivants :

  – Le premier problème d'exclusion mutuelle est la création de la nouvelle classe X_a. On le résout en doublant préventivement toutes les classes (sauf celle du pivot) par une classe X_a, située du bon côté : cela se fait en $O(1)$ en parallèle, puisque le nombre de classes est borné par n. Ensuite vient la phase des déplacements effectifs. Une troisième phase, toujours en parallèle, supprime les classes vides

  – Le deuxième problème est le calcul des bords des classes. Heureusement, connaître la valeur exacte des bords d'une classe n'est pas nécessaire : on a en fait juste besoin d'un indice pour calculer l'ordre relatif des classes. Un indice réel suffit : celui d'une nouvelle classe est la moyenne de celui des classes entre lesquelles il s'insère. Comme le nombre d'opération est borné par n, une précision de $\log n$ bits est suffisante. Or, le modèle de machine que nous utilisons suppose que les opérations arithmétiques sur les nombres à $\log n$ bits se font en temps constant.

  – Le troisième et dernier problème est de maintenir le nombre de sommets actifs d'une classe. Mais la valeur de ce nombre n'est utile que pour déterminer si une classe est monocaténaire. Cela peut se déterminer en $O(1)$, en parallèle : dans un premier temps on demande à tous les processeurs, en même temps, de fournir un sommet de la classe. On arrête la course dès qu'un a répondu. On refait ensuite la même demande, mais en interdisant au lauréat de la course précédente de re-courrir : si personne ne répond, la classe était monocaténaire.

– **Calcul de** `Ext-Centre`$(\mathcal{O}, c, c_p)$
Il utilise un mécanisme semblable au précédent, mais avec juste la phase de déplacements.

– **Calcul de** `TrouverChaine`$(\mathcal{O}, c_n, c, i, j)$
La première boucle qui remplit le tableau N n'est plus nécessaire car les vecteurs d'adjacence sont déjà construits, la troisième boucle non plus. Reste la deuxième, dont le temps total d'exécution est $O(n)$: il n'y a pas lieu de la paralléliser.

Théorème 38
Calculer une permutation factorisante d'un graphe peut se faire en temps $O(n)$ et mémoire $O(n^2)$ sur une machine parallèle à n processeurs.

6.7 Conclusion

Des théorèmes 36 et 37 on déduit le théorème principal :

Cet algorithme calcule une permutation factorisante d'un graphe non-orienté en $O(n+m)$.

Il peut donc servir à faire un algorithme 2-temps de décomposition modulaire, dont il constitue le premier temps (le deuxième étant décrit dans le chapitre 8).

Du théorème 38, et de la parallélisation de l'algorithme décrit au chapitre suivant, on conclut qu'un algorithme 2-temps de décomposition modulaire parallèle peut tourner en $O(n)$ sur une machine à n processeurs. Son graphe de précédence étant très simple, il est facile à ordonnancer sur une machine avec moins de processeurs. Ce temps d'exécution est supérieur à celui de $O(\log^2 n)$ de [Dah95a], mais en contrepartie moins de processeurs sont demandés, $O(n)$ contre $O(n+m)$. C'est donc un intermédiaire entre le séquentiel et le parallèle le plus rapide.

Nous pensons qu'étant donné les fortes relations entre la décomposition modulaire et le problème de l'*orientation transitive*, ces structures et mécanismes pourraient servir à faire un algorithme d'orientation transitive d'un graphe en $O(n+m)$...

Chapitre 7

Permutation factorisante d'un graphe orienté : algorithmes en $O(n+m)$

On two occasions I have been asked [by members of Parliament], "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Charles BABBAGE
Harper's New Monthly Magazine*, Dec. 1864*

LA DÉCOMPOSITION MODULAIRE DES GRAPHES ORIENTÉS est un problème au moins aussi intéressant et utile que celle des graphes non-orientés. Bien que moins de classes de graphes orientés aient été décrites que dans le cas des graphes non-orientés, il devient envisageable d'écrire des algorithmes de reconnaissance efficaces, grâce à l'outil de la décomposition modulaire (voir [Mül97] par exemple, et les études de [Gol80, BLS99] pour se convaincre de la puissance de la décomposition modulaire pour les algorithmes de reconnaissance).

La meilleure complexité d'un algorithme de décomposition modulaire des graphes orientés est $O(n^2)$ [EGMS94, McC95][1] ou bien $O(m \log n)$ [DGM02]. Ici est présenté le premier algorithme en $O(n+m)$. Ces résultats sont le fruit d'un travail commun avec Ross McConnell. Fin 2001, j'avais présenté à Barcelone une première version, certes déjà en temps linéaire, mais insatisfaisante car relativement compliquée et peu implémentable. Elle est présenté en annexe de cette thèse, pour mémoire, dans sa version anglaise. Ross McConnell eut connaissance de ce travail, et apporta des idées complètement nouvelles. Nous réécrîmes ensemble un papier totalement différent de l'original [MM03].

1. il s'agit en fait d'algorithmes pour le cas plus général des 2-structures

Ce chapitre est court, car les algorithmes utilisés sont décrit ailleurs dans ce mémoire ou dans la littérature. Le calcul d'une permutation factorisante d'un tournoi, présenté ici comme outil servant ensuite pour le cas général, est intéressant en lui-même car il montre que dans certains cas on peut obtenir, de façon très simple, une permutation factorisante, par affinage de partitions (extension d'ordres factorisants).

7.1 Cas des tournois

L'algorithme présenté ici calcule une permutation factorisante d'un tournoi. On peut remarquer qu'il s'agit d'un algorithme **simple** et néanmoins **efficace** : il ne fait que quelques lignes de code, se prouve simplement, s'implémente facilement, et tourne en $\Theta(n^2)$, ce qui est $\Theta(m)$ dans le cas des tournois, donc linéaire en la taille de la représentation. Rappelons qu'un tournoi est un graphe $G = (V, E)$ tel que il y ait nécessairement un arc simple entre toute paire de sommets. On a donc $m = \frac{n(n-1)}{2}$. Il existe $2^{\binom{n}{2}}$ tournois étiquetés différents, donc toute représentation d'un tournoi prend $\Omega(n^2)$ bits : cet algorithme est linéaire en la taille d'une représentation optimale pour le cas général. Sa simplicité justifie son existence : en effet, il existe déjà des algorithmes en $O(n^2)$ [EGMS94, McC95] calculant l'arbre de décomposition modulaire d'une 2-structure. Il sont applicables au cas des tournois. Mais l'algorithme présenté ici est beaucoup plus court et plus facile. De plus, c'est un outil pour la cas général, où nous avons besoin à certains moments de disposer de permutations factorisantes pour certains tournois. Cet algorithme a besoin, pour un sommet v donné, indifféremment de $\Gamma^+(v)$ ou $\Gamma^-(v)$. Il effectue donc au pire $\frac{n(n-1)}{4}$ opérations de est d'adjacence et d'affinage.

Théorème et définition 37
*Soit $G = (V, E)$. Une permutation σ de V est **totalement factorisante** si tout module de G (même faible) est facteur de σ.*

Un tournoi possède une permutation totalement factorisante.

Démonstration: Ce résultat est en fait vrai pour toute famille faiblement partitive sans nœud `complet` dans son arbre de décomposition : il suffit alors d'ordonner les fils d'un nœud `linéaire` de l'arbre dans le bon ordre pour obtenir une permutation totalement factorisante.

□

7.1.1 Description

Il est procédé par affinage de partition : une partition grossière est transformée en permutation, en utilisant comme pivot le voisinage d'un sommet, et en se restreignant à la seule classe contenant ce sommet. Remarquons que la permutation produite est un chemin hamiltonien du tournoi[2].

2. C'est un résultat classique que tout tournoi possède un chemin hamiltonien. Mais un chemin hamiltonien n'est pas forcément une permutation factorisante !

Chaque sommet est *utilisé* exactement un fois. L'utilisation en question consiste en une trissection de la classe qui le contient selon son voisinage. $\mathcal{P}_i$ est la partition de V à la $i$ème étape.

Algorithme 11: Calcul d'une permutation totalement factorisante d'un tournoi

Données : Un tournoi $G = (V, E)$
Résultat : Une permutation factorisante $\mathcal{P}_n$ de G
début
  On se donne un ordre quelconque $v_1 \dots v_n$ sur V.
  $\mathcal{P}_0 \leftarrow \{V\}$.
  pour i *de* 1 *à* n **faire**
    soient C la classe de $\mathcal{P}_{i-1}$ contenant v_i,
    $C_a = C \cap \Gamma^-(v_i)$ et $C_b = C \cap \Gamma^+(v_i)$.
    remplacer C par $C_a \oplus \{v_i\} \oplus C_b$ dans $\mathcal{P}_{i-1}$ pour obtenir $\mathcal{P}_i$.
    NB: si C_a (resp. C_b) est vide, on ne l'insère pas.
  fin
fin

7.1.2 Implémentation

La structure de partition ordonnée se programme sans difficulté : on peut utiliser une liste chaînée de classes, ce qui permet de connaître l'ordre des classes et de pouvoir insérer les nouvelles classes, en cas de césure, en temps constant. Les sommets de V sont placés dans un tableau, chacun pointant sa propre classe.

Le calcul de C_a et C_b est le suivant : pour chaque sommet inscrit dans la liste d'adjacence (sortante ou entrante) de v_i on regarde s'il appartient à C, et si oui on l'en extrait pour le mettre dans une nouvelle classe, en temps $O(1)$. En fait une seule des deux liste peut être considérée. Par exemple, considérons uniquement $\Gamma^+(v)$. Les sommets de cette liste appartenant à C sont placés dans C_b. Après cela, ceux qui restent dans C forment la classe C_a.

7.1.3 Preuve

À l'étape i, la partition courante est $\mathcal{P}_i$ et le sommet servant à affiner est v_i. La correction de cet algorithme repose sur les trois invariants suivants :

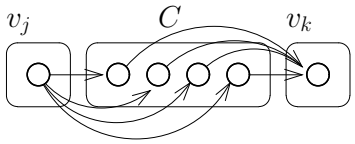
Invariant 1
Pour tout $0 \leqslant i \leqslant n$, $\mathcal{P}_i$ *est une partition de* V *possédant au moins* i *singletons.*

Démonstration: Il suffit de remarquer que C devient $C_a \oplus \{v_i\} \oplus C_b$: à l'étape i, tous les v_j, $j \leqslant i$ sont dans des singletons.

□

Invariant 2
Soit i l'étape en cours de l'algorithme, et C une classe de la partition possédant au moins deux éléments.

- *Si C n'est pas la première classe de $\mathcal{P}_i$, alors la classe immédiatement à gauche de C est un singleton $\{v_j\}$ avec $j < i$, et $C \subset \Gamma^+(v_j)$.*
- *Si C n'est pas la dernière classe de $\mathcal{P}_i$, alors la classe immédiatement à droite de C est un singleton $\{v_k\}$ avec $k < i$, et $C \subset \Gamma^-(v_k)$.*



Invariant 2

Démonstration: Cet invariant se démontre par récurrence sur i : il est trivialement vrai pour $\mathcal{P}_0$. À l'étape i, il reste vrai (par hypothèse d'induction) pour toutes les classes qui ne sont pas coupées, et est vérifié pour C_a et C_b :

- Pour les singletons qui les encadrent, vrai puisque vrai pour C.
- Pour $\{v_i\}$, vrai par construction de l'algorithme.

□

Invariant 3
$\mathcal{P}_i$ est un ordre factorisant.

Démonstration: Nous allons le démontrer par induction sur i. Il est bien sûr que $\mathcal{P}_0$ est factorisant. Supposons $\mathcal{P}_{i-1}$ factorisant et démontrons que $\mathcal{P}_i$ l'est aussi. Supposons pour contradiction qu'il existe deux modules forts M et N qui se croisent au sein de $\mathcal{P}_i$. Puisqu'ils ne se croisent pas dans $\mathcal{P}_{i-1}$, c'est que se sont les comparaisons rajoutées lors de la trissection de la classe C de v_i qui ont créé ce croisement. Donc M et N intersectent tous les deux C. Cependant, puisque leur intersection est vide, v_i ne peut appartenir qu'à l'un des deux. On peut donc supposer, sans perte de généralité, $v_i \notin N$. Voyons où sont M et N.

- Si $N||_{\mathcal{P}_{i-1}} v_i$, alors $N \subset C$. N est donc placé dans C_a ou dans C_b selon son adjacence à v_i.
- Si $N \preccurlyeq_{\mathcal{P}_{i-1}} v_i$ alors, d'après l'invariant 2, la classe qui précède C est $\{v_j\}$. Pour que le module $\{v_j\}$ ne croise pas N, on a $v_j \in N$. Et d'après l'invariant 2 $N \subset \Gamma^-(v_i)$. Donc $N \cap C = N \cap C_a$.
- De même si $N \succcurlyeq_{\mathcal{P}_{i-1}} v_i$ alors $N \cap C = N \cap C_b$.

Dans le cas $v_i \notin M$ on a les mêmes propriétés. Et si $v_i \in M$ alors

- Si $M||_{\mathcal{P}_{i-1}} v_i$, M ne peut intersecter que C_a et C_b et contient $\{v_i\}$.

– Si $M \preccurlyeq_{\mathcal{P}_{i-1}} v_i$ alors, puisque $v_j \in M$ et que, d'après l'invariant 2, $C \subset \Gamma^+(v_j)$, tout sommet de C_a distingue v_i de v_j et appartient à M : on a $\{v_j\} \uplus C_a \uplus \{v_i\} \subset M$.
– Si $M \succcurlyeq_{\mathcal{P}_{i-1}} v_i$ alors $\{v_i\} \uplus C_b \uplus \{v_j\} \subset M$.

Regardons les relations possibles entre M et N :

– Si $M||_{\mathcal{P}_{i-1}} N$ alors $M \subset C$ et $N \subset C$. Dans $\mathcal{P}_i$, N est inclus dans une seule classe (C_a ou C_b) et ne peut croiser de module.
– $M \preccurlyeq_{\mathcal{P}_{i-1}} N$ alors $M \not\succcurlyeq_{\mathcal{P}_{i-1}} v_i$ et $N \not\preccurlyeq_{\mathcal{P}_{i-1}} v_i$. On a donc $M \cap N \subset C_b$ et les deux modules ne peuvent se croiser.
– De même si $M \succcurlyeq_{\mathcal{P}_{i-1}} N$ alors $M \cap N \subset C_a$.

□

Théorème 39
L'algorithme proposé calcule une permutation totalement factorisante d'un tournoi en temps $\Theta(n^2)$.

Démonstration: La correction découle directement des invariants 1 et 3. À l'étape n, tous les v_i sont dans des singletons (invariant 1), donc nous avons un ordre total. Par l'invariant 3, c'est une permutation factorisante. La complexité annoncée vient du fait que chacun des n sommets est utilisé une fois, et que le calcul des C_a et C_b se peut faire en $O(\Gamma(n))$. Dans le cas d'un tournoi, $m = \Theta(n^2)$.

□

7.1.4 Décomposition des 2-structures antisymétriques

Définition 38
*Une 2-structure $G = (V, E, k)$ est **antisymétrique** si pour toute paire $\{x, y\}$ de sommets on a $E(x, y) \neq E(y, x)$.*

Comme me l'a fait observer Ross McConnell, on construit facilement un tournoi $T(V, E_T)$ tel que tout module de G est module de T, en prenant $(x, y) \in E_T$ ssi $E(x, y) \leqslant E(y, x)$. On peut donc ainsi produire une permutation factorisante de ce tournoi, puis calculer sa décomposition modulaire. Avec les idées présentées dans la section qui suit, on peut construire un algorithme qui transforme T_T en T_H. La complexité, $O(n^2)$, est semblable à ce qui existe déjà.

7.2 Cas général

Voici maintenant l'algorithme pour le cas général. Il utilise comme structures intermédiaires deux graphes non-orientés et une 2-structure, qui ont « presque » la même décomposition que G.

7.2.1 Trois objets auxiliaires. Propriétés de leurs modules.

Soit $G = (V, E)$ un graphe orienté. À partir de G nous définissons trois objets auxiliaires :

- Le graphe non-orienté $G_s = (V, E_s)$ tel que $(u, v) \in E_s$ ssi $(u, v) \in E \ \vee \ (v, u) \in E$
- Le graphe non-orienté $G_d = (V, E_d)$ tel que $(u, v) \in E_d$ ssi $(u, v) \in E \ \wedge \ (v, u) \in E$.
- La 2-structure $H = (V, E_H, 3)$ telle que $E_H(x, y)$ vaut 0 si $\{u, v\}$ est une non-arête, 1 s'il s'agit d'une arête et 2 s'il s'agit d'un simple arc.

Leurs arbres de décompositions modulaires seront respectivement T_s, T_d et T_H. Notons que comme H est symétrique, les décompositions modulaires de ces trois objets sont partitives et leurs arbres de décomposition n'ont pas de nœud `linéaire`.

Lemme 47 *Si M est un module de G alors M est un module de G_s et de G_d.*

Démonstration: Clairement supprimer tous les arcs simples, ou bien les remplacer par des arêtes, ne crée pas de distinctions.

□

Lemme 48 *M est un module de H si et seulement si il est module de G_s et de G_d.*

Démonstration: G_s est la 2-structure obtenue en identifiant les couleurs 1 et 2 dans H, et G_d en identifiant 0 et 2. Comme fusionner des couleurs ne crée visiblement pas de distinctions, un module de H est module de G_s et G_d. Réciproquement, si M est un module de G_s alors dans H $x \notin M$ est relié à M soit par des 0-arêtes et des 1-arêtes, soit par des 2-arêtes. Et si M est module de G_d alors x est relié à M soit par des 0-arêtes et des 2-arêtes, soit par des 1-arêtes. La conjonction des deux produit un module de H.

□

Corollaire 1
$T_H = T_s \wedge T_d$ *(avec la notation de §5.5).*

Corollaire 2
Tout module de G est module de H.

Cependant, un module fort de G peut être faible dans H. Il apparaît alors comme union de certains fils d'un nœud complet de T_H. Ce qui nous amène au lemme suivant :

Lemme 49 *Il existe une permutation factorisante de H qui est factorisante pour G.*

Démonstration: Un module de G qui est module fort de H est un facteur de toute permutation factorisante de H. Sinon ce module est union de fils d'un nœud `complet` de H. Nous allons ordonner les fils des nœuds de T_H afin de parvenir au résultat. L'ordre à donner aux fils d'un nœud `premier` est quelconque. Soit N un nœud `complet` de T_H et $\mathcal{F} = F_1, \ldots F_k$ ses fils.

Considérons les modules forts de G, $M_1, \ldots M_k$ qui sont union de plusieurs fils de N. L'ordre d'inclusion de ces modules forts est une forêt. Soit T_N l'arbre obtenu en ajoutant N comme racine à cette forêt et F_i comme feuille au plus petit nœud qui la contient. En greffant T_N entre N et ses fils F_i, et en faisant cela pour tout nœud `complet` N, on obtient un arbre dont toute permutation factorisante est module-factorisante pour G.

□

Notre algorithme ne fait rien d'autre que calculer T_H puis ordonner les fils des nœuds complets, de façons différentes pour les nœuds 0-complets et 1-complets de pour les 2-complets. On obtient une permutation factorisante de G. Combiné avec l'algorithme du chapitre 8, cela fournit un algorithme de décomposition modulaire des graphes orientés.

Cas des nœuds 0-complets et 1-complets

Soit N un nœud 0-complet ou 1-complet de T_H, ayant pour fils $F_1, \ldots F_k$. Soit $\mathcal{R}_N$ la relation sur $\{F_1, \ldots F_k\}$ telle que $F_i \mathcal{R}_N F_j$ ssi

1. F_i et F_j sont des modules de G, et
2. aucun sommet en-dehors de N ne distingue $F_i \cup F_j$.

Lemme 50 *Soit N un nœud 0-complet ou 1-complet de T_H et M un module fort de G qui est un module faible de H et tel que le plus petit module fort de H contenant M soit N. M est l'union de tous les membres d'une classe d'équivalence non-triviale de $\mathcal{R}_N$.*

Démonstration: Supposons que N soit 0-complet, et posons $M = \bigcup_{i \in I} F_i$. Entre deux F_i il n'y a que des 0-arêtes de H, donc des non-arêtes de G : M est un module parallèle de G. F_i est une composante 0-connexe de H donc ses sommets sont tous reliés entre eux par des 1-arêtes ou des 2-arêtes dans H, donc F_i est connexe dans G. F_i est donc une composante connexe de $G[M]$. $F_i \cup F_j$ est un module de G ssi $i, j \in I$, sinon un sommet hors de N le distingue (car ce n'est pas un module de G, mais il est non-adjacent à $N \setminus (F_i \cup F_j)$). Donc $\{F_i\}_{i \in I}$ est une classe d'équivalence non-triviale de $\mathcal{R}_N$.

La même preuve est valide pour des nœuds 1-complets (en prenant $\overline{G}$ à la place de G).

□

Cas des nœuds 2-complets

Soit N un nœud 2-complet de T_H, $F_1, \ldots F_k$ ses fils. Soit $S = \{x_1, \ldots x_k\}$ tel que pour tout i, $x_i \in F_i$. $G[S]$ est un tournoi.

Lemme 51 *Soit $\sigma = (x_{\sigma(1)}, x_{\sigma(2)}, ..., x_{\sigma(k)})$ une permutation totalement factorisante de $G[S]$. Toute sous-famille de $\{F_1, F_2, ..., F_k\}$ dont l'union est un module de G est un facteur de la permutation $(F_{\sigma(1)}, F_{\sigma(2)}, ..., F_{\sigma(k)})$.*

Démonstration: Soit M un module de G qui est union de fils de N. $M \cap S$ est un module de $G[S]$. Puisque σ est une permutation totalement module-factorisante de $G[S]$, $M \cap S$ est un facteur de σ. Donc M est un facteur de $(F_{\sigma(1)}, F_{\sigma(2)}, ..., F_{\sigma(k)})$.

□

7.2.2 Algorithme

Voici l'algorithme, en six étapes :

1. Calculer G_s, G_d et H (trivial)
2. Calculer T_s et T_d avec un des algorithmes de [MS99, CH94, DGM99] ou des chapitres 6 et 8
3. Calculer $T_H = T_s \wedge T_d$ avec l'algorithme de §5.5
4. Pour chaque nœud 0-complet ou 1-complet N, calculer les classes d'équivalence de $\mathcal{R}_N$ avec l'algorithme décrit ci-dessous, et ordonner les fils de N pour que chaque classe soit un facteur.
5. Pour chaque nœud 2-complet N, choisir un ensemble S arbitraire de représentants. Calculer une permutation totalement factorisante de $G[S]$ avec l'algorithme de §7.1, et trier les fils de N selon cette permutation.
6. L'ordre sur V résultant est une permutation factorisante de G. Utiliser l'algorithme du chapitre 8 pour trouver T_G.

Calcul des classes d'équivalences de $\mathcal{R}_N$

Le seul point demandant une explication complémentaire est le calcul des classes d'équivalence de $\mathcal{R}_N$. Il nous faut d'abord :

– trier toutes les listes d'adjacence du graphe selon une même permutation factorisante de H, σ (voir §5.4.1)

– calculer les modules de T_H qui sont modules de G (voir §5.4.3).

Pour un nœud N 0-complet ou 1-complet donné, il ne reste plus, pour tous ses fils F_i qui sont des modules, à calculer l'adjacence de F_i hors de N (ce qui est bien défini puisque F_i est un module). Puisque N est un facteur de σ, il suffit de parcourir l'adjacence triée d'un sommet *représentatif* de F_i et d'éliminer tout sommet dans ce facteur.

Ensuite un algorithme d'affinage de partition trouve les listes identiques. Comme elles sont triées dans le même ordre, il suffit de couper les classes selon le premier sommet de l'adjacence des fils, puis selon le deuxième, etc : quand cet algorithme termine, les sommets de la même classe sont $\mathcal{R}_N$-équivalents. Cela se fait clairement en temps linéaire en la liste des adjacences.

7.2.3 Analyse de complexité

Théorème 40
Cet algorithme calcule l'arbre de décomposition modulaire d'un graphe orienté en $O(n+m)$.

Démonstration: Nous prouvons cette borne pour chacune des six étapes de l'algorithme

- L'étape 1 se fait trivialement en $O(n+m)$
- L'étape 2 peut se faire en $O(n+m)$ en utilisant les algorithmes de décomposition modulaires en temps linéaire existants, tels celui McConnell & Spinrad [MS94b, MS99], celui de Cournier & Habib [CH94], celui de Dahlhaus, Gustedt & McConnell [DGM01] ou encore celui présenté dans ce mémoire.
- D'après le théorème 35 page 151, la taille de la famille des modules forts d'un graphe G est en $O(n(G)+m(G))$. Comme $m(G_s) = O(m(G))$ et $m(G_d) = O(m(G))$, la taille des familles de modules forts de G_s et G_d est $O(n+m)$. D'après le théorème 34 page 148, l'arbre partitif de leur intersection peut être calculé en temps linéaire.
- D'après le théorème 39, calculer une permutation factorisante d'un tournoi de n sommets prend $O(n+m) = O(n^2)$. Pour chaque nœud 2-complet N à k fils, chaque fils étant représenté par $x_i \in S$, il y a un simple arc de G joignant x_i à x_j. Le coût $O(k^2)$ du calcul de $G[S]$ peut donc être facturé à ces arcs (qui ne sont « utilisés » qu'une fois) pour un coût total de $O(n+m)$.
- Comme on l'a vu en §5.4, trier toutes les listes d'adjacence d'un graphe selon une même permutation σ peut se faire en $O(n+m)$, et discriminer les nœuds d'un arbre qui sont des modules se fait en $O(n+m)$ également. L'algorithme pour trouver les classes est linéaire en la taille des adjacences des fils. Or, on peut trouver un représentant différent par nœud d'un arbre de décomposition, et l'adjacence d'un module est une liste plus courte que l'adjacence de son représentant, ce qui prouve que cette étape se fait en $O(n+m)$ en tout.
- Enfin l'algorithme du chapitre 8 tourne en $O(n+m)$.

□

7.3 Conclusion

Cet algorithme constitue le premier algorithme en temps linéaire de décomposition modulaire des graphes orientés. Il ouvre donc la voie à de nombreux algorithmes linéaires de reconnaissance de classes de graphes orientés, ainsi qu'à la résolution efficace de problèmes de décision ou d'optimisation divers sur les graphes orientés.

Il utilise beaucoup d'algorithme existants : tri linéaire, décomposition modulaire des graphes non-orientés, chevauchements d'une famille de parties, intersection de familles partitives, et permutation factorisante d'un tournoi. Cela le rend un peu lourd et dépendant. Nous pensons qu'il

existe un moyen « direct » de parvenir à une permutation factorisante :

Conjecture 2
Il est possible de calculer une permutation factorisante d'un graphe orienté en temps $O(n+m)$ par extension d'ordres factorisants.

Chapitre 8

Décomposition modulaire : algorithme en $O(n+m)$, connaissant une permutation factorisante

Auprès de mon arbre, je vivais heureux.

Georges BRASSENS

VOICI MAINTENANT LA DEUXIÈME PARTIE de notre Algorithme Simple de décomposition modulaire. Cette section est la plus vielle de ce mémoire, puisqu'elle correspond au stage que j'ai effectué en DEA (sous la direction de Michel Habib, déjà !) Il s'agissait de reformuler un algorithme de Christian Capelle [CH97, Cap97b], obtenant l'arbre de décomposition modulaire d'un graphe à partir d'une permutation factorisante. Mais c'est finalement un algorithme entièrement neuf qui en est sorti. Il a donné lieu à un article coécrit avec Christian Capelle et Michel Habib, et publié dans *Discrete Mathematics and Theoretical Computer Sciences* sous le titre « Graph Decomposition and Factorizing Permutations » [CHM02].

Par ailleurs, on peut étendre l'algorithme, pour la décomposition modulaire des graphes d'intervalles et des graphes de permutations. Ces objets peuvent être codés en $\Theta(n+m)$ comme graphes classiques, mais aussi en $\Theta(n)$ sous la forme d'un *réaliseur*, qui est un *modèle d'intervalles normé* ou un *modèle de permutation*, respectivement. Ces codages sont optimaux pour ces deux classes. Or ces structures sont très proches (intervalles) ou exactement (permutation) une permutation factorisante. Qui plus est, on peut calculer en temps $O(1)$ les séparateurs extrêmes de deux sommets. Cela permet d'éliminer la phase en $O(m)$ de l'algorithme : on peut calculer en temps $O(n)$ la décomposition modulaire de ces graphes, codés en leur modèle ! Les deux algorithmes, proposés en fin de chapitre, sont les premiers à atteindre la borne $O(n)$, *sous-linéaire* en la taille du graphe ! Ils sont autonomes (ne faisant appel à nul autre algorithme) et simples. Ils sont le fruit d'une séance de travail avec Ross McConnell.

Revenons au problème principal : étant donné un graphe (codé sous forme de listes d'adjacences) et une de ses permutations factorisantes, il faut calculer l'arbre de décomposition modulaire du graphe. On ne s'intéresse pas à la façon dont la permutation factorisante a été produite. En particulier, pour les graphes orientés, l'algorithme peut travailler avec **toute** permutation d'un ordre total, et non pas seulement les permutations qui respectent cet ordre. Ainsi, la stricte définition des permutations factorisante est respectée, et on se donne le moins de contraintes possibles sur les permutations factorisantes utilisées. Bien évidemment, les deux chapitres précédents sont complémentaires de celui-ci, puisqu'ils proposent des algorithmes calculant une permutation factorisante pour le cas non-orienté et pour le cas orienté. Les permutations produites (dans le cas des graphes orientés) respectent les ordres : la sixième passe n'est alors pas nécessaire.

Cet algorithme est *simple* –en tous cas suffisamment simple pour avoir été programmé – et *efficace*, puisque les calculs se font en un temps $O(n+m)$ optimal pour le codage standard des graphes (listes d'adjacences).

Voici son principe en bref : une paire de sommets consécutifs, le long d'une permutation factorisante, ne peut être distinguée que par des sommets proches d'elles, puisque membres du même module fort, donc du même facteur. Partant de cette constatation, on définit la *fracture* comme étant le plus petit facteur contenant les séparateurs d'une paire et la paire elle-même. On les coupe en deux : une fracture gauche et une fracture droite. Ces $O(n)$ facteurs sont entourés de *parenthèses* : on obtient alors une **permutation factorisante parenthésée**. À un mot parenthésé correspond, de façon bijective, un arbre. Cet **arbre des fractures** est une *bonne approximation* de l'arbre de décomposition modulaire : quelques manipulations sont suffisantes pour le transformer effectivement en $\mathcal{T}_G$.

Les premières sections de ce chapitre présentent un objet qui se construit avec une permutation factorisante, l'*arbre des fractures*, et établissent sa grande proximité avec l'arbre de décomposition modulaire du graphe. Ensuite l'algorithme proprement dit est décrit. Par souci de clarté, il est présenté ici en six étapes —alors qu'on peut le programmer en trois étapes— dont chacune prend en entrée la sortie de la précédente. La première étape réclame le graphe et une permutation factorisante, la dernière délivre l'arbre de décomposition modulaire.

8.1 Fractures

Soit $G=(V,E)$ un graphe (orienté ou non) et σ une permutation factorisante de G. L'ordre le long de σ se note $\prec$; le prédécesseur de x est $'x$ et son successeur x' ; $\min_\sigma(S)$ pour $S \subset V$ retourne l'élement x qui minimise $\sigma(x)$ dans S et $\max_\sigma$ celui qui maximise. Rappelons également qu'un sommet x ***sépare*** une partie $S \subset V$ si $x \notin S$ et que S n'est pas un module de $G[S \cup \{x\}]$. Pour un graphe non-orienté, cela veut dire que S chevauche le voisinage de x. $\mathrm{Sep}(S)$ est l'ensemble des séparateurs de S. Les modules sont une notion découlant directement de ce concept :

Proposition 67
M est un module si et seulement si $\mathrm{Sep}(M) = \emptyset$.

Pour tout module M et tout $S \subset M$, $\mathrm{Sep}(S) \subset M$.

Considérons deux sommets consécutifs dans σ, x et x', et soit M le *plus petit* module contenant $\{x, x'\}$. Nous allons *approximer* M en regardant le facteur de σ correspondant aux séparateurs de $\{x, x'\}$. Plus formellement :

Définition 39
Soit $\{x, x'\}$ une paire de sommets consécutifs dans σ.
*Le **premier séparateur** de $\{x, x'\}$ est $ps(\{x, x'\}) = \min_\sigma(Sep(\{x, x'\}) \cup \{x, x'\})$.*
*Le **dernier séparateur** de $\{x, x'\}$ est $ds(\{x, x'\}) = \max_\sigma(Sep(\{x, x'\}) \cup \{x, x'\})$.*
*Si $ps(\{x, x'\}) \neq x$, la **fracture gauche** de $\{x, x'\}$ est le facteur $[ps(\{x, x'\}), \dots x]$.*
*Si $ds(\{x, x'\}) \neq x'$, la **fracture droite** de $\{x, x'\}$ est le facteur $[x', \dots ds(\{x, x'\})]$.*

Un facteur F est une **fracture** si elle est fracture gauche ou droite d'une paire de sommets consécutifs.

Lemme 52 *Un module ne peut pas chevaucher une fracture.*

Démonstration: Soit M un module et F une fracture (gauche ou droite, peu importe) correspondant à une paire de sommets consécutifs $\{x, x'\}$. Regardons les relations entre $\{x, x'\}$ et M.

- D'après la proposition 67, si $\{x, x'\} \subset M$ alors $Sep(\{x, x'\}) \subset M$, donc $F \subset M$.
- Si $x \in M$ et $x' \notin M$, alors la fracture droite de $\{x, x'\}$ n'intersecte pas M. Et si F est la fracture gauche de $\{x, x'\}$, alors M et $\{x, x'\}$ sont deux facteurs de σ partageant la même extrémité droite, et donc ne peuvent se chevaucher.
- Le cas $x \notin M$ et $x' \in M$ est symétrique du précédent.
- enfin si ni x ni x' n'appartiennent à M, vu que M est un module, on a $M \subset \mathrm{Sep}(\{x, x'\})$ donc $M \subset F$, ou bien $M \cap \mathrm{Sep}(\{x, x'\}) = \emptyset$ donc $M \cap F = \emptyset$.

□

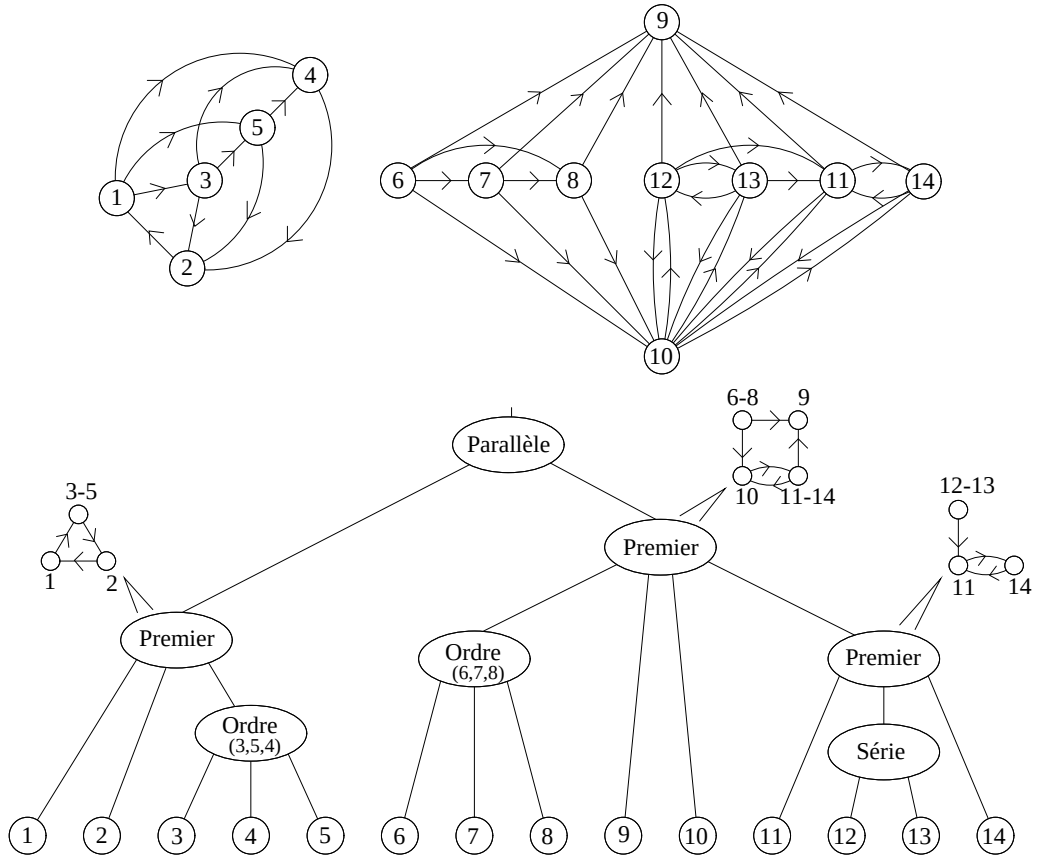
Considérons un module fort M, correspondant au facteur $[a, \dots b]$ de σ. Puisque les fractures ne chevauchent pas les modules, les fractures qui contiennent $'a$ et a contiennent tous les sommets de M jusqu'à b, et celles qui contiennent b et b' contiennent aussi tout M.

8.2 Parenthésage de fracture

Puisqu'une fracture est un facteur de σ, on peut l'entourer de **parenthèses**. Si $[a, \dots b]$ est une fracture, on « dessine » sur σ une parenthèse ouvrante entre $'a$ et a, et une parenthèse fermante entre b et b'. La **permutation factorisante parenthésée** $\hat{\sigma}$ est le *mot de parenthèses* ainsi obtenu. Formellement, il s'agit d'une suite à valeurs dans $V \cup \{(,)\}$. Quand plusieurs parenthèses ouvrantes et fermantes tombent entre les deux mêmes sommets, toutes les fermantes sont d'abord

écrites, puis toutes les ouvrantes. Ainsi, une paire de parenthèses n'enserre jamais du vide « ( ) » mais toujours au moins un sommet.

Prenons un exemple. Considérons le graphe suivant, dessiné avec son arbre de décomposition modulaire.



Il admet l'identité $\sigma = 1, \dots 14$ comme permutation factorisante. Celle-ci, après parenthésage, devient

$$\hat{\sigma} = \overset{5}{(}\ \overset{2}{(}\ 1\ \overset{1}{(}\ 2\ \overset{2}{)}\ \overset{2}{(}\ 3\ \overset{3}{(}\ 4\ 5\ \overset{3}{)}\ \overset{2}{)}\ \overset{1}{)}\ \overset{5}{)}\ \overset{5}{(}\ \overset{10}{(}\ 6\ 7\ 8\ \overset{8}{(}\ 9\ \overset{9}{(}\ 10\ \overset{10}{)}\ \overset{5}{)}\ \overset{10}{(}\ \overset{13}{(}\ 11\ \overset{11}{(}\ 12\ 13\ \overset{13}{)}\ \overset{10}{)}\ 14\ \overset{11}{)}\ \overset{9}{)}\ \overset{8}{)}$$

Pour plus de clarté, un petit nombre écrit en haut de chaque parenthèse signale le numéro du premier sommet de la paire qui a donné naissance à la fracture ayant généré cette parenthèse. Par exemple, la paire $\{1, 2\}$ ne génère qu'une fracture droite, $[2, \dots 5]$. On trouve donc entre 1 et 2 une parenthèse ouvrante (surmontée d'un 1), et entre 5 et 6 une parenthèse fermante (également surmontée d'un 1). Il n'est pas utile dans la suite de l'algorithme de savoir quelle paire génère quelle parenthèses : la seule information que nous utiliserons est juste le parenthésage.

On appelle **mot de Dyck** un mot de parenthèse correct, c'est-à-dire ayant autant de parenthèses ouvrantes que de fermantes, et jamais plus de parenthèses fermées que d'ouvertes. Appelons **hauteur de Dyck** le nombre de parenthèses ouvertes à un endroit donné du mot. La hauteur de Dyck est donc positive ou nulle partout. Puisque nous faisons un parenthésage avec autant de

parenthèses ouvrantes que de parenthèses fermantes et que la parenthèse ouvrante d'une fracture est toujours écrite avant la fermante, $\hat{\sigma}$ est un mot de Dyck.

Notons $h(x)$ la hauteur de Dyck au niveau du sommet x de $\hat{\sigma}$. Soit M un module fort. Une fracture est **interne** à M si elle est incluse (strictement) dans M. Retirons toutes les fractures internes de M. Il n'y a plus que des fractures traversant M de part en part, donc

$$\forall x, y \in M \; h(x) = h(y)$$

Cette quantité sera appelée la **hauteur** de M et notée $h(M)$. Remettons maintenant les fractures internes : on a

$$\forall x \in M \; h(x) \geqslant h(M)$$

M et les parenthèses de toutes ses fractures internes induisent un mot de Dyck.

8.3 L'arbre des fractures

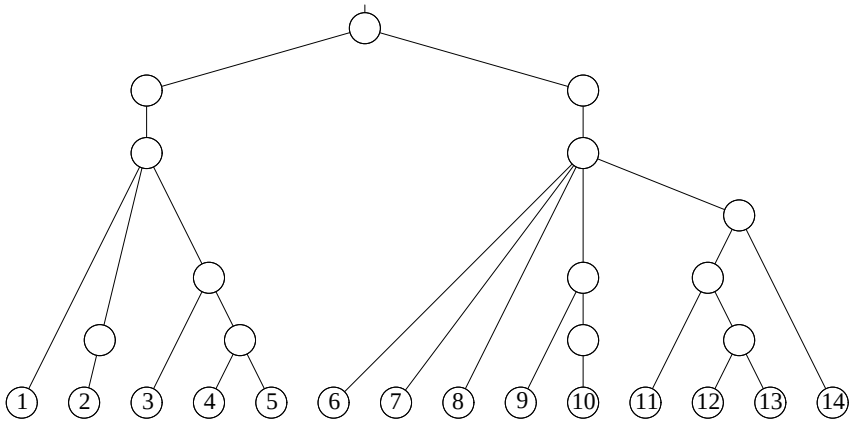
Un exercice classique de combinatoire est de mettre en bijection les mots de Dyck et les arbres enracinés ordonnés. Dans notre cas, le mot de parenthèse contient aussi des lettres (les sommets de V), mais on peut toujours lui associer un arbre, de manière biunivoque. C'est exactement le même procédé que celui permettant de passer d'une expression arithmétique à son arbre d'évaluation.

Soit w un mot de Dyck. L'arbre correspondant à w, noté $\mathcal{T}_w$, est produit par un automate transducteur, lisant w de gauche à droite et produisant $\mathcal{T}_w$ (algorithme 12).

Nous appellerons **arbre des fractures** l'arbre produit par cet algorithme quand on lui donne $\hat{\sigma}$ en paramètre. on le notera $\mathcal{T}_\sigma$. On notera qu'un parcours en profondeur de $\mathcal{T}_\sigma$ redonne σ.

$$\hat{\sigma} = \overset{5}{(}\,\overset{2}{(}\,1\,\overset{1}{(}\,2\,)\,\overset{2}{(}\,\overset{2}{3}\,\overset{3}{(}\,4\,5\,)\,)\,)\,\overset{3}{(}\,\overset{2}{(}\,6\,7\,\overset{1}{8}\,\overset{5}{(}\,9\,\overset{5}{(}\,10\,)\,)\,\overset{10}{(}\,\overset{5}{(}\,\overset{10}{(}\,11\,\overset{13}{(}\,12\,13\,)\,)\,14\,)\,)\,)$$

permutation factorisante parenthésée de l'exemple



Arbre des fractures correspondant.

Algorithme 12: Transformation d'un mot de Dyck en arbre

Données : Un mot de Dyck w sur un alphabet $V \uplus \{(,)\}$
Résultat : L'arbre enraciné ordonné $\mathcal{T}_w$ associé à w
début
  NœudCourant←*Racine*
  pour i *de* 1 *à* $|w|$ **faire**
    si $w[i] = ($ **alors**
      Créer un nouveau nœud N et l'insérer comme dernier fils de *NœudCourant*
      Père$[N]$ ←*NœudCourant*
      NœudCourant← N
    sinon si $w[i] =)$ **alors**
      NœudCourant← Père[*NœudCourant*]
    sinon
      /* $w[i] \in V$ */
      Créer un nouvelle feuille étiquetée par $w[i]$ et l'insérer comme dernier fils de *NœudCourant*
    fin
  fin
fin

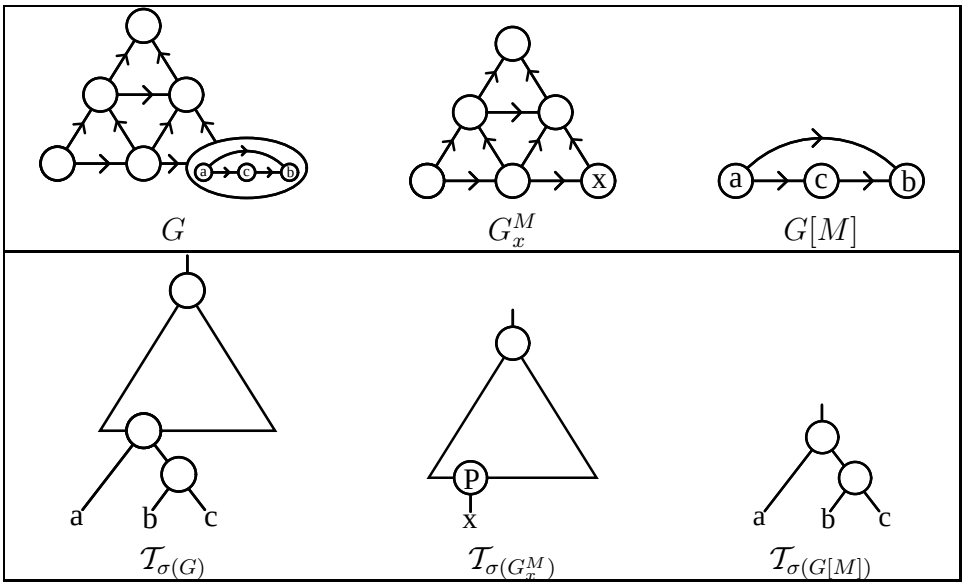
Par la suite, pour $\mathcal{T}_\sigma$ comme pour l'arbre de décomposition modulaire $\mathcal{T}_G$, un nœud est identifié à l'ensemble des feuilles qui descendent de lui, et on dira qu'un module est **représenté** s'il y a un nœud qui lui est identifiable. Le **bord gauche** d'un nœud N est la plus à gauche (dans σ) de ses feuilles, on le note $bg(N)$. De même le **bord droit** de N est noté $bd(N)$.

$\mathcal{T}_\sigma$ est une *approximation* de $\mathcal{T}_G$. En effet, il est dépendant de la permutation factorisante choisie. De plus, il possède certains nœuds en trop (qui ne représentent donc rien), tandis que des nœuds qui devraient normalement être présents dans $\mathcal{T}_G$ peuvent être absents de $\mathcal{T}_\sigma$ (et on a donc des modules non-représentés). C'est le but des étapes suivantes de cet l'algorithme que de remédier à ces défauts. Mais en attendant, voyons en quoi l'approximation fournie par $\mathcal{T}_\sigma$ est bonne.

8.4 Nœuds manquants : les fusions

Lemme 53 *Soit M un module fort de G, et G_x^M le graphe obtenu en compactant le module M en un nouveau sommet x. Considérons une permutation factorisante $\sigma(G)$ et les deux permutations factorisantes compatibles obtenues à partir d'elle $\sigma(G_x^M)$ et $\sigma(G[M])$.*

$\mathcal{T}_{\sigma(G)}$ est la composition de $\mathcal{T}_{\sigma(G_x^M)}$ et de $\mathcal{T}_{\sigma(G[M])}$, en identifiant la racine de $\mathcal{T}_{\sigma(G[M])}$ avec le père *de la feuille x de $\mathcal{T}_{\sigma(G_x^M)}$.*



Démonstration: Regardons les fractures de $\sigma(G)$. D'après le lemme 52 il y en a de trois sortes seulement :

- celles qui contiennent M ou ne l'intersectent pas,
- celles qui sont strictement incluses dans M,
- celles qui sont égales à M.

Or, les fractures du premier type deviennent exactement les fractures de $\sigma(G_x^M)$ (en transformant M en x). La preuve est immédiate ; en particulier une fracture produite par un des bords de M mais contenant d'autres sommets que M devient une fracture produite par x. De même, les fractures du deuxième type sont exactement les fractures de $\sigma(G[M])$, puisqu'internes à M. Enfin, les fractures du troisième type sont au nombre de deux au plus : une fracture droite causée par le bord gauche de M, et une fracture gauche causée par le bord droit de M. Selon leur nombre (0, 1 ou 2), il pendra à la place de x une chaîne à zéro, un ou deux nœuds d'arité un.

□

La démonstration de ce lemme fait apparaître le problème auquel nous allons devoir faire face. Appelons P le père de x dans $\mathcal{T}_{\sigma(G_x^M)}$. S'il n'y a aucune fracture représentant M dans $\sigma(G)$, et si la racine de $\mathcal{T}_{\sigma(G[M])}$ a plusieurs fils $F_1 \dots F_k$, alors ces fils apparaissent dans $\mathcal{T}_{\sigma(G)}$ comme fils de P directement et donc **il n'y a pas de nœud pour représenter M dans $\mathcal{T}_{\sigma(G)}$!** Par contre dans tous les autres cas, donc si une fracture est égale à M ou si la racine de $\mathcal{T}_{\sigma(G[M])}$ n'a qu'un fils, il y a bien un nœud pour représenter M. C'est le rôle du théorème suivant de préciser quand cela arrive.

Théorème 41 (Représentation des modules forts)
Soit M un module fort de G. Si M est un module `premier`*, ou si le père de M dans $\mathcal{T}_G$ est un module* `série`*,* `parallèle` *ou* `ordre`*, alors il existe un nœud de $\mathcal{T}_\sigma$ qui le représente.*

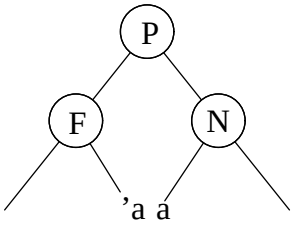
Démonstration: Le paragraphe précédent fait apparaître les deux cas où M sera représenté :

- il existe une fracture égale à M ;
- la racine de $\mathcal{T}_{\sigma(G[M])}$ n'a qu'un fils.

Le théorème est la conséquence des deux lemmes suivants :

Lemme 54 *Soit N un nœud de $\mathcal{T}_G$ et P son père. Si N et P sont dégénérés (de type* `série`*,* `parallèle` *ou* `ordre`*) alors pour toute permutation factorisante σ de G il y a une fracture égale à N.*

Démonstration: Rappelons que dans un arbre de décomposition modulaire il n'y a pas de nœud d'arité un, donc que P a un autre fils. Forcément un fils de P est à côté de N dans σ. Supposons qu'il y ait un module fort F fils de P *à gauche* de F dans σ (la preuve serait la même pour un fils à droite, en prenant le symétrique de σ). Soit a le bord gauche de N ; $'a$ est donc le bord droit de F.



Une propriété connue de $\mathcal{T}_G$ est que les fils d'un nœud dégénéré n'ont pas le même type que leur père. Soit $x \neq a$ un descendant de N. $\mathrm{LCA}(a, x) = N$ et $LCA('a, x) = P$. Donc, d'après la propriété 17 page 51, x sépare a et $'a$. Soit y un sommet à droite de a qui ne descend pas de N. Si $y \in P$ alors $\mathrm{LCA}('a, y) = LCA(a, y) = P$ donc y ne sépare pas a et a'. Et si $y \notin P$ alors d'après la proposition 67 y ne sépare pas non plus a de a' qui sont dans le même module. Donc la fracture droite de $\{'a, a\}$ est exactement N.

□

Il reste le cas des nœuds `premier` : nous devons démontrer que la racine de $\mathcal{T}_{\sigma(G[M])}$ n'a qu'un fils si M est un module `premier` de G. Le lemme 53 permet de se ramener au cas des graphes premiers, puisqu'on n'aura qu'à substituer les fils de M dans $\mathcal{T}_G[M]$ ensuite. On déduit immédiatement du lemme suivant, et de la façon dont l'algorithme 12 page 202 construit $\mathcal{T}_\sigma$, que la racine de $\mathcal{T}_{\sigma(G[M])}$ n'a qu'un fils.

Lemme 55 *Soit G un graphe premier, $\hat{\sigma}$ une de ses permutations factorisantes parenthésée. La hauteur de Dyck n'est jamais nulle à l'intérieur de $\hat{\sigma}$.*

Démonstration: Rappelons que par construction de $\hat{\sigma}$ entre deux sommets on place d'abord toutes les parenthèses fermantes puis toutes les ouvrantes. Supposons qu'entre deux sommets x et x' de $\hat{\sigma}$ la hauteur de Dyck s'annule.

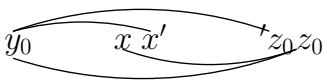
On place toujours la parenthèse ouvrante d'une fracture avant sa parenthèse fermante. C'est pourquoi $[xx']$ ne saurait être facteur d'aucune fracture, sans quoi la hauteur de Dyck entre x et x' serait au moins de 1.

Appelons T le type de relation entre x et x'. T peut être un arc vers la gauche (de x à x'), un arc vers la droite, un double arc ou pas d'arc.

- Supposons que $'x$ existe. Le lien entre $'x$ et x' est T, sans quoi x' distinguerait $\{'x, x\}$, donc $[xx']$ serait facteur de la fracture droite de $\{'x, x\}$, ce qu'on vient d'interdire.
- De même si $''x$ existe le lien entre $''x$ et x' est T sans quoi x' distinguerait $\{''x,' x\}$,donc $[xx']$ serait facteur de la fracture droite de $\{''x,' x\}$.
- En itérant il vient
$$\forall y \prec x' \;\; yTx'$$
- La même démonstration montre
$$\forall x \prec z \;\; xTz$$
- Nous allons maintenant prouver
$$\forall y, z \;\; (y \prec x \prec x' \prec z) \Rightarrow \; yTz$$

  Démontrons-le par contradiction. Supposons qu'il existe un sommet y_0 à gauche de x qui n'entretient pas le lien T avec tous les sommets à droite de x'. Soit z_0 le plus à gauche des sommets à droite de x' tels qu'on ait pas y_0Tz_0. Puisqu'on a au moins y_0Tx', on a $y_0T'z_0$. Donc y_0 sépare la paire $\{'z_0, z_0\}$, et donc $[xx']$ est un facteur de la fracture gauche de cette même paire, contradiction.

σ :



Mais puisque $\forall x \prec z \;\; xTz$, selon le type de T (double arc, simple arc ou non-arc), G est une composition série, ordre ou parallèle des ensembles $\{\sigma^{-1}(1), \ldots x\}$ et $\{x', \ldots \sigma^{-1}(n)\}$ et n'est donc pas premier : contradiction.

□

Ces deux lemmes prouvent le théorème.

□

Ce théorème laisse apparaître un seul trou : des nœuds dégénérés, qui sont fils dans $\mathcal{T}_G$ d'un nœud `premier`, peuvent manquer dans $\mathcal{T}_\sigma$. Nous appellerons **fusion** un tel cas. Soit N un nœud dégénéré, P son père `premier` et $F_1 \ldots F_k$ ses fils. P est un nœud de $\mathcal{T}_\sigma$ [1]. La preuve du lemme 53 fait apparaître que soit N est aussi un nœud de $\mathcal{T}_\sigma$, et tout va bien, soit pas, mais dans ce cas $F_1 \ldots F_k$ sont des fils de P.

Les fusions arrivent effectivement : ainsi dans notre exemple page 201, le module fort `ordre` $\{6, 7, 8\}$ est fusionné au module fort `premier` $\{6, \ldots 14\}$, et il n'y a pas de nœud pour le représenter. Heureusement, les fusions ne concernent jamais que la fusion d'*un* nœud avec son père ; il

1. Dans le cas d'une famille arborescente, on identifie sans ambiguïtés les nœuds de l'arbre et les éléments de la famille. Ici, l'existence de nœuds à un seul fils amène à commettre un petit abus...

n'y a pas de fusions cascadées. Cela va nous simplifier considérablement la tâche pour retrouver ces fusions.

Définition 40
Deux sommets x et y sont demi-jumeaux si ils sont unis par un arc simple et si tout sommet distinguant $\{x, y\}$ est uni à x et à y par un arc simple. On peut aussi dire que $\{x, y\}$ est un module de la 2-structure $H = (V, E_H)$ où $E_H(u, v)$ compte le nombre d'arcs entre u et v (0, 1 ou 2).

Soit J_0 (resp. $J_{1/2}$, J_1) la relation de $V \times V$ telle que xJ_0y (resp. $xJ_{1/2}y$, xJ_1y) si x et y sont faux jumeaux (resp. demi-jumeaux, vrais jumeaux). Ces trois relations sont des équivalences. Une classe d'équivalence non-triviale (possédant plus d'un élément) de J_0, $J_{1/2}$ ou J_1 est un **bloc de jumeaux** . On montre facilement que les blocs de jumeaux sont d'intersection vide ; de plus étant donné un graphe G et une permutation factorisante σ de G, les blocs de jumeaux sont des facteurs de σ. Les blocs de vrais jumeaux induisent des cliques, ceux de faux jumeaux des stables, et ceus de demi-jumeaux des tournois. Utilisons ce concept pour la caractérisation des fusions.

Définition 41
*Soit $\mathcal{T}$ un arbre, σ une permutation factorisante de cet arbre, N un nœud de cet arbre et $F_1 \dots F_k$ les fils de N dans $\mathcal{T}$. Une **permutation factorisante quotiente** de N, σ_N, est le sous-mot obtenu en gardant dans σ exactement un sommet par fils F_i (il est donc de longueur k).*

Lemme 56 *Soit N un nœud* `premier` *de $\mathcal{T}_\sigma$ qui est un module et dont tous les fils $F_1 \dots F_k$ sont des modules. Soit F un module dégénéré ayant fusionné avec N.*

$$\exists 1 \leqslant a < b \leqslant k \; tq. \; F = \bigcup_{i=a}^{i=b} F_i$$

$[f_a \dots f_b]$, $f_a \in F_a$, $f_b \in F_b$, est un bloc de jumeaux, facteur de la permutation quotiente de N (cela implique que $b - a$ est maximum).

Démonstration: Puisque F est un module fort, c'est un facteur de $\sigma(G)$. Le théorème 41 assure que les fils de F dans $\mathcal{T}_G$ sont représentés dans $\mathcal{T}_\sigma$. Puisque par hypothèse N et tous ses fils sont des modules, les fils de F (dans $\mathcal{T}_G$) sont des fils de N dans $\mathcal{T}_\sigma$. Puisque F est un module fort, c'est un facteur de $\sigma(G)$. Les fils de F sont donc *consécutifs* dans $\mathcal{T}_\sigma$: ce sont $F_a, \dots F_b$.

$\sigma(G_N)$ étant compatible avec $\sigma(G)$, $[f_a \dots f_b]$ est un facteur de $\sigma(G_N)$.

Si F est un module `série` (resp. `parallèle`), il est clair que $\forall i, j \in [a, \dots b]$, f_i et f_j sont vrais (resp. faux) jumeaux de G_N, puisque J_0 et J_1 sont des équivalences.

Montrons que cela est vrai aussi si F est un module `ordre`. Supposons pour contradictions que f_i et f_j ne soient pas demi-jumeaux. Il existe nécessairement l compris entre i et j tel que f_l et f_{l+1} ne soient pas demi-jumeaux de G_N. En nous ramenant à G, cela veut dire que $\mathrm{Sep}(\{bd(F_l), bg(F_{l+1})\})$ contient des sommets en-dehors de $F_l \cup F_{l+1}$. Cette paire génère

donc une fracture. Lors de la construction de $\mathcal{T}_\sigma$, les parenthèses de cette fracture font que F_l et F_{l+1} sont séparés et ne descendent plus du même nœud. C'est une contradiction. Donc tous les sommets représentatifs correspondant à des modules `ordre` fusionnés sont demi-jumeaux (on en déduit qu'ils suivent l'ordre total de F).

□

8.5 Nœuds supplémentaires : les artefacts

Quels peuvent bien être les nœuds *en trop* de $\mathcal{T}_\sigma$? Tout nœud de $\mathcal{T}_G$ représente un module fort de G, et réciproquement tout module fort de G est représenté par un seul nœud de $\mathcal{T}_G$. Dans $\mathcal{T}_\sigma$ ces propriétés ne sont pas forcément respectées : caractérisons les nœuds qui les violent.

Définition 42
*Un **artefact** est un nœud de $\mathcal{T}_\sigma$ qui soit*

1. *n'a qu'un seul fils*
2. *n'est pas un module de G*
3. *est un module de G, mais pas un module fort.*

*Un nœud qui n'est pas un artefact est **authentique**.*

On peut supprimer les artefacts de $\mathcal{T}_G$ en les **compactant**, c'est-à-dire en rattachant les fils de l'artefact directement au père de l'artefact. Ainsi on n'aura plus que les nœuds authentiques de $\mathcal{T}_\sigma$. Or le théorème 41 dit que l'arbre des nœuds authentiques est *presque* $\mathcal{T}_G$: il ne manque que de rares nœuds dégénérés.

Comment retrouver les artefacts ? Les artefact de première espèce (ceux répondant à l'axiome 1 de la définition, qui n'ont qu'un seul fils) sont triviaux à détecter. Notons au passage que tous les nœuds représentant le même module fort forment une chaîne de nœuds d'arité 1, sauf le dernier de la chaîne. C'est justement ce dernier qui est authentique, tandis que tous les autres sont des artefacts de première espèce. Les artefacts de deuxième espèces peuvent être détectés grâce à l'algorithme de §5.4.3, en voyant si les séparateurs extrêmes d'un nœud sont égaux aux bords de ce nœud.

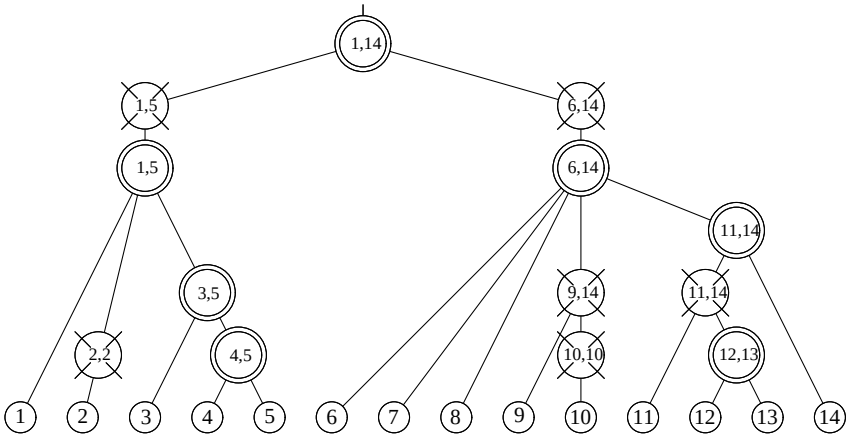
Enfin, intéressons-nous aux artefacts de la troisième espèce. Ils sont faciles à détecter à cause de la propriété suivante :

Proposition 68
Soit $\mathcal{T}$ un arbre ne possédant que des nœuds authentiques et des artefacts de troisième espèce. On peut alors typer *un nœud en* `série`, `parallèle`, `ordre` *ou* `premier`, *selon que le quotient*[2] *est une clique, un stable, un ordre total ou autre chose.*

Un nœud N de $\mathcal{T}$ est un artefact de troisième espèce si et seulement si c'est un nœud dégénéré du même type que son père.

2. Le quotient d'un nœud est bien défini puisque ce nœud est un module.

Dans notre algorithme en six passe, au début de la sixième passe on possède effectivement un arbre possédant cette propriété, ce qui permet de supprimer les artefacts de troisième espèce.



L'arbre des fractures de l'exemple page 200. A l'intérieur de chaque nœud sont marqués ses premiers et derniers séparateurs. Les artefacts de première et deuxième espèce sont barrés. C'est en fait l'arbre obtenu après la troisième passe de l'algorithme.

Remarquons toutefois qu'on a peu de chances de tomber sur des artefacts de troisième espèce. En effet

Proposition 69
$\mathcal{T}_\sigma$ n'a pas d'artefact de troisième espèce de type `série` *ou* `parallèle`*. De plus, si dans σ tous les modules forts fils d'un module fort* `ordre` *arrivent dans le bon ordre, alors $\mathcal{T}_\sigma$ n'a pas d'artefact de troisième espèce de type* `ordre` *non plus.*

Nous ne fournissons pas de preuve à cette proposition (qui n'est pourtant pas immédiate) car elle n'est pas nécessaire au fonctionnement de l'algorithme. Cependant, elle permet d'éviter la sixième passe de l'algorithme dans tous les cas concrets :

1° Quand on décompose des graphes non-orientés

2° Quand on décompose des graphes orientés avec un algorithme qui donne les fils d'un module `ordre` dans le bon ordre, tel que celui du chapitre 7.

Il est donc raisonnable de supposer cela. Mais après tout la sixième passe est très facile à programmer et est rapide...

8.6 Implémentation séquentielle

Le but de cette section est de prouver :

Théorème 42
Étant donnée une permutation factorisante d'un graphe G (orienté ou non), on peut calculer l'arbre de décomposition modulaire de G en temps $O(n+m)$.

Étant donnée une permutation factorisante σ d'un graphe G (orienté ou non) et deux fonctions $ps(x, x')$ et $ds(x, x')$ renvoyant respectivement le premier et le dernier sommet séparant $\{x, x'\}$ de σ, calculables en $O(1)$, on peut calculer l'arbre de décomposition modulaire de G en temps $O(n)$.

Les objets et propriétés définis ci-dessus vont nous permettre d'écrire un algorithme linéaire en temps pour retrouver $\mathcal{T}_G$, étant donné σ. La première passe de l'algorithme travaille en $O(n + m)$ ou $O(n)$ selon le cas, et les suivantes en $O(n)$, ce qui donne bien un algorithme de la complexité annoncée. La première passe construit le parenthésage de fracture. La deuxième passe construit $\mathcal{T}_\sigma$. La troisième détecte les artefacts de première et de deuxième espèce. La quatrième les supprime. La cinquième détecte les fusions et les corrige, tout en typant les nœuds. Enfin la sixième détecte et supprime les artefacts de troisième espèce.

8.6.1 Première étape : parenthésage de fracture

Stocker la permutation factorisante parenthésée $\hat{\sigma}$ se fait sans difficulté en utilisant deux tableaux `PO[ ]` et `PF[ ]`, de taille $n - 1$, où `PO`[i] compte le nombre de parenthèses ouvrantes entre $\sigma^{-1}(i)$ et $\sigma^{-1}(i + 1)$ tandis que `PF`[i] compte les parenthèses fermantes.

Grâce à l'algorithme de §5.4.2, on peut calculer $\mathrm{Sep}(\{x, x'\})$. Il ne reste plus qu'à extraire le plus grand et le plus petit élément pour construire les fractures. Le voisinage de chaque sommet x est regardé au plus deux fois, une fois pour la paire dont x est le membre droit et une fois pour la paire dont il est membre gauche. Or, calculer $\mathrm{Sep}(\{x, x'\})$ puis extraire $ps(x, x')$ et $ds(x, x')$ prend $O(|\Gamma(x)| + |\Gamma(x')|)$. Cette première étape de l'algorithme tourne donc en $O(n + m)$, en n'utilisant que des algorithmes de tri et des manipulations simples de listes.

Si de plus $ps(x, x')$ et $ds(x, x')$ peuvent se calculer en $O(1)$ alors cette passe se fait en temps $O(n)$.

8.6.2 Deuxième étape : construction de l'arbre des fractures

Passer de $\hat{\sigma}$ à $\mathcal{T}_\sigma$ se fait grâce à l'algorithme 12, dont personne ne doutera qu'il soit implémentable en $O(n)$ (qui est la taille de l'entrée et de la sortie). Il n'y a pas plus de $2n-2$ fractures, donc $\mathcal{T}_\sigma$ possède $O(n)$ nœuds.

8.6.3 Troisième étape : détection des artefacts de première et deuxième espèce

Il est facile de détecter les artefacts de première espèce dans $\mathcal{T}_\sigma$: ce sont les nœuds à un seul fils. Les artefacts de la deuxième espèce peuvent être détectés en $O(n+m)$, grâce à l'algorithme de §5.4.3. Cet algorithme peut même être optimisé jusqu'à $O(n)$: comme la première étape calcule les premiers et derniers séparateurs de chaque paire, on peut conserver cette information dans deux tableaux afin de la retrouver en $O(1)$. Ensuite, un parcours en profondeur de $\mathcal{T}_\sigma$ permet

d'obtenir pour chaque nœud N ses bords gauche et droits, ainsi que ses premiers et derniers séparateurs en utilisant les relations de récurrence (5.4) et (5.5) page 143, en $O(n)$. Cette donnée permet, via la proposition 64, de détecter les artefacts de deuxième espèce.

8.6.4 Quatrième étape : suppression de ces artefacts

Cette étape transforme $\mathcal{T}_\sigma$ en un nouvel arbre $\mathcal{T}'_\sigma$ en enlevant tous les artefacts. Cette suppression se fait en éliminant chaque artefact N, c'est-à-dire en attribuant tous les fils de N au père de N puis en effaçant N de l'arbre. Il faut veiller à respecter l'ordre des fils et à les insérer tous consécutivement à la place de N, afin que la permutation factorisante σ corresponde au parcours de $\mathcal{T}'_\sigma$ (comme elle correspondait au parcours de $\mathcal{T}_\sigma$). Ces opérations peuvent facilement être réalisée en un temps $O(n)$. $\mathcal{T}'_\sigma$ est un arbre où tous les nœuds représentent des modules, et où tous les modules forts sont représentés, *sauf* certains modules dégénérés descendant dans $\mathcal{T}_G$ d'un nœud `premier`.

8.6.5 Cinquième étape : détection des fusions

Tous les nœuds de l'arbre $\mathcal{T}'_\sigma$ sont maintenant des modules, donc vérifient les condition du lemme 56. En le lisant attentivement, on constate qu'il n'est pas nécessaire de calculer les graphes quotients G_N pour chaque nœud N, mais qu'il suffit d'extraire leurs permutations factorisantes quotientes, ce qui est facile (*cf.* définition 41).

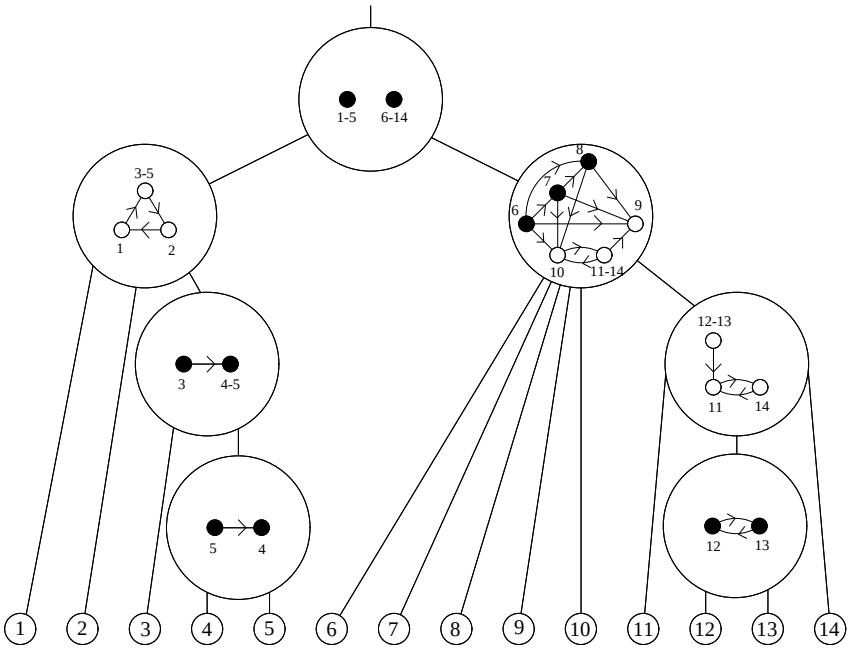
Possédant cette permutation quotiente $\sigma(G_N)$, il faut calculer les blocs de jumeaux. Ils sont facteurs de $\sigma(G_N)$. Comment savoir si f_i et f_{i+1} sont jumeaux dans G_N, sans avoir à calculer G_N ?

Proposition 70
f_i et f_{i+1} sont jumeaux dans G_N si et seulement si

$$ps(\{bd(F_i), bg(F_{i+1})\}) \succcurlyeq bg(F_i) \tag{8.1}$$
$$\text{et } ds(\{bd(F_i), bg(F_{i+1})\}) \preccurlyeq bd(F_{i+1}) \tag{8.2}$$

Détecter les fusions se fait donc en regardant si les équations (8.1) et (8.2) sont vérifiées pour toutes les paires de bords adjacents de fils de N. Un nouveau nœud F est créé pour chaque bloc détecté, et les fils de N correspondant à ce bloc sont attribués à F.

Exemple Reprenons la graphe de la page 200. Voici l'arbre $\mathcal{T}'_\sigma$ correspondant. À l'intérieur de chaque nœud son graphe quotient est dessiné. Les sommets jumeaux sont en noirs.



Considérons le module $N = \{6, \dots 14\}$. Une permutation factorisante quotiente de G_N est $(6, 7, 8, 9, 10, 11)$, en prenant 11 comme sommet représentant du module $\{11, \dots 14\}$. On a $ps(\{6, 7\}) = 6$ et $ds(\{6, 7\}) = 7$, donc 6 et 7 sont jumeaux dans G_N : on a détecté une fusion. On continue le long de $\sigma(G_N)$ pour trouver le bloc de jumeaux auquel ils appartiennent. 7 et 8 sont jumeaux, par contre 8 et 9 ne sont pas jumeaux car $ds(\{8, 9\}) = 14 \not\leq 9$. Le bloc de jumeaux est donc $[6, 7, 8]$. Il trahit la fusion des trois fils $\{6\}$, $\{7\}$ et $\{8\}$ d'un module dégénéré perdu (un `ordre` en l'occurrence) dans N. Un nouveau nœud doit être créé, avec ces trois nœuds comme fils.

Typage des nœuds Le calcul des blocs de jumeaux donne également le type du nœud correspondant. Si tous les fils d'un nœud appartiennent au même bloc du jumeaux, alors ce nœud correspond à un module dégénéré, et sinon à un module `premier`. Dans le cas d'un module dégénéré, il n'y a qu'à tester la relation entre les deux premiers sommets représentatifs pour savoir si ce nœud est `série`, `ordre` ou `parallèle`.

Implémentation et analyse de complexité Le calcul des blocs de jumeaux se fait en regardant les premiers et derniers séparateurs de chaque paire de sommets consécutifs, qui ont été déjà calculés lors de la première passe et donc peuvent être retrouvés en $O(1)$ grâce à un tableau. Une paire $\{x, x'\}$ est regardée exactement une fois, lors de l'examen du nœud $\mathrm{LCA}(x, x')$. La complexité de cette passe est donc $O(n)$.

8.6.6 Sixième étape : suppression des artefacts de troisième espèce

L'arbre $\mathcal{T}''_\sigma$ obtenu à l'issue de la cinquième étape est *presque* l'arbre de décomposition modulaire. En effet, tous les modules forts sont représentés par un nœud, et tous les nœuds représentent un module. Cependant, certains nœuds représentent un module qui n'est pas fort : ce sont les artefacts de troisième espèce. On peut les détecter grâce à la proposition 68, et les supprimer avec la technique de la quatrième étape. Dans l'exemple page 200 qui nous a accompagné tout au long de la description de l'algorithme, il y a un artefact de troisième espèce : le module `ordre` $\{4, 5\}$ est faible et doit être fusionné avec son père, le module `ordre` (fort) $\{3, 4, 5\}$.

La proposition 69 exprime que dans les cas pratiques cela n'arrive pas, et que la sixième étape de l'algorithme peut être omise. Cela revient à affirmer $\mathcal{T}''_\sigma = \mathcal{T}_G$.

La complexité de cette passe est clairement $O(n)$.

8.7 Implémentation parallèle

Théorème 43
Étant donnée une permutation factorisante d'un graphe G, on peut calculer l'arbre de décomposition modulaire d'un graphe (orienté ou non) en temps $O(n)$ sur une machine parallèle à $O(n)$ processeurs.

Démonstration: On utilise une première phase parallèle. Un processeur est assigné à chaque couple $\{x, x'\}$ de sommets consécutifs. On a vu que le calcul de $ps(x, x')$ et $ds(x, x')$ peut être fait par un parcours simultané des deux listes d'adjacences des sommets, en $O(n)$. Ce résultat est stocké dans un tableau, d'où il peut être retrouvé en $O(1)$. On peut donc appliquer le deuxième paragraphe du théorème 42 : la deuxième phase de l'algorithme est séquentielle.

□

8.8 Décomposition en $O(n)$ des graphes d'intervalles

Une classe de graphe parmi les plus connues est la suivante :

Définition 43
*Un **intervalle** $I = [x, y] \subset \mathbb{R}$ est l'ensemble $\{z \mid x \leqslant y \leqslant z\}$. x est la **borne gauche** de l'intervalle noté $g(I)$ et y sa* borne droite *notée $d(I)$.*
*Un **modèle d'intervalles**, est un ensemble V de n intervalles.*

*Un **modèle d'intervalles normé** est un ensemble V de n intervalles dont les bornes sont des entiers tous différents compris entre 1 et $2n$ (inclus).*

*Le **graphe d'intervalles** d'un modèle d'intervalles V est $G = (V, E)$ et $([x, y], [x', y']) \in E$ ssi $[x, y] \cap [x', y'] \neq \emptyset$. On dit que le modèle d'intervalles est un **réaliseur** du graphe.*

Un modèle d'intervalles est un objet non-manipulable efficacement, en particulier parce qu'un « vrai » nombre réel n'est pas manipulable par une machine à mémoire finie. Heureusement

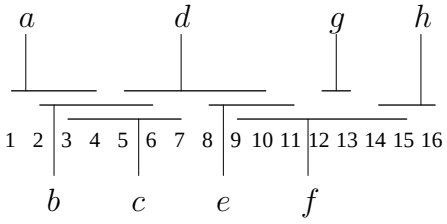
Proposition 71
Étant donné un modèle d'intervalles V il existe une fonction $\phi : \mathbb{R}^2 \mapsto [\![1, 2n]\!]^2$ tels que $\phi(V)$ est un modèle d'intervalles normé ayant le même graphe d'intervalles (en renommant $\phi(v)$ un sommet v).

Démonstration: V contient n intervalles donc au plus $2n$ bornes. On construit un premier modèle équivalent (ayant le même graphe) que le précédent et où deux intervalles ne partagent pas la même borne. Pour cela, on détecte (facilement) les bornes communes puis on ajoute à chaque borne droite b une valeur ϵ_b, et on retire à chaque borne gauche g une valeur ϵ_g, de sortes que deux bornes ne soient plus égales mais qu'aucune intersection ne soit créée ou détruite. Ces nouvelles bornes sont totalement ordonnées : il existe une permutation $\sigma : \mathbb{R} \mapsto [\![1, 2n]\!]$ telle que que $x < y$ ssi $\sigma(x) \leqslant \sigma(y)$. $\phi : [x, y] \to [\sigma(x), \sigma(y)]$ est la fonction recherchée, puisque $[x, y] \cap [x', y'] \neq \emptyset$
ssi $(x \leqslant x' \leqslant y \leqslant y' \vee x' \leqslant x \leqslant y' \leqslant y)$
ssi $(\sigma(x) \leqslant \sigma(x') \leqslant \sigma(y) \leqslant \sigma(y') \vee \sigma(x') \leqslant \sigma(x) \leqslant \sigma(y') \leqslant \sigma(y)$
ssi $[\sigma(x), \sigma(y)] \cap [\sigma(x'), \sigma(y')] \neq \emptyset$

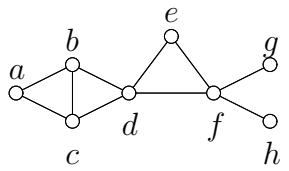
□

$a = (1, 4)$ $b = (2, 6)$
$c = (3, 7)$ $d = (5, 10)$
$e = (8, 11)$ $f = (9, 15)$
$g = (12, 13)$ $h = (14, 16)$

Modèle d'intervalles normé



Modèle d'intervalles, version graphique



Graphe d'intervalles

L'avantage d'un modèle normé est double : primo, les nombres entiers de l'ordre de grandeur de n sont manipulés en temps unité par notre modèle de machine ; secundo, il existe des algorithmes de tri linéaires de $2n$ nombres de l'ordre de n [CLR90], donc on peut supposer qu'on dispose d'une structure de donnée stockant V qui donne la $i$ème borne (selon l'ordre usuel $<$) en temps $O(1)$. Dans toute la suite, le modèle d'intervalles sera supposé normé.

$[x, y]$ est *à gauche* de $[z, t]$ si $y < z$ et *à droite* si $t < x$. Sinon, les deux intervalles se coupent. Ils sont adjacents dans G ssi ils se coupent. De plus, les sommets de $\overline{G}$ sont ordonnés par la relation « être à gauche de » : le complémentaire d'un graphe d'intervalles est un graphe de comparabilité.

Considérons la permutation σ_g de V obtenue en rangeant les intervalles selon l'ordre de leurs bords gauches, et σ_d en les rangeant selon leur bords droits[3]. Malheureusement ni σ_g

3. On peut vérifier (nous ne le prouvons pas car nous ne l'utilisons pas) que si deux modèles d'intervalles produisent les même couples de permutation alors leurs graphes d'intervalles sont isomorphes. Un modèle d'intervalles

ni σ_d ne sont en général des permutations factorisantes, comme le prouve l'exemple suivant : $\overline{a \ \overline{\quad b \quad} \ c}$, où $\sigma_g = \sigma_d = (a, b, c)$ ne sont pas factorisantes. Ce qui nous amène à définir

Définition 44
*Un modèle d'intervalles normé est **factorisant** si pour tout module fort non-trivial M du graphe d'intervalles, pour tous $x \in M$ et $y \notin M$, les intervalles x et y ne se chevauchent pas.*

Donc, dans un tel modèle, les bords gauches (resp. droits) d'un même module fort ne sont entrecoupés par aucun bord gauche (resp. droit) d'un intervalle extérieur au module, et donc σ_g et σ_d sont des permutations factorisantes.

Théorème 44
Étant donné un modèle d'intervalles normé V on peut construire un modèle d'intervalles normé factorisant, ayant le même graphe d'intervalles, en temps $O(n)$.

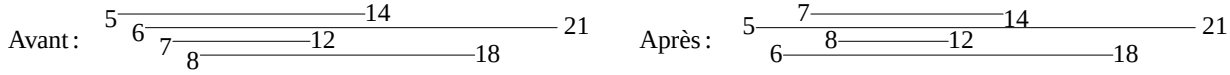
Démonstration: Soit $M \subset V$ un module fort non-trivial de G. Un intervalle $x \notin M$ est soit à gauche de tous les sommets de G, soit à droite de tous, soit il est voisin de tous. Si x contient tous les intervalles de M il n'y a pas de problème : dans une permutation des bornes gauches (resp. droite), la sienne vient avant (resp. après) toutes celles de M. Le problème est que x peut chevaucher des intervalles de M.

Supposons qu'il existe $m \in M$ tel que $g(m) < g(x)$. Un intervalle z vérifiant $g(m) < d(z) < g(x)$ distinguerait M. Si x possède un voisin $y \notin M$, alors y est à droite de M. Donc pour tout $m' \in M$ $d(m') < d(x)$. Et sinon $M = \Gamma(x)$. $M \cup \{x\}$ est un module. Or M est fort, donc $G[M]$ n'est pas une clique. Il existe donc un sommet $m' \in M$ non-adjacent à m. m' ne peut être à gauche de m : il serait à gauche de x donc non-voisin de x. On a alors $d(x) > g(m')$ donc $d(m) < d(x)$.

On prouve de même que s'il existe $m \in M$ tel que $d(x) < d(m)$ alors $g(x) < g(m)$ et il n'y a aucune borne droite entre $d(x)$ et $d(m)$.

Deux intervalles sont **gauche-équivalents** si nulle borne droite n'est intercalée entre leurs bornes gauches, et **droite-équivalents** si nulle borne gauche ne s'intercale entre leurs bornes droites. Deux sommets sont vrais jumeaux si et seulement si ils sont gauche-équivalents et droite-équivalents.

Les sommets gauche-équivalents sont ordonnés par leurs bornes droites. Il existe une façon, unique, de renuméroter les bornes gauches de sorte que les intervalles soit tous inclus les uns dans les autres. Pour toute classe de gauche-équivalence de k intervalles, il existe a tel que les bornes gauches soient dans $[\![a, a + k - 1]\!]$. On renumérote a la borne gauche de l'intervalle maximisant $d(I)$, $a + 1$ la suivante, etc.



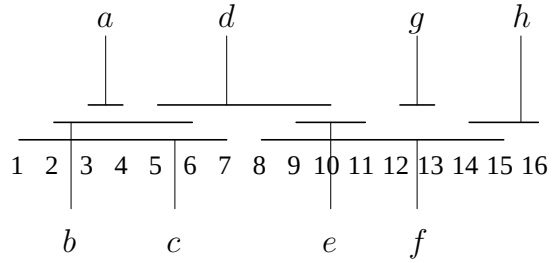
tel $\sigma_g = \sigma_d$ correspond à un graphe d'intervalles *propre* : aucun intervalle n'est inclus dans un autre. Les graphes d'intervalles propres sont exactement les graphes d'intervalles *unitaires* : ils possède un modèle où tout intervalle est de la forme $[x, x + 1]$.

Après avoir trié de la sorte les intervalles gauche-équivalents et droite-équivalents, on obtient un modèle de permutation de même graphe (aucune intersection d'intervalles n'est créée ou supprimée) et normé (il possède les mêmes bornes). Il est de plus factorisant, car d'après ce qui précède nul intervalle n'appartenant pas à un module fort M ne peut chevaucher un intervalle de M.

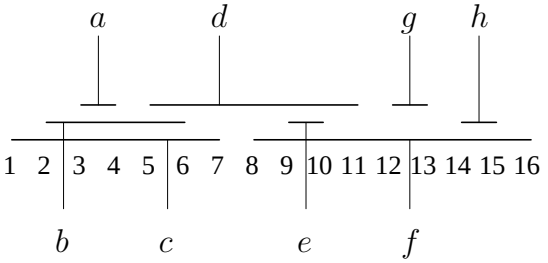
Il est trivial de trouver les gauche-équivalents et les droite-équivalents, si on peut énumérer toutes les bornes dans l'ordre. Cela fournit de plus les constantes a de chaque classe. Chaque élément d'une classe est marquée par un identifiant unique de classe. Ensuite on trie les intervalles selon leurs bornes droites. On peut alors renuméroter leurs bords gauches en parcourant toute la liste, et en conservant pour chaque classe (dans un tableau de taille $O(n)$) la valeur de a incrément à chaque mise à jour d'un bord gauche. Cet algorithme prend $O(n)$. Ensuite, on renumérote les bords droits de la même façon.

□

Dans l'exemple page 213, les classes de sommets gauche-équivalents sont $\{a, b, c\}$, $\{d\}$, $\{e, f\}$, $\{g\}$ et $\{h\}$ et celles de sommets droite-équivalents $\{a\}$, $\{b, c\}$, $\{d, e\}$, $\{g\}$ et $\{f, h\}$. $\sigma_g = (c, b, a, d, f, e, g, h)$ et $\sigma_d = (a, b, c, e, d, g, h, f)$ sont factorisantes.



Après la renumérotation des bords gauches



Après la renumérotation des bords droites

Supposons donc, quitte à appliquer l'algorithme précédent, que le modèle d'intervalles est factorisant. À ce point on possède donc deux permutations factorisantes, celle des bords gauches et celles des bords droits. Considérons, par exemple, celle des bords gauches.

Considérons deux intervalles $u = [x, y]$ et $u' = [x', y']$. On suppose sans perte de généralité $x < y$. $v = [z, t]$ distingue $\{u, u'\}$ ssi

- soit $x \leqslant t < x'$: v est voisin de u et à gauche de v ;
- soit $y \leqslant z < y'$: v est à droite de u et voisin de v.

Proposition 72
Le premier séparateur *(selon σ_g) de $\{u, u'\}$ est le sommet qui minimise $g(I)$ pour tout I tel que $x < d(I) < x'$, et le* dernier séparateur *(selon σ_g) de $\{u, u'\}$ est le sommet qui maximise $g(I)$ pour tout I tel que $y < g(I) < y'$.*

Soit f la fonction $[\![1, 2n]\!] \mapsto \mathbb{N}$ telle que pour tout intervalle $[x, y]$ du modèle, $f(x) = +\infty$ et $f(y) = x$ (c'est en quelque sorte la fonction « *bord gauche associé* »). On rend f injective bornée par $3n + 1$ en interprétant $+\infty$ comme représentant des nombres tous distincts et compris

entre $2n+1$ et $3n+1$. On peut donc appliquer les outils de §5.4.4. La requête MinInt (x, x') produit le plus petit bord gauche d'un intervalle ayant sont bord droit entre x et x', et calcule donc $ps([x,y],[x',y'])$. D'après le théorème 32 page 146, il se calcule en $O(1)$, après une phase de pré-traitement en $O(n)$ (construction du tarbre croissant et marquage pour le plus petit ancêtre commun).

De même pour trouver $dx(u,u')$ on utilise un tarbre décroissant pour la fonction f qui, pour tout intervalle $[x,y]$ du modèle, vaut $f(x)=y$ et $f(y)=-\infty$. La requête MaxInt (x,x') produit le plus grand bord droit d'un intervalle ayant sont bord gauche entre x et x'. Donc :

Théorème 45
Étant donné un modèle d'intervalles normé, on peut calculer en temps $O(n)$ l'arbre de décomposition modulaire du graphe correspondant.

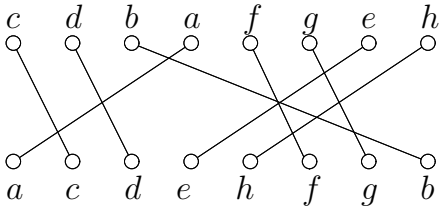
8.9 Décomposition en $O(n)$ des graphes de permutation

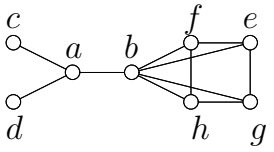
Introduisons une autre classe de graphe largement utilisée :

Définition 45
*Un **modèle de permutation** est la donnée de deux permutations σ_1 et σ_2 d'un ensemble V.*
*Le **graphe de permutation** est le graphe de sommets V et où (x,y) est une arête ssi ($\sigma_1(x) < \sigma_1(y)$ et $\sigma_2(x) > \sigma_2(y)$), ou ($\sigma_1(x) > \sigma_1(y)$ et $\sigma_2(x) < \sigma_2(y)$). On dit que le modèle de permutation est un **réaliseur** du graphe.*

Les graphes de permutation sont les graphes d'intersection de segments reliant deux points situés sur des droites différentes parallèles, comme le montre le dessin suivant. Un graphe de permutation est le graphe d'incomparabilité de l'ordre partiel [4] défini par $x \prec y$ ssi $\sigma_1(x) < \sigma_1(y)$ et $\sigma_2(x) < \sigma_2(y)$.

$\sigma_1 = (c,d,b,a,f,g,e,h)$
$\sigma_2 = (a,c,d,e,h,f,g,b)$





Modèle de permutation — Le même, forme graphique — Graphe du modèle

4. C'est un ordre de dimension 2 : deux extensions linéaires, σ_1 et σ_2, le définissent sans ambiguïté.

Proposition 73
Chacune des deux permutations d'un modèle de permutation est une permutation factorisante du graphe de ce modèle.

Démonstration: Supposons qu'il existe un module M, $a, b \in M$ et $c \notin M$ avec $\sigma_1(a) \leqslant \sigma_1(c) \leqslant \sigma_1(b)$.

Distinguons deux cas. Le premier est si c n'est pas adjacent à M. Soient $M_g = \{x \in M \mid \sigma_1(x) < \sigma_1(c)\}$ et $M_d = \{x \in M \mid \sigma_1(x) > \sigma_1(c)\}$. Pour tous $x \in M_g$ et $y \in M_d$, comme c ne leur est pas adjacent, on a $\sigma_2(x) < \sigma_2(c) < \sigma_2(y)$. Donc x et y ne sont pas adjacents. Or M_g contient a et M_d contient b. On en déduit que $G[M \cup \{c\}]$ n'est pas connexe et que M n'est pas un module fort, car chevauché par le module $M_g \cup \{c\}$.

Et si G est adjacent à M, on est ramené au cas précédent en remarquant que $\overline{G}$ est le graphe de permutation obtenu en retournant σ_2 ($\sigma_2^r(x) = (n+1) - \sigma_2(x)$).

□

Proposition 74
Soient $x, x' \in V$ tels que $\sigma_1(x) = 1 + \sigma_1(x')$. $y \in V$ distingue $\{x, x'\}$ si et seulement si $\sigma_2(x) < \sigma_2(y) < \sigma_2(x')$ ou $\sigma_2(x') < \sigma_2(y) < \sigma_2(x)$.

Démonstration: $\sigma_1(y)$ est soit supérieur à $\sigma_1(x)$ et $\sigma_1(x')$, soit inférieur aux deux. Supposons le premier cas. Si $\sigma_2(x) < \sigma_2(y)$ alors y est non-adjacent à x, sinon il lui est adjacent. De même y est adjacent à x' ssi $\sigma_2(y) < \sigma_2(x')$. Une séparation signifie une et une seule adjacence ; on en déduit la proposition. Même preuve pour le cas $\sigma_1(y)$ inférieur à $\sigma_1(x)$ et $\sigma_1(x')$.

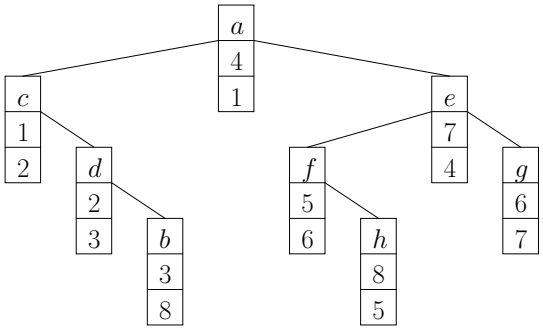
□

Soit I l'intervalle entier comprenant tous les nombres strictement compris entre $\sigma_2(x)$ et $\sigma_2(x')$. On a $Sep(\{x, x'\}) = \sigma_2^{-1}(I)$, $ps(x, x') = \min\{\sigma_2(y) \mid y \in Sep(\{x, x'\})$ et $ds(x, x') = \max\{\sigma_2(y) \mid y \in Sep(\{x, x'\})$.

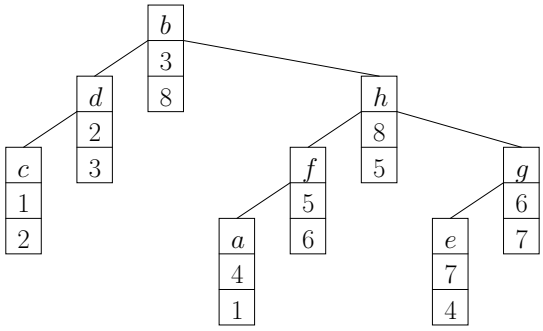
Soit f la fonction telle de $f(\sigma_1(x)) = \sigma_2(x)$ (ou encore $f(x) = \sigma_2(\sigma_1^{-1}(x))$). f est injective de $[\![1, n]\!] \mapsto \mathbb{N}$ et bornée par n. On peut donc appliquer les outils de §5.4.4. $ps(x, x')$ est une requête MaxInt et $ds(x, x')$ une requête MinInt. D'après le théorème 32 page 146, ils se calculent en $O(1)$, après une phase de pré-traitement en $O(n)$ (construction de deux tarbres et marquage pour le plus petit ancêtre commun). Donc :

Théorème 46
Étant donné un modèle de permutation, la décomposition modulaire du graphe de permutation de ce modèle peut se calculer en $O(n)$.

Notons que ce temps est sous-linéaire en la taille du graphe de permutation, pour lequel on peut avoir $m = \Theta(n^2)$.



tarbre croissant de l'exemple



tarbre décroissant de l'exemple

8.10 Conclusion

L'algorithme séquentiel calcule, en un temps $O(n+m)$, l'arbre de décomposition modulaire d'un graphe (orienté ou non) en connaissant une permutation factorisante. Cet algorithme est **simple** : il ne fait appel à aucune structure de données autre que des listes chaînées ou des tableaux, et n'utilise aucun autre algorithme à part un algorithme de tri linéaire stable, qui peut se programmer en quelques lignes, et des parcours d'arbres. Sa preuve n'est somme toute pas très compliquée, et son implémentation est très facile. Elle ne prend que 580 lignes de code C (très commenté et aéré !). Elle est disponible en ligne sur
`http://www.lirmm.fr/~montgolfier/DM`

Le premier algorithme passant des permutations factorisantes aux graphes orientés, écrit par Christian Capelle et Michel Habib [Cap97b, CH97], et qui historiquement est arrivé avant les algorithmes linéaires de production de permutations factorisantes, a démontré que le calcul d'une permutation factorisante est un problème linéairement équivalent à la décomposition modulaire. Il a ouvert la voie à des algorithmes de décomposition en deux temps. Un autre exemple d'algorithme « 2 temps » de ce style est la décomposition en blocs des graphes d'héritage [Cap97a].

Les algorithmes en $O(n)$ de décomposition des graphes d'intervalles et de permutation sont les premiers à atteindre cette borne, ce qui n'est bien sûr possible que si le modèle est connu.

Conclusion

sapientiam scribe in tempore vacuitatis et qui minoratur actu sapientiam percipiet

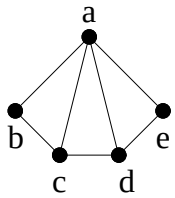
ECCLÉSIASTIQUE 38, 24

À la fin de cette thèse, il m'apparaît que ma compréhension de la décomposition modulaire et des mécanismes afférents a bien augmentée depuis mes débuts hésitants en D.E.A ; compréhension que je me suis efforcé de faire partager au lecteur au long de cet ouvrage. Cette conclusion présente le travail fait, mais aussi celui qui reste à faire...

—

Le premier chapitre a présenté les familles partitives et bipartitives, formalisme unique dans lequel sont rentrées beaucoup de décompositions de graphes présentées dans les chapitres suivants. Le lecteur en a vu bon nombre d'exemples dans ces pages. Il y a sans doute beaucoup d'autres familles partitives ou bipartitive connues, qui nécessiteraient un long travail de bibliographie pour être identifiées sous la plume d'auteurs variés, qui en retrouvent les propriétés sous des formalismes sans doute variés eux aussi, et ignorent peut-être les travaux de [CHM81, CE80, MR84]... Un tel travail de longue haleine n'apporte à priori que peu de résultats nouveaux, mais unifie diverses branches des mathématiques et de l'informatique.

Qui dit décomposition arborescente ne dit pas nécessairement famille partitive. La **treewidth** d'un graphe est une décomposition en arbre très connue ([Bod93] est une très bonne introduction). Il n'est pas immédiat et canonique d'associer une famille partitive à une décomposition de type *treewidth*. Le concept central est celui de *séparateurs*. Mais prenons par exemple le graphe *diamant*, qui est d'ailleurs un graphe d'intervalles et de permutation.



Ses séparateurs sont $\{a, b, d\}$, $\{a, c\}$, $\{a, c, d\}$, $\{a, c, e\}$, $\{a, d\}$: ce n'est pas une famille partitive. Peut-être peut-on cependant extraire une famille partitive d'une décomposition en largeur

(*treewidth, cliquewidth,* etc.)?

—

La connaissance de la décomposition 2-modulaire a bien avancé par-rapport à l'existant, qui se résumait –à ma connaissance– aux travaux de Cunningham sur la décomposition en coupes [Cun82], à l'algorithme de Everett, Klein et Reed [EKR97], et aux travaux de Lanlignel sur la cliquewidth [Lan01]. Ce mémoire présente six familles de 2-modules (modules, sesquimodules, 2-modules boiteux, coupes, co-coupes, 2-joints) qui ont une *structure* partitive ou bipartitive, c'est-à-dire un arbre de décomposition partitif. Sesquimodules et 2-modules boiteux en fait des clones de la décomposition modulaire. La famille nouvelle des 2-joints est bien comprise, avec une caractérisation des graphes complètement décomposables, un algorithme de décomposition et une technique de squelettisation. Qui plus est, les trois familles *parfaites* sont pratiquement d'intersection vide, ce qui fait que sauf cas particulier la famille des 2-modules parfaits d'un graphe est bipartitive (théorème 18 page 83). Ce résultat permet d'affirmer que les 2-modules parfaits sont une puissante généralisation des modules.

Mais ce n'est pas le cas des 2-modules en général. Je montre à quel point la famille des 2-modules échoue de peu à être faiblement partitive (théorème 14). Il est donc frustrant de ne pas pouvoir, au terme de cette thèse, exhiber un arbre de décomposition 2-modulaire. Je pense que ce n'est pas la « non-partitivité » qui est à l'origine de cette absence, car le théorème 14 circonscrit les *conflits* dans un territoire assez restreint, et ils n'empêchent pas l'existence d'un arbre dans le cas biparti. Non, c'est plutôt le *non-passage au quotient*, le fait qu'un 2-module de $G[M]$ puisse *ne pas* être un 2-module de G, quand bien même M est un 2-module ! Cela condamne toute approche récursive.

Puisqu'un 2-module « conflictuel » M possède un sommet M-homogène, une voie d'approche non-récursive est de considérer seulement la famille des 2-modules sans sommet homogène, puis de tenter de rendre partitive la famille en la fermant par des opérations d'union, intersection et différence d'éléments se chevauchant. J'ai échoué dans cette démarche, non par obtention d'un contre-exemple, mais seulement parce que la complexité des preuve (il faut montrer que la famille obtenue ne contient que des 2-modules) devenait rebutante à cause du nombre de cas à traiter. Il reste du travail dans cette voie...

—

Le chapitre 4 est plus abouti, car il présente une théorie autonome est assez complète de la décomposition bimodulaire. Il est frappant de voir à quel points les liens avec la décomposition modulaire des graphes orientés sont nombreux : arbre avec quatre types de nœuds (deux `complet` qui s'échangent par (bi)complémentation, un `linéaire` et un `premier`), substitution, passage au quotient...

Seuls manquent la construction d'un vrai quotient (il faut mettre une arête spéciale), l'hérédité de la primalité, et la différence la plus importante : l'arbre n'est, à la base, pas celui de tous les bimodules mais seulement celui des bimodules *canoniques*, et des « bricolages » sont nécessaires pour qu'il décrive toute la famille. Cela vient de la difficulté créée par les *conflits*,

qui rendent la famille non-partitive : il est rétrospectivement étonnant qu'ils soient, finalement, si peu « nocifs ».

Le travail qui reste à faire dans ce domaine concerne l'amélioration de l'algorithme de décomposition, dont la complexité peut sans doute être abaissée jusqu'à $O(n+m)$ (c'est déjà le cas des graphes bisplit étendus [FGV99]) ou, comme a suggéré J.M. Vanherpe, de s'intéresser à la $2K_2$-connexité, qui pourrait être l'analogue biparti de la P_4-connexité des graphes premiers [BO99].

—

La deuxième partie pose peu de question : elle se contente d'exhiber cinq algorithmes de décomposition modulaires (pour les graphes non-orientés, les tournois, les graphes orientés, les graphes d'intervalles codés par leur modèle et les graphes de permutation codés par leur modèle), tous linéaires en la taille de l'entrée. Le lecteur pourra juger de leur simplicité ; je pense qu'elle n'est pas pire que les algorithmes existants (pour les deux premiers cas : les trois derniers sont nouveaux pour cette borne). La version parallèle des algorithmes des chapitres 6 et 8 permet de décomposer en $O(n)$ un graphe non-orienté, sur une machine à $O(n)$ processeurs ; il est de plus facile de répartir la charge si la machine réelle dispose de moins de processeurs. Les algorithmes de décomposition des graphes d'intervalles et des graphes de permutation sont très simples, et ne nécessitent pas de recherche supplémentaire.

Il serait mieux d'écrire un algorithme de décomposition modulaire des graphes orientés par une méthode « directe », c'est-à-dire par extension d'ordre factorisant. Mais, indépendemment même de la complexité, il semble que les règles d'affinage ne soient pas faciles à définir, car celles du cas non-orienté (chapitre 6) et celles du cas des tournois (§7.1) font exactement l'inverse l'une de l'autre. L'algorithme du chapitre 7 présente « un peu trop » d'appels à d'autres algorithmes comme routines, mais d'un autre côté cela le rend étonnamment concis...

Les permutations factorisantes sont, à mon avis, sous-utilisées. Il s'agit d'un concept puissant, qui serait sans doute d'une grande utilité, pour le calcul de décompositions partitives ou bipartitives selon le même schéma « deux-temps ». [Dah00] donne un algorithme en $O(n+m)$ pour la décomposition en coupes, mais cela n'exclut pas d'en chercher un autre, ou bien de s'intéresser à la décomposition en 2-joints, la décomposition bimodulaire, etc. L'extension d'ordres est par ailleurs utilisée pour l'*orientation transitive*, et là aussi il faut peut-être chercher à simplifier l'existant [MS99]. Et il y a, je pense, des problèmes de coloration où une permutation factorisante permet à l'algorithme glouton de colorier le graphe alors que n'importe quel ordre n'y suffit pas.

Le théorème 35 est également sous-employé : il a sans doute d'autres applications en preuve de complexité.

Mais le principal problème ouvert de l'algorithmique de décomposition modulaire demeure, à mon avis, l'écriture d'un algorithme complètement dynamique (*fully-dynamic* : supportant ajout et suppression d'arêtes et de sommets).

Annexe : algorithme de [Mon01] de décomposition modulaire des graphes orientés

Cette annexe est la version initiale de l'algorithme de §7.2. Y est présenté un algorithme de décomposition modulaire des graphes orientés, en $O(n+m)$. Il est plus compliqué que celui du chapitre 7, car il n'utilise pas la 2-structure H ni l'algorithme d'intersection de familles partitives de §5.5. Il correspond à un travail que j'ai effectué seul, avant celui de §7.2 qui, fruit d'un travail commun avec Ross McConnell, bénéficie de toute l'expérience de ce dernier, ce qui le rend bien plus beau (à mon avis). L'avantage de la version présentée ici est d'être indépendant de l'algorithme de Dahlhaus [Dah00] dont la preuve est un peu lourde.

Cet algorithme fonctionne en cinq étapes, qui seront présentées, et repose sur une analyse plus fine des propriétés de G_s et de G_d (rendue nécessaire par l'ignorance de H). La principale différence est la fonction `IdentArbre`, qui fait le travail le plus dur, et dont la preuve est un peu complexe. Les idées pour faire cette procédure étant à mon avis intéressante, j'ai jugé profitable de les présenter en annexe de cette thèse.

A.1 Nouvelles propriétés de G_s et G_d

Soit $G = (V, E)$ un graphe orienté. On définit $G_s = (V, E_s)$ et $G_d = (V, E_d)$ comme au chapitre 7 : $(u, v) \in E_s$ ssi $(u, v) \in E$ ou $(v, u) \in E$, et $(u, v) \in E_d$ ssi $(u, v) \in E$ et $(v, u) \in E$. Les arbres de décomposition modulaire de ces graphes sont notés respectivement T_s et T_d. On note T_G l'arbre de décomposition modulaire de G, qui est la résultat de l'algorithme.

D'après le lemme 47, tout module de G est module de G_s et de G_d. Malheureusement, un module fort de G peut être faible dans ces deux graphes. Nous allons étudier plus en détails le devenir de chaque module. Un module est un module `tournoi` si son quotient est un tournoi (c'est donc un module `premier` ou `ordre`).

Lemme 57

- *Tout module* `série` *de* G *est un module* `série` *de* G_s *et de* G_d
- *Tout module* `parallèle` *de* G *est un module* `parallèle` *de* G_s *et de* G_d

- *Tout module* `tournoi` *de* G *est un module* `série` *de* G_s *et un module* `parallèle` *de* G_d
- *Tout module* `premier` *de* G *est, dans* G_s*, soit un module* `premier` *soit un module* `série` *; et dans* G_d *est soit un module* `premier` *soit un module* `parallèle`

Démonstration: Pour les nœuds `série`, `parallèle` et `tournoi` ce résultat vient de la proposition 17. Soit M un module `premier` de G. Si M était un module `parallèle` de G_s, comme les non-arêtes de G_s sont exactement les non-arêtes de G, M serait un module `parallèle` de G, contradiction. De même si M était un module `série` de G_d il serait un module `série` de G car les arêtes de G_d sont exactement les arêtes de G.

□

Nous allons définir quelques notions. Un module fort M de G est **identifié** si M est un module fort de G_s ou de G_d : M est alors un nœud de T_s ou de T_d. Sinon, M est non-identifié. Dans ce cas, en notant M_s (resp. M_d) le plus petit module fort de G_s (resp. G_d) contenant M, on dit que M est **fusionné dans** G_s ou G_d. Un **artefact** est un module fort de G_s ou de G_d qui n'est pas un module de G. Étant donné un sous-ensemble $S \subset V$, soit $\Lambda(S)$ le plus petit module fort de G contenant S. Un module non-identifié de G est **identifiable** s'il existe un module fort N de G_s ou de G_d tel que $M = \Lambda(N)$. Un module identifiable est donc soit identifié ($M = N$) soit il existe un artefact A de G_s ou de G_d, inclus dans M mais inclus dans aucun autre module de G plus petit que M. On dit que M **génère** A. Les raisons de ces définitions vont apparaître :

Théorème 47
Les modules forts suivants de G *sont identifiables :*

1. *les modules* `série` *de* G *descendant dans* T_G *d'un nœud* `parallèle` *ou* `tournoi`
2. *les modules* `parallèle` *de* G *descendant dans* T_G *d'un nœud* `série` *ou* `tournoi`
3. *les modules* `tournoi` *de* G *descendant dans* T_G *d'un nœud* `parallèle` *ou* `série`
4. *les modules* `premier` *de* G *dont le quotient n'est pas un tournoi*

Démonstration: D'après le lemme 57, un module `série` de G devient un module `série` de G_d, et un module `parallèle` ou `tournoi` de G devient un module `série` de G_d. Donc tout module fort `série` de G descendant (dans T_G) d'un nœud `parallèle` ou `tournoi` est un module fort de G_d, donc identifié, donc identifiable. De même un module `tournoi` de G descendant dans T_G d'un nœud `série` est un module `parallèle` de G_d descendant d'un `série`, donc est identifié, donc identifiable.

On prouve de même le cas 2 et la fin du cas 3 en montrant que ce module est fort dans G_s.

Pour le quatrième cas, un graphe qui n'est pas un tournoi possède soit une arête, soit une non-arête. Supposons d'abord que le quotient d'un module `premier` M possède une arête. Si M est un module fort de G_d, alors il est identifié, donc identifiable. Sinon, d'après le lemme 57, M est un module `parallèle` de G. Or puisque son quotient possède une arête, d'après la proposition 17, le plus petit module fort de G_d contenant les deux fils F_1 et F_2 correspondant aux deux extrémités de l'arête ne peut être M mais est un artefact A, généré par M, tel que $M = \Lambda(A)$.

Donc M est identifiable. De même si le quotient de M possède une non-arête on montre que M est identifiable grâce à G_s.

□

Corollaire 1
Seuls peuvent être non-identifiables les modules forts

- `série` *ou* `parallèle` *fils (dans T_G) d'un nœud* `premier`
- `tournoi`*, sauf ceux qui sont fils (dans T_G) d'un nœud* `série` *ou* `parallèle`*.*

A.1.1 Modules identifiables

Nous allons nous intéresser aux modules identifiables, mais non identifiés. D'après leur définition, pour les retrouver il suffirait de calculer $\Lambda(N)$ pour tout module fort de G_s ou de G_d, mais cela conduit à un algorithme en $O(nm)$. Nous allons procéder autrement. D'abord, caractérisons-les. En corollaire du lemme précédent :

Corollaire 2
Les modules identifiables non identifiés sont des modules `premier` *qui ne sont pas des tournois*

Lemme 58 *Soit M un module identifiable non-identifié et M_s (resp. M_d) le plus petit module fort de G_s (resp. G_d) qui le contienne. M_s est* `série` *et M_d* `parallèle`*. M est une union de composantes de $G_s[M_s]$, et une union de composantes de $G_d[M_d]$*

Démonstration: Immédiat, puisque dans G_s M_s est `série` comme M, et dans G_d M_d est `parallèle` comme M

□

On dit que M a **fusionné** avec M_s et avec M_d. Nous allons *défaire les fusions*, c'est-à-dire pour chaque nœud où une fusion peut avoir lieu, les nœud `série` de G_s et `parallèle` de G_d, rechercher les unions de composantes qui sont des modules identifiables de G.

Le *quotient* d'un nœud de T_s est défini dans G_s, mais non dans G, car ce nœud peut ne pas être un module de G. Au lieu de graphes quotients, nous allons travailler avec des **graphes caractéristiques**. Étant donné un nœud N de T_s, ayant comme fils $N_1 \dots N_k$ (ce sont les modules forts maximaux de $G_s[N]$), le graphe caractéristique de N est $G_N = (N, E_N)$ où $(u, v) \in E_N$ ssi $(u, v) \in E$ et $u \in N_i$, $v \in N_j$, $i \neq j$. G_N est donc semblable à $G[N]$ sauf qu'il ne contient pas d'arcs « internes » dans un N_i. On définit de même le graphe caractéristique d'un nœud de T_d.

Lemme 59 *Soit M un module identifiable non-identifié, fusionné avec M_s dans G_s et avec M_d dans M_d. M est un module* `premier` *de G_{Ms} et de G_{Md}*

Démonstration: M est un module de $G[M_s]$. M est une union de fils de M_s (lemme 58) donc aucun arc entre M et $M_s \setminus N$ ne change entre G_{Ms} et $G[M_s]$: M est un module de G. Si ce module

est `série`, `parallèle` ou `ordre` alors il l'est aussi dans $G[M_s]$, donc dans G, contradiction. Même preuve pour G_d.

□

Tout au long de cette section, M sera un module fort fusionné avec N. Si le quotient de M possède un arc, on considère que N est un nœud de T_d ; et s'il possède un non-arc, que c'est un nœud de T_s : dans tous les cas ou veut qu'un des fils de N soit un artefact généré par M. On note $\mathcal{F}_N = \{N, N_1 \dots N_k\}$ l'ensemble formé de N et de ses fils $N_1 \dots N_k$ dans T_s (ou T_d, selon l'arbre auquel N appartient).

A.1.2 La fonction `IdentArbre`

Nous allons construire un *arbre* T_N, de racine N et de nœuds $\mathcal{F}_N$. Dans T_s, ils forment déjà un arbre, mais trivial : une racine N et des feuilles $N_1 \dots N_k$. Nous allons réorganiser $\mathcal{F}_N$ de sorte que, si M est un module fusionné avec N, $M = \bigcup_{i \in I} N_i$, alors il existe $i_0 \in I$ tel que N_i descend de N_{i_0} ssi $i \in I$. En d'autres termes, N_{i_0} est un nœud de T_N qui *est* M, puisque M est formé des feuilles situées en-dessous de ce nœud ! C'est le rôle de la procédure `IdentArbre` que de construire cet arbre.

Un module identifiable *génère* toujours un artefact. N_{i_0} sera le premier artefact généré par M que `IdentArbre` rencontre. `IdentArbre` est une procédure récursive. Elle prend deux variables : la première est le nœud N dont elle doit réarranger les fils. Le second est le nœud courant C. `IdentArbre` procède par *ajout itéré* de séparateur : les fils du nœud C sont exactement les nœuds non-encore placés dans T_N qui distinguent des éléments du sous-arbre de racine C. Si C_2 distingue C il sera alors placé comme descendant de C. Et si C_3 distingue C_2 mais non C, il sera placé comme descendant de C_2, donc de C. Ainsi, on ne construit pas tout-à-fait des modules, car les sommets déjà placés dans T_N ne peuvent pas être re-déplacé plus d'une fois. Mais, pour le premier artefact N_{i_0} d'un module fusionné M rencontré par `IdentArbre`, M sera exactement le sous-arbre de racine `IdentArbre`. Le graphe considéré est le graphe caractéristique G_N, et non G. Considérer G produirait le même résultat mais avec une moins bonne complexité temporelle. La section A.3 prouve la correction et la complexité de cette procédure, ce qui est la partie technique de cet algorithme.

Variables

La procédure `IdentArbre` utilise les tableaux suivants, de taille k (indiciés sur les membres de $\mathcal{F}_N$; ce sont des variables globales) :

$Pere[C]$ qui représente le père d'un nœud dans T_N (initialisé à N, sauf pour N lui-même)

$Vu[C]$ vaut `vrai` ssi `IdentArbre`(C, N) a déjà été appelée avec C comme premier argument (initialisé à `faux`)

$Deplacable[C]$ est `vrai` ssi `IdentArbre` est autorisé à déplacer C dans T_N en changeant son père (initialisé à `vrai`, sauf pour N)

Et elle utilise les variables locales suivantes :

C (premier argument de `IdentArbre`) : le nœud courant

N (second argument de `IdentArbre`) : le nœud dont on réarrange les fils

L : file (FIFO) qui stocke les séparateurs du nœud courant

F : nœud fils de C

v : sommet qui est le représentant soit de C soit du fils de C vu juste avant le fils courant (l'instanciation précédente de F)

Algorithme 13: La procédure `IdentArbre`

Procédure `IdentArbre`(C, N)
début
  $Deplacable[C] \leftarrow$ `faux`
  $Vu[C] \leftarrow$ `vrai`
  $L \leftarrow ListeVide$
  si *appel initial* $(C = N)$ **alors**
    `PousserTous`$(L, \mathcal{F}_N)$
  sinon
    `PousserTous`$(L,$ `Sep`$_{Noeud}(C))$
  fin
  $v \leftarrow$ `representant`(C)
  tant que L *n'est pas vide* **faire**
    $F \leftarrow$ `Extraire`$(\mathtt{L})$
    si $Deplacable[F]$ *et* $Pere[F] \neq C$ **alors**
      $Pere[F] \leftarrow C$
      $Deplacable[F] \leftarrow$ `faux`
      `PousserTous`$(L,$ `Sep`$_2(v,$ `representant`$(F)))$
      $v \leftarrow$ `representant`(F)
    fin
    si $Vu[F] =$ `faux` **alors**
      `IdentArbre`(F, N)
    fin
  fin
  pour chaque *fils* F *de* C **faire**
    $Deplacable[F] \leftarrow$ `faux`
  fin
  $Deplacable[C] \leftarrow$ `vrai`
fin

Fonctions auxiliaires

`representant`(C) retourne un sommet appartenant à C

`PousserTous`(L, S) place tous les éléments de l'ensemble S en queue de la file L

`Extraire`(L) ôte la tête de la file L et la retourne comme résultat

$\texttt{Sep}_2(u, v)$ est l'ensemble des nœuds $\{C_i \in \mathcal{F}_N \mid \exists x \in C_i$ tel que x distingue $\{u, v\}\}$.
§5.4.2 discute comment cette fonction peut être implémentée en $O(|\Gamma(u)| + |\Gamma(v)|)$.

$\texttt{Sep}_{Noeud}(C)$ donne l'ensemble de nœuds $\{C_i \in \mathcal{F}_N \mid \exists x \in C_i$ tel que x distingue $C\}$.
Il peut être implémenté avec la fonction $\texttt{Sep}_2$ grâce à l'identité

$$\texttt{Sep}_{Noeud}(C) = \bigcup_{i=1}^{i=h-1} \texttt{Sep}_2(u_i, u_{i+1})$$

A.2 Description et preuve de l'algorithme

Nous proposons un algorithme en cinq étapes. Le but de cette section est de prouver

Théorème 48
Cet algorithme calcule la décomposition modulaire d'un graphe orienté en temps $O(n + m)$

A.2.1 Première étape : les graphes auxiliaires

Elle consiste à calculer G_s et G_d (trivial) puis T_s et T_d. Cela peut se faire en $O(n+m)$ comme expliqué en divers endroits de ce mémoire. À la fin de cette étape, on a :
tout module identifié de G est un nœud de T_s ou de T_d

A.2.2 Deuxième étape : retrouver les modules identifiables

Ensuite, on recrée les modules identifiables non-identifiés dans T_s et dans T_d, en appelant la fonction `IdentArbre` sur chaque nœud susceptible d'abriter une fusion (un nœud `série` de T_s ou `parallèle` de T_d). À la fin de cette étape, on a :
tout module identifiable de G est un nœud de T_s^2 ou de T_d^2
Cela est donné par le lemme 60 établissant la correction de `IdentArbre`. De plus d'après les lemmes 61 et 62, le calcul peut se faire en temps $O(n_N + m_N)$, avec $\sum m_N = m$. Donc, d'après le corollaire 2 du théorème 35, le calcul pour tous les nœuds de T_s et de T_d prend $O(n + m)$ en tout, en remarquant que $n(G) = n(G_s) = n(G_d)$ et $m(G_s) \leqslant m(G)$ et $m(G_d) \leqslant 2m(G)$.

A.2.3 Troisième étape : suppression des artefacts

On supprime maintenant tous les artefacts de T_s^2 et T_d^2, ce qui donne deux nouveaux arbres T_s^3 et T_d^3. La suppression des artefacts peut se faire en $O(n + m)$ pour chaque arbre, grâce à

l'algorithme de §5.4.3 page 142. À la fin de cette étape, on a :
tout module identifiable de G est un nœud de T_s ou de T_d, et réciproquement tout nœud de T_s et de T_d est un module identifiable

A.2.4 Quatrième étape : mélange des deux arbres

À ce point on possède deux arbres dont les nœuds sont exactement les modules identifiables de G. Mais ils sont organisés en *deux* arbres. On peut les réorganiser en un seul arbre, grâce à l'algorithme de §5.5.3 page 150. Il travaille en temps proportionnel à la *Taille* de la famille, donc en temps $O(n+m)$ d'après le théorème 35 page 151. Appelons T^4 l'arbre d'inclusion produit. À la fin de cette étape, on a :
T^4 *est l'arbre d'inclusion des modules identifiables de* G

A.2.5 Cinquième étape : retrouver les modules non-identifiables

Maintenant nous avons un arbre de décomposition unique T^4. Cet arbre contient tous les modules forts identifiables de G, donc (d'après le corollaire du théorème 47) tous les modules forts de G sauf éventuellement

1. les modules `série`, `parallèle` ou `tournoi` fils de module `premier` non-tournoi
2. les `tournoi` fils de `tournoi`

On peut supposer que les nœuds de T^4 sont étiquetés `série`, `parallèle`, `tournoi` ou candidat-`premier` selon que leur quotient comporte uniquement des arêtes, uniquement des non-arêtes, uniquement des arcs simples, ou sinon. Un nœud candidat-`premier` possède un quotient dont les seuls modules non-triviaux sont, d'après ce qui précède, des cliques, des stables et des tournois.

Pour la décomposition des `tournoi`, il n'y à qu'à appliquer l'algorithme 11 de §7.1.

Et pour trouver les cliques, stables et tournois induits dans le quotient d'un candidat-`premier` nous allons chercher des vrais jumeaux, faux jumeaux et demi-jumeaux (voir définition 40 page 206). Un algorithme optimal très simple existe pour trouver des jumeaux dans un graphe (il est issu du folklore de l'affinage de partition ; voir [Pau98]).

`Couper`(C) remplace au sein de $\mathcal{P}$ la classe C par les classes :

- $C \cap (\{x_i\} \cup \Gamma^{\pm}(x_i))$, $C \cap \Gamma^{+}(x_i)$, $C \cap \Gamma^{-}(x_i)$ et $C \cap \Gamma^{0}(x_i)$ pour la recherche de vrais jumeaux
- $C \cap (\{x_i\} \cup \Gamma^{0}(x_i))$, $C \cap \Gamma^{+}(x_i)$, $C \cap \Gamma^{-}(x_i)$ et $C \cap \Gamma^{\pm}(x_i)$ pour la recherche de faux jumeaux
- $C \cap (\{x\} \cup \Gamma^{+}(x_i) \cup \Gamma^{-}(x_i))$, $C \cap \Gamma^{\pm}(x_i)$ et $C \cap \Gamma^{0}(x_i)$ pour la recherche de demi-jumeaux

Soit N un module `premier` de G (c'est un nœud candidat-`premier` de T^4) et G_N son quotient. Les classes non-triviales de vrais jumeaux, faux jumeaux et demi-jumeaux de G_N ne se chevauchent pas. De plus, un module maximal de G_N est soit une classe de jumeaux, soit inclus dans une classe de demi-jumeaux. On recrée donc facilement les nœuds `série` et `parallèle` manquants. Soit C une classe de demi-jumeaux. $G_N[C]$ est un tournoi, qui peut être décomposé

Algorithme 14: Recherche de (vrais,faux,demi)-jumeaux dans un graphe

Entrée : Un graphe G
Sortie : Une partition (non-ordonnée) $\mathcal{P}$ de V en classes de vrais (resp. faux; demi) jumeaux
début
  On se donne un ordre quelconque $x_1 \dots x_n$ sur V.
  $\mathcal{P} = \{V\}$
  pour *i de 1 à n* **faire**
    pour *Toute classe C de $\mathcal{P}$* **faire**
      `Couper`(C)
    fin
  fin
fin

grâce à l'algorithme de §7.1. Or tout module de G inclus dans C est un module de $G_N[C]$. Donc on cherche, parmi les modules produits, ceux qui sont des artefacts et ceux qui sont authentiques. On ajoute les authentiques modules à T^4. Cela se fait comme lors de la quatrième étape. Quand ce processus est achevé pour tous les nœuds N, l'arbre obtenu est l'arbre de décomposition modulaire de G !

L'algorithme trouvant les vrais jumeaux d'un nœud N tourne en temps linéaire en la taille de son quotient. De même pour celui trouvant les faux jumeaux, celui trouvant les demi-jumeaux, et aussi celui décomposant les tournois. Donc, d'après le corollaire 1 du théorème 35, le temps d'exécution total de cette cinquième et dernière phase est $O(n + m)$. À la fin de cette étape, on a :
T_G est l'arbre d'inclusion des modules forts de G

A.3 Preuve de correction de la procédure `IdentArbre`

Cette section contient les preuves de `IdentArbre`. La preuve de correction est donnée par le lemme suivant :

Lemme 60 *Soit M un module identifiable fusionné dans N_s (dans G_s) et dans N_d (dans G_d). Si le quotient de M possède au moins une arête (resp. une non-arête) alors* `IdentArbre`(N_d, N_d) *(resp.* `IdentArbre`(N_s, N_s)*) crée un arbre T_{Ns} (resp. T_{Nd}) où un nœud est*[1] *M.*

Donc tout module identifiable est un nœud de T_s^2 ou de T_d^2. La complexité est la conséquence des lemmes suivants :

1. c'est-à-dire que l'ensemble des feuilles pendant sous ce nœud forme M

Lemme 61 *Les graphes caractéristiques de tous les nœuds de T_s et de T_d peuvent être calculés en temps total $O(n+m)$*

Lemme 62 *Le temps de calcul de la procédure* `IdentArbre`(N, N) *est $O(n_N + m_N)$, où n_N et m_N comptent les sommets et les arêtes du graphe caractéristique de N*

Preuve du lemme 60

Définissons tout d'abord un peu de terminologie. Pour un nœud N de T_s ou de T_d, soit $\mathcal{F}_N = \{C_1, \dots C_k\}$ l'ensemble de ses fils. Nous identifions un nœud de l'arbre avec la partie de V que constituent ses feuilles. Un artefact A est *généré* par un module fusionné M si $A \subsetneq M$ et que tout autre module fort de G vérifiant cette propriété contient M. A est le *premier artefact* généré par M *rencontré par* `IdentArbre` si, quand l'appel `IdentArbre`(A, N) commence, pour tout autre artefact A' généré par M on a $Vu[M] =$ `faux`.

Soit M un module fusionné dans N. On pose $\mathcal{M}(M) = \{C_i \in \mathcal{F}_N \mid C_i \subset M\}$. D'après le lemme 59, $M = \cup\mathcal{M}(M)$. Pour un nœud $C \in \mathcal{F}_N$, on pose $\mathcal{C}(C) = \{C_i \in \mathcal{F}_N \mid$ *quand* `IdentArbre`(C, N) *retourne, C_i est un descendant de C*$\}$, avec la convention que C descend de lui-même. Enfin, $\mathcal{L}(C) \subset \mathcal{F}_N$ est l'ensemble des nœuds qui ont été mis dans la file M par l'appel `IdentArbre`(C, N), et $\mathcal{L}^*(C)$ est l'union de $\mathcal{L}(C_i)$ pour tout descendant $C_i \in \mathcal{C}(C)$ de C.

La procédure `IdentArbre`, appelée sur un nœud N, se contente de *réarranger* $\mathcal{F}_N$, en changeant éventuellement le père d'un nœud $F \in \mathcal{F}_N$, lui donnant comme valeur $F' \in \mathcal{F}_N$ au lieu de N. Cela change l'ensemble de sommets de V correspondant à F. Par contre, pour tout module identifié M, M reste un nœud de T_s^2 (resp. T_d^2).

Dans cette preuve nous nous concentrons sur un module fort M, identifiable mais non-identifié, fusionné avec N. D'après le lemme 59, M est un module `premier` du graphe caractéristique G_N. D'après le théorème 47, M est un module `premier` de G dont le quotient n'est pas un tournoi. La preuve de ce théorème fait apparaître que si ce quotient contient une arête (resp. non-arête) alors M génère un artefact dans G_s (resp. G_d). Donc pour prouver le lemme 60 il n'y a qu'à prouver :

Lemme 63 *Soit M un module fort de G fusionné dans un nœud N. Si N possède un fils qui est un artefact généré par M alors* `IdentArbre`(N, N) *réarrange les fils de N de sorte que, pour le premier artefact A généré par M rencontré par* `IdentArbre`*,*

$$\mathcal{M}(M) = \mathcal{C}(A)$$

Remarque 3
Quand l'appel `IdentArbre`(A, N) *termine, tout nœud descendant de A n'est pas déplaçable. Cela est garanti par la ligne «* `pour chaque` *fils F de C* `faire` $Deplacable[F] \leftarrow$ `faux` *», et est récursivement vrai pour tout descendant de A. Mais A lui-même est déplaçable.*

L'appel initial $\texttt{IdentArbre}(N, N)$, place tous les fils de N dans une file L ; donc au moins un artefact A généré par M sera rencontré par IdentArbre. D'après la remarque 3, après l'appel $\texttt{IdentArbre}(A, N)$, A est déplaçable mais aucun de ses descendants ne l'est. Donc si ce lemme est vrai quand $\texttt{IdentArbre}(A, N)$ termine, il reste vrai jusqu'à la fin de processus. Une dernière remarque avant de nous lancer dans la preuve elle-même :

Proposition 75
Soit G un graphe orienté premier et $\{V_1, V_2\}$ une bipartition de ses sommets. Si $|V_1| \geqslant 1$ et $|V_2| \geqslant 2$,alors il existe $x \in V_1$ et $u, v \in V_2$ tels que x distingue $\{u, v\}$.

Démonstration: Sinon, V_2 est un module non-trivial de G.

□

Nous allons prouver $\mathcal{C}(A) \subset \mathcal{M}(M) \subset \mathcal{L}^*(A) \subset \mathcal{C}(A)$.

1. $\mathcal{C}(A) \subset \mathcal{M}(M)$

Soit T_A le sous-arbre de T_A ayant $\mathcal{C}(A)$ comme nœuds. On numérote $B_1 = A, B_2 \dots B_r$ ces nœuds par un parcours en largeur de T_A, partant de A et tel que, entre deux frères B_i et B_j tels que $i < j$, l'appel $\texttt{IdentArbre}(B_i, N)$ a eu lieu avant l'appel $\texttt{IdentArbre}(B_j, N)$. Nous allons procéder par induction. Supposons que $B_1 \dots B_{i-1}$ appartiennent à $\mathcal{M}(M)$ et montrons $B_i \in \mathcal{M}(M)$.

$B_1 = A$ appartient à $\mathcal{M}(M)$ car A est un artefact généré par M. B_i a reçu comme père le nœud B_j, $j < i$, par l'action de la ligne « $Pere[B_i] \leftarrow B_j$ » de l'algorithme 13. Cela a eu lieu pendant l'appel $\texttt{IdentArbre}(B_j, N)$. Comme B_j descend de A, $B_j \neq N$ et donc B_i a été mis dans la file L par la ligne « $\texttt{PousserTous}(L, \texttt{Sep}_{Noeud}(B_j))$ » ou par la ligne « $\texttt{PousserTous}(L, \texttt{Sep}_2(v, \texttt{representant}(B_l)))$ », $l < i$. L'hypothèse d'induction donne que les sommets de B_j, de B_l, ou que $v \in B_{l-1}$ appartiennent à M. D'après la définition des fonctions $\texttt{Sep}_{Noeud}$ et $\texttt{Sep}_2$, cela signifie qu'il existe un certain sommet $v \in B_i$ qui distingue certains sommets du quotient de M. D'après la définition d'un module, $v \in M$. D'après le lemme 59, $B_i \subset M$.

2. $\mathcal{M}(M) \subset \mathcal{L}^*(A)$

Supposons pour contradiction l'existence de $C \in \mathcal{F}_N$ qui soit membre de $\mathcal{M}(M)$ mais non de $\mathcal{L}^*(A)$. Nous considérons l'arbre T_A, sous-arbre de T_N de racine A. Soient V_2 les sommets du quotient de M qui soient dans $\mathcal{L}^*(A)$ (ce sont « presque » les feuilles de T_A, à l'opération de quotient près), et V_1 autres sommets du quotient de M. V_2 possède plus de deux sommets car $A \subset V_2$. On a supposé pour contradiction V_1 non-vide. D'après la proposition 75, il existe $x \in V_1$ et $u, v \in V_2$ tels que x distingue $\{u, v\}$. Soit S le fils de N auquel x appartient. $S \subset V_1$, donc $S \notin \mathcal{L}^*(A)$. Nous allons montrer que x distingue deux sommets sur lesquels la fonction $\texttt{Sep}_{Noeud}$ ou $\texttt{Sep}_2$ a été appelés, ce qui amène à $S \in \mathcal{L}^*(A)$, contradiction : V_1 est vide. Donc $C \in \mathcal{L}^*(A)$.

Nous pouvons supposer que, pour un x donné, u et v soient pris à *distance minimale* dans T_N (la distance d étant la longueur du chemin dans cet arbre les séparant. Elle vaut au moins 2). Si $d(u,v) = 2$ alors u et v sont des feuilles du même nœud $C \in \mathcal{C}(A)$. S a alors été placé dans la file L de l'appel $\texttt{IdentArbre}(C, N)$ par la ligne « $\texttt{PousserTous}(L, \texttt{Sep}_{Noeud}(C))$ » et donc $S \in \mathcal{L}^*(A)$.

Sinon, soit U_0 tel que $u \in U_0$, V_0 tel que $v \in V_0$, U_1 le père de U_0 dans T_N, V_1 le père de V_0 dans T_N, et ainsi de suite jusqu'à leur dernier ancêtre commun $U_k = V_l$. x ne distingue aucune paire de sommets qui descendent tous deux de U_{k-1} : cela contredirait la minimalité de $d(u,v)$. x ne distingue pas non plus deux descendants de V_{l-1}. Mais comme x distingue $\{u, v\}$, il distingue $\{\texttt{representant}(U_{k-1}), \texttt{representant}(V_{l-1})\}$. Soient $C_1 \dots C_z$ les fils de $U_k = V_l$, et $r_1 \dots r_z$ leurs sommets représentants. Comme x distingue deux des leurs ($r_a = \texttt{representant}(U_{k-1})$ et $r_c = \texttt{representant}(V_{l-1})$), il en distingue deux *consécutifs* r_b et r_{b+1}, $a \leqslant b < c$. Donc la ligne « $\texttt{PousserTous}(L, \texttt{Sep}_2(v, \texttt{representant}(F)))$ », avec $r_b = v$ et $r_{b+1} = \texttt{representant}(F)$, place $S \ni x$ dans la file L de $U_l \in \mathcal{C}$, donc $S \in \mathcal{L}^*(A)$.

3. $\mathcal{L}^*(A) \subset \mathcal{C}(A)$

Le code de la procédure IdentArbre permet de voir que si $F \in L$, pour un nœud C donné, et si $Deplacable[F] = \texttt{vrai}$, alors au final $Pere[F] \leftarrow C$. Donc tout nœud F de $\mathcal{L}^*(A)$, tel que $Deplacable[F] = \texttt{vrai}$ quand $\texttt{IdentArbre}(A, N)$ a commencé, appartient à $\mathcal{C}(A)$. Par hypothèse A est le premier artefact généré par M rencontré par IdentArbre ; tous les autres artefacts générés par M n'étant pas encore vus, donc déplaçables. Un nœud C non-Vu est $Deplacable$, donc s'il est un artefact généré par M il est placé dans $\mathcal{C}(A)$ par IdentArbre. Seuls peuvent ne pas descendre de A les nœuds déjà vus de $\mathcal{L}^*(A)$, qui peuvent ne pas être déplaçable.

Soit $C \in \mathcal{L}^*(A)$ un tel nœud, déjà marqué Vu quand $\texttt{IdentArbre}(A, N)$ débute. L'appel $\texttt{IdentArbre}(C, N)$ a donc déjà eu lieu, *avant* l'appel $\texttt{IdentArbre}(A, N)$. Puisque A est le premier artefact généré par M rencontré par IdentArbre, C n'est pas un artefact généré par M. Il existe donc un module fort $M' \subsetneq M$ tel que $C \subset M'$. Si $C = M'$ alors comme M' est un module, $L = \texttt{Sep}_{Noeud}(C) = \emptyset$: $\texttt{IdentArbre}(C, N)$ se contente de faire $Deplacable[C] \leftarrow \texttt{vrai}$: C est déplaçable. Sinon, C est un artefact généré par M'.

Terminons la preuve en utilisant l'induction. Supposons ce lemme 63 vérifié par tout module fort $M' \subsetneq M$. Soit A' le premier artefact généré par M' rencontré par IdentArbre. par hypothèse d'induction, $\mathcal{M}(M') = \mathcal{C}(A')$, donc C descend de A'. D'après la remarque 3, un sommet qui descend de A' n'est pas déplaçable, à l'exception de A' lui-même. Ainsi, A' est déplaçable et est membre de $\mathcal{L}^*(A)$, donc quand l'appel $\texttt{IdentArbre}(A, N)$ termine, A' descend de A. Et tous les autres sommets de M' sont non-déplaçables et descendent de A', donc de A ! Donc $\mathcal{L}^*(A) \subset \mathcal{C}(A)$. Cela termine la preuve du lemme 60. □

Preuve du lemme 61

Nous allons considérer simultanément tous les nœuds de T_s (la même preuve vaut pour T_d). Soit $\mathcal{L}$ la liste de tous les arcs de G, vus comme des couples (v, v'). Soit N un nœud de T_s. Un arc (v, v') appartient à la *liste d'arcs* de N ssi N est le dernier ancêtre commun de v et v' dans T_s. Il existe des algorithmes [HT84, AGKR02] de calcul du dernier ancêtre commun, demandant un temps de pré-traitement de $O(n)$ par arbre puis un temps d'exécution de $O(1)$ par requête, permettant de construire simultanément les listes d'arcs de tous les nœuds, en examinant les arcs un par un. Ensuite, pour les listes d'adjacence de G_N on n'a qu'à trier ces listes selon une même permutation, grâce à un tri linéaire (voir §5.4.1). □

Preuve du lemme 62

1. D'abord, montrons que `IdentArbre` *est appelée une et une fois pour tout fils* C_i *de* N.
`IdentArbre` est récursivement appelé sur chaque élément non-Vu de L. Or durant l'appel initial tous les fils de N sont mis dans la file L par $\texttt{PousserTous}(L, \mathcal{F}_N)$. Donc `IdentArbre` est appelée au moins une fois pour chacun d'eux. Mais de plus `IdentArbre` ne peut être appelée sur un nœud C_i que si $Vu[C_i] =$ `faux`. Or, la première chose que fait `IdentArbre` sur un nœud est de mettre Vu à `vrai`, et cette valeur ne peut revenir à `faux`. Donc `IdentArbre` est appelée au plus une fois par nœud.

2. Puis montrons que *le père d'un nœud* $C_i \in \mathcal{F}_N$ *peut être changé au plus deux fois par* `IdentArbre`
Le père de C_i ne peut être changé que si $Deplacable[C_i] =$ `vrai`. Or immédiatement après un changement de père, $Deplacable[C_i]$ est mis à `faux`. Elle n'est mise à `vrai` que dans deux circonstances : l'initialisation, et un appel $\texttt{IdentArbre}(C_i, N)$. Or d'après le point précédent cela n'arrive qu'une fois.

3. *Un sommet donné de* G_N *peut être utilisé au maximum sept fois comme argument de la fonction* $\texttt{Sep}_2$.
En effet un sommet $u \in C_i$ est d'abord utilisé deux fois comme argument de $\texttt{Sep}_2$ par l'appel $\texttt{Sep}_{Noeud}(C_i)$, qui n'a lieu qu'une fois (point 1). Si u n'est pas un *représentant* les choses s'arrêtent là, et si $u = \texttt{representant}(C_i)$ alors il va servir aux appels de la ligne « $\texttt{PousserTous}(L, \texttt{Sep}_2(v, \texttt{representant}(F)))$ ». Il sert une fois comme première instanciation de v dans l'appel $\texttt{IdentArbre}(C_i, N)$, et deux fois comme instanciation de v et de $\texttt{representant}(F)$ dans l'appel du père de C_i (on a alors $C_i = F$). D'après le point précédent le père de C_i peut changer seulement deux fois. Donc il y a au plus $2 + 1 + 2 \times 2 = 7$ appels.

4. *Le temps d'exécution de* $\texttt{IdentArbre}(N, N)$*, mis a part le temps pris par les appels de* $\texttt{Sep}_{Noeud}$ *et* $\texttt{Sep}_2$*, est* $O(n_N + m_N)$
En effet, soit $p(C)$ le nombre d'éléments insérés dans la file L par $\texttt{IdentArbre}(C, N)$, et soit $s(C)$ le nombre de fils de C à la fin de l'appel $\texttt{IdentArbre}(C, N)$. La boucle « **tant que** L n'est pas vide **faire** » est appelée $p(C)$ fois. Chaque tour dans cette boucle prend $O(1)$ plus le nombre d'éléments enfilés dans L par $\texttt{Sep}_2(v, \texttt{representant}(v)$ (qui sont $O(p(C))$ en tout) et plus le

temps mis par les appels récursifs $\texttt{IdentArbre}(F, N)$.

La boucle finale « **pour chaque** *fils* F *de* C » prend un temps $O(s(C))$. Donc la procédure $\texttt{IdentArbre}(C, N)$ s'exécute en temps $O(p(C) + s(C))$, plus le temps mis par les appels récursifs, si le temps de calcul de $\texttt{Sep}_{Noeud}$ and $\texttt{Sep}_2$ est négligé.

Or $p(C)$ vaut exactement $|\texttt{Sep}_{Noeud}(C)| + \sum |\texttt{Sep}_2(u, v)|$; un appel à $\texttt{Sep}_{Noeud}$ étant en fait $k-1$ appels à $\texttt{Sep}_2$. La taille de la sortie de $\texttt{Sep}_2(u, v)$ est $O(|\Gamma(u)| + |\Gamma(v)|)$ (voir §5.4.2). Et d'après le point 3, un sommet v est utilisé au plus sept fois comme argument de $\texttt{Sep}_2$ (en incluant les appels à $\texttt{Sep}_{Noeud}$). Donc :

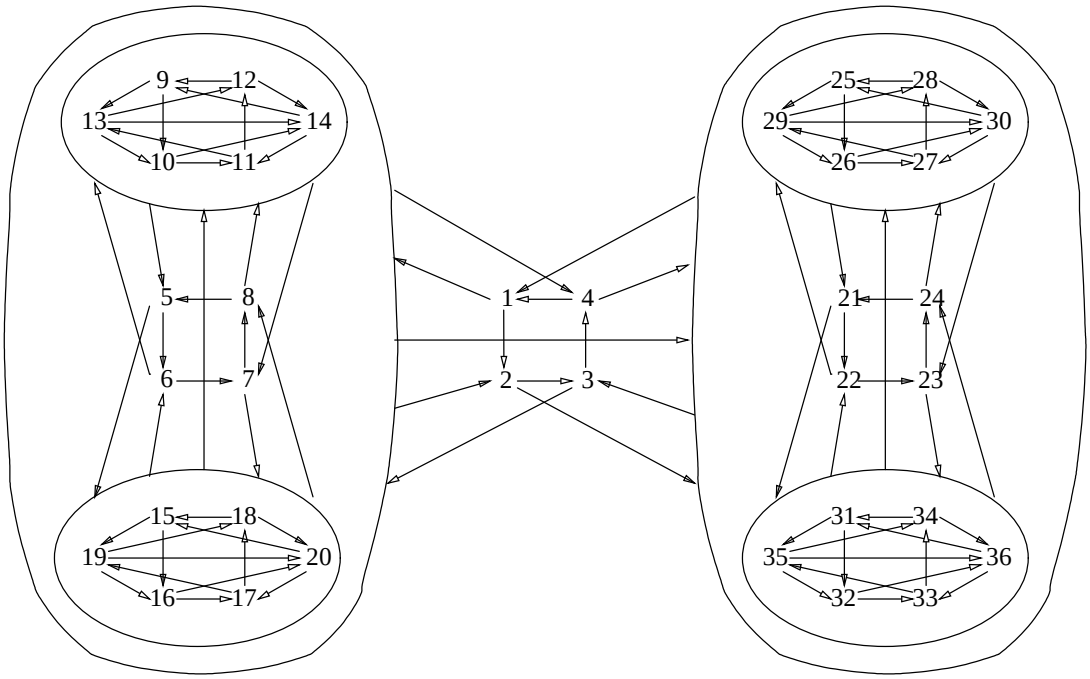
$$\sum_{C \in \mathcal{F}_N} p(C) \leqslant 7 \sum_{v \in V_N} |N(v)| = O(n_N + m_N)$$

Par ailleurs $\sum_{C \in \mathcal{F}_N} s(C) = O(k) = O(n_N + m_N)$, car il y a k nœuds dans $\mathcal{F}_N$.

Pour terminer la preuve du lemme 62, considérons le temps mis pour calculer $\texttt{Sep}_2$. Puisque cette fonction est appelée au plus sept fois par sommet et tourne en $O(|\Gamma(u)| + |\Gamma(v)|)$, ce temps est borné par $7(n_N + m_N)$. Donc le temps d'exécution de $\texttt{IdentArbre}(N, N)$, de ses appels récursifs et de ses appels à $\texttt{Sep}_{Noeud}$ et $\texttt{Sep}_2$ est $O(n_N + m_N)$. □

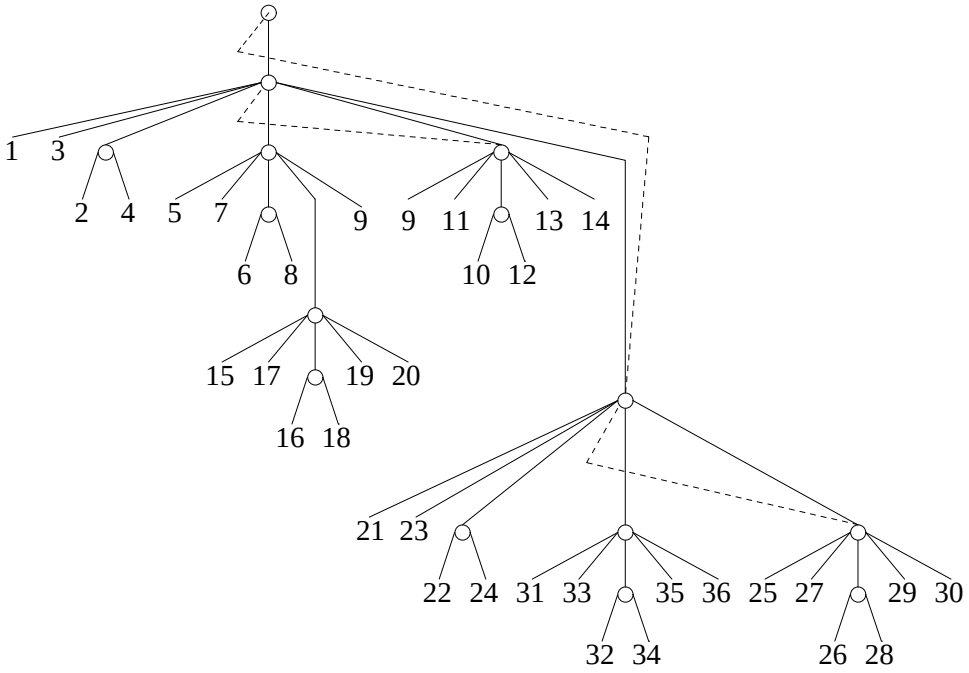
A.4 Exemple

Cet exemple illustre seulement le fonctionnement de la fonction IdentArbre. Prenons le graphe suivant :



La figure de base est un octaèdre orienté. Ensuite, on substitue aux deux sommets des pyramides des copies de lui-même, et le processus est recommencé une fois. Ainsi, nous obtenons pour G_d un stable, et pour G_s une composition série des modules suivants (les modules à deux sommets sont `parallèle`) : $\{1,3\}$, $\{2,4\}$, $\{5,7\}$, $\{6,8\}$, $\{9,11\}$, $\{10,12\}$, $\{13\}$, $\{14\}$, $\{15,17\}$, $\{16,18\}$, $\{19\}$, $\{20\}$, $\{21,23\}$, $\{22,24\}$, $\{25,27\}$, $\{26,28\}$, $\{29\}$, $\{30\}$, $\{31,33\}$, $\{32,34\}$, $\{35\}$, $\{36\}$.

On pourrait continuer à substituer plus longtemps : ce processus crée un graphe ayant un arbre de décomposition modulaire de hauteur arbitraire. Or, G_d est toujours une composition série de $\overline{K_1}$ et de $\overline{K_2}$, et G_d toujours un stable. Heureusement, tous les modules `premier` de G possèdent un non-arête et sont tous identifiables dans G_s. Donc, `IdentArbre` produit l'arbre suivant, pour la racine (série) de T_s, où l'on voit que tous les modules premiers sont représentés. Les arêtes pointillées indiquent la position initiale d'un nœud, avant qu'il ne soit re-déplacé.



Trace de l'exécution de `IdentArbre` (les sommets en italique sont déjà vus mais déplaçables)

IdentArbre ({1,2,...35,36},N)
On choisit $F_i = \{21, 23\}$
Sep_2(21,23)=$\{22, 24\}$
IdentArbre ($\{22, 24\}$,21)
Sep_2(22,24)=$\emptyset$
Sep_2(23,22)=$\{25, 26, ...35, 36\}$
IdentArbre ($\{25, 26, ...35, 36\}$,21)
On choisit $F_i = \{25, 27\}$
Sep_2(25,27)=$\{26, 28\}$
IdentArbre ($\{26, 28\}$,25)
Sep_2(26,28)=$\emptyset$
Sep_2(27,26)=$\{29, 30\}$
IdentArbre ($\{29, 30\}$,25)
Sep_2(26,29)=$\emptyset$
Sep_2(29,30)=$\emptyset$
25 devient déplaçable
Relance avec $F_i = \{31, 33\}$
Sep_2(31,33)=$\{32, 34\}$
IdentArbre ($\{32, 34\}$,31)
Sep_2(32,34)=$\emptyset$
Sep_2(33,32)=$\{35, 36\}$
IdentArbre ($\{35, 36\}$,31)
Sep_2(32,35)=$\emptyset$
Sep_2(35,36)=$\emptyset$
25 est déplacé
Sep_2(22,25)=$\emptyset$
Sep_2(25,31)=$\emptyset$
21 devient déplaçable
On choisit $F_i = \{1, 3\}$
Sep_2(1,3)=$\{2, 4\}$
IdentArbre ($\{2, 4\}$,1)
Sep_2(2,4)=$\emptyset$
Sep_2(3,2)=$\{5, 6, ...20, \mathit{21}\}$
IdentArbre ($\{5, 6, ...20, \mathit{21}\}$,1)
On choisit $F_i = \{9, 11\}$
Sep_2(9,11)=$\{10, 12\}$
IdentArbre ($\{10, 12\}$,9)
Sep_2(10,12)=$\emptyset$
Sep_2(11,10)=$\{13, 14\}$
IdentArbre ($\{13, 14\}$,9)
Sep_2(10,13)=$\emptyset$
Sep_2(13,14)=$\emptyset$
9 devient déplaçable
On choisit $F_i = \{5, 7\}$
Sep_2(5,7)=$\{6, 8\}$
IdentArbre ($\{6, 8\}$,5)
Sep_2(6,8)=$\emptyset$
Sep_2(7,6)=$\{\mathit{9}, 15, ...20\}$
IdentArbre ($\{\mathit{9}, 15, ...20\}$,5)
On choisit $F_i = \{15, 17\}$
Sep_2(15,17)=$\{16, 18\}$
IdentArbre ($\{16, 18\}$,15)
Sep_2(16,18)=$\emptyset$
Sep_2(17,16)=$\{19, 20\}$
IdentArbre ($\{19, 20\}$,15)
Sep_2(16,19)=$\emptyset$
Sep_2(19,20)=$\emptyset$
9 est déplacé
Sep_2(6,15)=$\emptyset$
Sep_2(15,9)=$\emptyset$
21 est déplacé
Sep_2(2,5)=$\emptyset$
Sep_2(5,21)=$\emptyset$

Liste des Algorithmes

Table des notations

Relations logiques		
$\wedge$	et	conjonction
$\vee$	ou	disjonction
$\neg$	non	négation

Relations ensemblistes		
$\uplus$	union disjointe	$A \uplus B = A \cup B$ avec $A \cap B = \emptyset$
$\setminus$	différence	$x \in (A \setminus B)$ ssi $x \in A$ et $x \notin B$
Δ	différence symétrique	$A \Delta B = (A \setminus B) \cup (B \setminus A)$
$\owedge$	chevauchement	$A \owedge B$ ssi $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$ et $B \setminus A \neq \emptyset$

Ensembles divers		
$[\![a, \dots b]\!]$	Intervalle entier	$n \in [\![a, \dots b]\!]$ si $n \in \mathbb{N}$ et $a \leqslant n \leqslant b$
$\mathfrak{S}_n$	Ensemble des permutations de $[\![1, \dots n]\!]$	
$\mathfrak{S}_E$	Ensemble des permutations de E	
2^E	Ensemble des parties de E	
$\mathcal{T}_\mathcal{P}$	Arbre partitif ou bipartitif de $\mathcal{P}$	
$\mathcal{T}_G$	Arbre de décomposition (en particulier modulaire) de G	
$Taille(\mathcal{F})$	taille	$Taille(\mathcal{F}) = \sum_{E \in \mathcal{F}} \lvert E \rvert$

Graphes non-orientés		
$G[X]$	graphe induit	$G[X] = (X \subset V, E \cap (X \times X))$
$\Gamma(x)$	voisinage	$y \in \Gamma(x)$ ssi $(x, y) \in E$
$\overline{\Gamma(x)}$	non-voisinage	$y \in \overline{\Gamma(x)}$ ssi $(x, y) \notin E$

Graphes orientés		
$\Gamma(x)$	voisinage	$\Gamma(x) = \Gamma^{\pm}(x) \cup \Gamma^{+}(x) \cup \Gamma^{-}(x)$
$\Gamma^{\pm}(x)$	voisinage double	$y \in \Gamma^{\pm}(x)$ ssi $(x, y) \in E$ et $(y, x) \in E$
$\Gamma^{+}(x)$	voisinage sortant	$y \in \Gamma^{+}(x)$ ssi $(x, y) \in E$ et $(y, x) \notin E$
$\Gamma^{-}(x)$	voisinage entrant	$y \in \Gamma^{-}(x)$ ssi $(x, y) \notin E$ et $(y, x) \in E$
$\Gamma^{0}(x)$	non-voisinage	$y \in \Gamma^{0}(x)$ ssi $(x, y) \notin E$ et $(y, x) \notin E$

Permutations et ordres partiels		
$\prec_{\mathcal{O}}$	précédence	$\prec_{\mathcal{O}}$ est un ordre partiel sur V
$\|\|_{\mathcal{O}}$	incomparabilité	$x\|\|_{\mathcal{O}}y$ ssi $x \not\prec_{\mathcal{O}} y$ et $x \not\succ_{\mathcal{O}} y$
$\|\|_{\mathcal{O}}$	$\|\|_{\mathcal{O}}$ ensembliste	$A\|\|_{\mathcal{O}}B$ ssi $\forall a \in A\ \forall b \in B\ a\|\|_{\mathcal{O}}b$
$\preccurlyeq_{\mathcal{O}}$	précédence ensembliste	$A \preccurlyeq_{\mathcal{O}} B$ si $\exists a \in A\ \exists b \in B\ a \prec_{\mathcal{O}} b$ et mais $\neg(A\|\|_{\mathcal{O}}B)$
$\perp_{\mathcal{O}}$	croisement	$A\perp_{\mathcal{O}}B$ si $\exists a, a' \in A\ \exists b, b' \in B\ a \prec_{\mathcal{O}} b$ et $a' \succ_{\mathcal{O}} b'$
x'	successeur	$\sigma^{-1}(i)' = \sigma^{-1}(i+1)$
$'x$	prédécesseur	$'\sigma^{-1}(i) = \sigma^{-1}(i-1)$
$\oplus$	succession de classes	$C_1 \oplus C_2$ si ces classes se suivent dans $\mathcal{O}$
$[\ldots]$	facteur	$x \in [a \ldots b]$ ssi $\sigma(a) \leqslant \sigma(x) \leqslant \sigma(b)$
$[\![, \ldots]\!]$	facteur	$x \in [\![i, \ldots j]\!]$ ssi $i \leqslant \sigma^{-1}(x) \leqslant j$

Comparaisons asymptotiques		
$O(f)$	$\leqslant$ asymptotique	$g = O(f)$ si $\exists c > 0\ \ \exists N\ \ n > N \Rightarrow g(n) \leqslant c.f(n)$
$\Theta(f)$	$=$ asymptotique	$g = \Theta(f)$ si $g = O(f)$ et $g = \Omega(f)$
$\Omega(f)$	$\geqslant$ asymptotique	$g = \Omega(f)$ si $\exists c > 0\ \ \exists N\ \ n > N \Rightarrow g(n) \geqslant c.f(n)$

Algorithmes de décomposition		
Sep	Ens. des séparateurs	$x \in \mathrm{Sep}(S)$ ssi S pas module de $G[S \cup \{x\}]$
bg	bord gauche	$bg(N) = \min_\sigma(N)$
bd	bord droit	$bd(N) = \max_\sigma(N)$
ps	premier séparateur	$ps(x, x') = \min_\sigma(\mathrm{Sep}(\{x, x'\}))$
ds	dernier séparateur	$ds(x, x') = \max_\sigma(\mathrm{Sep}(\{x, x'\}))$
$\hat{\sigma}$	Permutation factorisante parenthésée	
$\mathcal{T}_\sigma$	Arbre des fractures pour σ de G	
g	borne gauche	$g([x, y]) = x$ (graphe d'intervalles)
d	borne droite	$d([x, y]) = y$

Index

Bibliographie

[AGKR02] S. ALSTRUP, C. GAVOILLE, H. KAPLAN, et T. RAUHE. Nearest common ancestors : A survey and a new distributed algorithm. In *Proc. 14th Annual Symposium on Parallel Algorithms and Architectures*. ACM, 2002. Cité page(s) 144, 234

[AP90] G. ADHAR et S. PENG. Parallel algorithms for cographs and parity graphs with applications. *Journal of algorithms*, vol. 11, pages 252–284, 1990. Cité page(s) 131

[APT79] B. ASPVALL, M. PLASS, et R. TARJAN. A linear-time algorith for testing the truth of certain quantified boolean formulas. *Information processing letters*, vol. 8, n^o 3, pages 121–132, 1979. Cité page(s) 80

[Ash59] R. L. ASHENHURST. The decomposition of switching functions. In *Proceedings of the International Symposium of the Theory of Switching*. Harvard University Press, Cambridge, 1959. Cité page(s) 39

[Bau96] S. BAUMANN. A linear algorithm for the homogeneous decomposition of graphs. Rapport Tech. TUM-M9615, Technische Universität München, 1996. Cité page(s) 52

[BB01] D. BRYANT et V. BERRY. A structured familly of clustering and tree construction method. *Advances in applied mathematics*, vol. 27, pages 705–732, 2001. Cité page(s) 30

[BBM03a] T. BENNOUAS, M. BOUKLIT, et F. DE MONTGOLFIER. Un modèle gravitationnel du web. In *Actes des 1ères Journées Francophones de la Toile*. 2003. http://www.antsearch.univ-tours.fr/jft2003/. Cité page(s) 18, 19

[BBM03b] T. BENNOUAS, M. BOUKLIT, et F. DE MONTGOLFIER. Un modèle gravitationnel du web. In *Actes de* ALGOTEL03 *5ème Rencontres Francophones sur les aspects Algorithmiques des Télécommunication*. 2003. http://dept-info.labri.u-bordeaux.fr/algotel03/CameraReady/51.ps. Cité page(s) 18, 19

[BCHP03] A. BRETSCHER, D. CORNEIL, M. HABIB, et C. PAUL. A simple linear-time LexBFS cograph recognition algorithm, 2003. Manuscript. Cité page(s) 131

[BDV95] P. BONIZZONI et G. DELLA VEDOVA. Modular decomposition of hypergraphs. In *Graph-Theoretic Concepts in Computer Science, 21th International Workshop*

WG'95, edité par NAGL ET AL., vol. 1017 de *Lecture Notes of Computer Science*. Springer-Verlag, 1995. Cité page(s) 58, 131

[BDV99] P. BONIZZONI et G. DELLA VEDOVA. An algorithm for the modular decomposition of hypergraphs. *Journal of Algorithms*, vol. 32, n^o 2, pages 65–85, 1999. Cité page(s) 58, 131

[Ber70] C. BERGE. *Graphes et Hypergraphes*. Dunod, 1970. Cité page(s) 41, 43

[Bil71] L. J. BILLERA. On the composition and decomposition of clutters. *J. Comb. Theory*, vol. B, n^o 11, pages 234–245, 1971. Cité page(s) 58

[BL76] S. BOOTH et G. LUEKER. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-trees algorithms. *Journal of Computing Systems*, vol. 13, pages 335–379, 1976. Cité page(s) 29, 134

[Bla78] A. BLASS. Graphs with unique maximal clumping. *Journal of Graph Theory*, vol. 2, pages 19–24, 1978. Cité page(s) 44, 131

[BLS99] A. BRANDSTÄDT, V. LE, et J. SPINRAD. *Graph Classes : a Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999. Cité page(s) 43, 44, 49, 50, 52, 187

[BM83] H. BUER et R. MÖHRING. A fast algorithm for the decomposition of graphs and posets. *Math. Oper. Res.*, vol. 8, pages 170–184, 1983. Cité page(s) 131

[BM01] P. BONIZZONI et R. M. MCCONNELL. Nesting of prime substructures in k-ary relations. *Theoretical Computer Science*, vol. 259, pages 341–357, 2001. Cité page(s) 57

[BO99] L. BABEL et S. OLARIU. On the p-connectedness of graphs – a survey. *Discrete Applied Mathematics*, vol. 95, pages 11–33, 1999. Cité page(s) 221

[Bod93] H. L. BODLAENDER. A tourist guide through treewidth. *Acta Cybernetica*, vol. 11, pages 1–23, 1993. Cité page(s) 219

[Bol78] B. BOLLOBAS. *Extremal graph theory*. Academic press, New York, 1978. Cité page(s) 51

[Bon94] P. BONIZZONI. A tight lower bound for primitivity in k-structures. In *Automata Langages and Programming : 21st International Colloquium, ICALP' 94*, edité par S. ABITEBOUL et E. SHAMIR, vol. 820 de *LNCS*, pages 556–567. Springer-Verlag, Berlin, 1994. Cité page(s) 57

[Bra96] A. BRANDSTÄDT. Partition of graphs into two independant sets and cliques. *Discrete Mathematics*, vol. 152, pages 47–54, 1996. Cité page(s) 96

[Cap97a] C. CAPELLE. Block decomposition of inheritance hierarchies. In *Graph-Theoretic Concepts in Computer Science - WG'97*, edité par R. MÖHRING, n° 1335 in LNCS, pages 118–131. Berlin, juin 1997. Cité page(s) 58, 218

[Cap97b] C. CAPELLE. *Décompositions de Graphes et Permutations Factorisantes*. Thèse de doctorat, Université Montpellier II, LIRMM, janvier 1997. Cité page(s) 16, 58, 59, 131, 138, 197, 218

[CC85] G. CORNUÉJOLS et W. CUNNINGHAM. Compositions for perfect graphs. *Discrete Mathematics*, vol. 55, pages 245–254, 1985. Cité page(s) 73

[CE80] W. H. CUNNINGHAM et J. EDMONDS. A combinatorial decomposition theory. *Canad. J. Math*, vol. 32, n^o 3, pages 734–765, 1980. Cité page(s) 12, 24, 32, 39, 58, 219

[CER93] B. COURCELLE, J. ENGELFRIET, et G. ROZENBERG. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, vol. 46, pages 218–270, 1993. Cité page(s) 52

[CH93] A. COURNIER et M. HABIB. An $O(n + m \log n)$ algorithm for substitution decomposition. In *CODIGRAF'93*. Universitat Autónoma de Barcelona and Universitat Politécnica de Catalunya, septembre 1993. Cité page(s) 131

[CH94] A. COURNIER et M. HABIB. A new linear algorithm for modular decomposition. In *Trees in algebra and programming—CAAP 94, 19th International Colloquium, Edinburgh, U.K.*, edité par S. TISON, vol. 787 de *Lecture Notes in Computer Science*, pages 68–84. Springer-Verlag, Berlin, avril 1994. Cité page(s) 16, 49, 81, 130, 131, 138, 194, 195

[CH97] C. CAPELLE et M. HABIB. Graph Decompositions and Factorizing Permutations. In *proceedings of ISTCS'97*, pages 132–143. IEEE, Ramat Gan (Israel), juin 1997. Cité page(s) 16, 130, 131, 138, 197, 218

[Cha47] A. CHATELET. Algèbre des relations de congruence. *Ann. Sci. de l'école Normale Supérieure*, vol. 66, pages 332–368, 1947. Cité page(s) 44

[CHM81] M. CHEIN, M. HABIB, et M. C. MAURER. Partitive hypergraphs. *Discrete Mathematics*, vol. 37, pages 35–50, 1981. Cité page(s) 12, 23, 24, 26, 27, 28, 44, 55, 58, 219

[CHM02] C. CAPELLE, M. HABIB, et F. DE MONTGOLFIER. Graph decomposition and factorizing permutations. *Discrete Mathematics and Theoretical Computer Sciences*, vol. 5, n^o 1, 2002.
URL `http://dmtcs.loria.fr/volumes/abstracts/dm050104.abs.htm`. Cité page(s) 18, 131, 197

[CJS72] D. D. COWAN, L. O. JAMES, et R. G. STANTON. Graph decomposition for undirected graphs. In *3rd S-E Conf. Combinatorics, Graph Theory and computing, Utilitas Math*, edité par R. B. L. e. F. HOFFMAN, pages 281–290. Winnipeg, 1972. Cité page(s) 131

[CLR90] T. CORMEN, C. LEISERSON, et R. RIVEST. *Introduction to Algorithms*. MIT Press, 1990. Traduction française *Introduction à l'algorithmique*, Dunod, 1994. Cité page(s) 141, 150, 213

[CLS81] D. G. CORNEIL, H. LERCHS, et L. K. STEWART. Complement reducible graphs. *Discrete Applied Mathematics*, vol. 3, pages 163–174, 1981. Cité page(s) 50, 129

[Cou93] A. COURNIER. *Sur Quelques Algorithmes de Décomposition de Graphes*. Thèse de doctorat, Université Montpellier II, LIRMM, février 1993. Cité page(s) 131

[CPS85] D. G. CORNEIL, Y. PERL, et L. K. STEWART. A linear recognition algorithm for cographs. *SIAM journal of computing*, vol. 14, n^o 4, pages 926–934, 1985. Cité page(s) 110

[CRST] M. CHUDNOVSKY, N. ROBERTSON, P. SEYMOUR, et R. THOMAS. The strong perfect graph theorem. Manuscript disponible sur `http://www.math.gatech.edu/~thomas/`. Cité page(s) 62

[CS87] V. CHVÁTAL et N. SBIHI. Bull-free berge graphs are perfect. *Graphs Combinatorics*, vol. 3, pages 127–139, 1987. Cité page(s) 61, 62

[CS99a] S. CICERONE et G. D. STEFANO. Graph classes between parity and distance-hereditary graphs. *Discrete Applied Mathematics*, vol. 95, pages 197–216, 1999. Cité page(s) 72

[CS99b] S. CICERONE et G. D. STEFANO. One the extension of bipartite to parity graphs. *Discrete Applied Mathematics*, vol. 95, pages 181–195, 1999. Cité page(s) 72

[Cun73] W. H. CUNNINGHAM. *A combinatorial decomposition theory*. Thèse de doctorat, University of Waterloo, Waterloo, Ontario, Canada, 1973. Cité page(s) 24, 32, 39

[Cun82] W. H. CUNNINGHAM. Decomposition of directed graphs. *SIAM Jounal on Algebraic and Discrete Methods*, vol. 3, pages 214–228, 1982. Cité page(s) 14, 32, 39, 72, 81, 99, 131, 220

[Dah95a] E. DAHLHAUS. Efficient parallel modular decomposition, extended abstract. In *Graph-Theoretic Concepts in Computer Science, 21th International Workshop WG'95*, edité par NAGL ET AL., vol. 1017 de *Lecture Notes of Computer Science*, pages 290–302. Springer-Verlag, 1995. Cité page(s) 131, 186

[Dah95b] E. DAHLHAUS. Efficient parallel recognition algorithms of cographs and distance-hereditary graphs. *Discrete Applied Mathematics*, vol. 57, pages 29–44, 1995. Cité page(s) 131

[Dah00] E. DAHLHAUS. Parallel algorithms for hierarchical clustering, and applications to split decomposition and parity graph recognition. *Journal of Algorithms*, vol. 36, n^o 2, pages 205–240, 2000. Cité page(s) 72, 149, 221, 223

[DGM97] E. DAHLHAUS, J. GUSTEDT, et R. M. MCCONNELL. Efficient and practical modular decomposition. In *8th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 26–35. janvier 1997. Cité page(s) 16, 49, 126, 130, 131, 138

[DGM99] E. DAHLHAUS, J. GUSTEDT, et R. MCCONNELL. Partially complemented Representation of Digraphs. Rapport Tech. RR3832, INRIA, 1999. Soumis à *Discrete Mathematics and Theoretical Computer Science*. Cité page(s) 131, 194

[DGM01] E. DAHLHAUS, J. GUSTEDT, et R. M. MCCONNELL. Efficient and practical algorithms for sequential modular decomposition. *Journal of Algorithms*, vol. 41, n^o 2, pages 360–387, 2001. Cité page(s) 81, 126, 130, 131, 195

[DGM02] E. DAHLHAUS, J. GUSTEDT, et R. M. MCCONNELL. Partially complemented representations of digraphs. *Discrete Mathematics and*

Theoretical Computer Science, vol. 5, n^o 1, pages 147–168, 2002. http://dmtcs.loria.fr/volumes/abstracts/dm050110.abs.html. Cité page(s) 131, 187

[DH89] R. DUCOURNAU et M. HABIB. La multiplicité de l'héritage dans les langages à objects. *Technique et Science Informatique*, vol. 8, n^o 1, pages 41–62, 1989. Cité page(s) 59, 138

[EGMS94] A. EHRENFREUCHT, H. N. GABOW, R. M. MCCONNELL, et S. J. SULLIVAN. An $O(n^2)$ Divide-and-conquer algorithm for the Prime Tree decomposition of 2-structures and the Modular Decomposition of graphs. *Journal of Algorithms*, vol. 16, n^o 2, pages 283–294, 1994. Cité page(s) 53, 126, 130, 131, 187, 188

[EHPR96] A. ENGELFRIET, T. HARJU, A. PROSKUROWSKY, et G. ROZENBERG. Characterization and complexity of uniformly nonprimitive labeled 2-structures. *Theoretical Computer Science*, vol. 154, pages 247–282, 1996. Cité page(s) 53

[EHR94] A. EHRENFEUCHT, T. HARJU, et G. ROZENBERG. Incremental construction of 2-structures. *Discrete Mathematics*, vol. 128, pages 113–141, 1994. Cité page(s) 53, 131

[EKR97] H. EVERETT, S. KLEIN, et B. REED. An algorithm for finding homogeneous pairs. *Discrete Applied Mathematics*, vol. 72, n^o 3, pages 209–218, 1997. Cité page(s) 61, 79, 220

[EM94] A. EHRENFEUCHT et R. MCCONNELL. A k-structure generalization of the theory of 2-structures. *Theoretical Computer Science*, vol. 132, pages 209–227, 1994. Cité page(s) 44, 53, 54, 55, 57

[ER90a] A. EHRENFEUCHT et G. ROZENBERG. Primitivity is hereditary for 2-structures. *Theoretical Computer Science*, vol. 3, n^o 70, pages 343–358, 1990. Cité page(s) 53, 57

[ER90b] A. EHRENFEUCHT et G. ROZENBERG. Theory of 2-structures, Part I : clans, basic subclasses and morphisms. *Theoretical Computer Science*, vol. 3, n^o 70, pages 277–303, 1990. Cité page(s) 44, 53, 55, 56

[ER90c] A. EHRENFEUCHT et G. ROZENBERG. Theory of 2-structures, Part II : Representations throught tree labelled families. *Theoretical Computer Science*, vol. 3, n^o 70, pages 304–342, 1990. Cité page(s) 44, 53, 55

[FG97] J.-L. FOUQUET et V. GIAKOUMAKIS. On semi-p_4-graphs. *Discrete Mathematics*, vol. 165/166, pages 277–300, 1997. Cité page(s) 52

[FGV99] J. L. FOUQUET, V. GIAKOUMAKIS, et J.-M. VANHERPE. Bipartite graphs totally decomposable by canonical decomposition. *International Journal of Foundations of Computer Science*, vol. 10, n^o 4, pages 513–534, 1999. Cité page(s) 15, 98, 103, 110, 221

[FJ90] G. N. FREDERICKSON et R. JANARDAN. Space-ecient message routing in c-decomposable networks. *SIAM Journal on Computing*, vol. 19, n^o 1, pages 164–181, 1990. Cité page(s) 103

[FJKM90] H. FROST, M. JACOBSON, J. KABELL, et F. MORRIS. Bipartite analogues of split graphs and related topics. *Ars Combinatorica*, vol. 29, pages 282–288, 1990. Cité page(s) 102

[Gal67] T. GALLAI. Transitiv orientierbare Graphen. *Acta Math. Acad. Sci. Hungar.*, vol. 18, pages 25–66, 1967. Cité page(s) 44, 129

[GBT84] H. GABOW, J. BENTLEY, et R. E. TARJAN. Scaling and related techniques for geometry problems. *Proceedings of the ACM symposium on theory of computing*, vol. 16, pages 135–143, 1984. Cité page(s) 143

[Gol80] M. C. GOLUMBIC. *Algorithmic graph theory and perfect graphs*. Academic Press, New-York, 1980. Cité page(s) 44, 52, 187

[GR97] V. GIAKOUMAKIS et I. RUSU. Weighted parameters in $(p_5, \overline{P_5})$-free graphs. *Discrete Applied Mathematics*, vol. 80, pages 255–261, 1997. Cité page(s) 52

[GRT97] V. GIAKOUMAKIS, F. ROUSSEL, et H. THUILLIER. On P_4-tidy graphs. *Discrete Mathematics and Theoretical Computer Sciences*, vol. 1, pages 17–41, 1997. Cité page(s) 51, 129

[Gur77] V. GURVICH. On repetition-free boolean functions. *Uspekhi Mat. Nauk*, vol. 32, pages 183–184, 1977. Cité page(s) 50

[GV97a] V. GIAKOUMAKIS et J. VANHERPE. On extended P_4-reducible and extended P_4-sparse graphs. *Theoretical Computer Science*, vol. 180, pages 269–286, 1997. Cité page(s) 52

[GV97b] V. GIAKOUMAKIS et J.-M. VANHERPE. Bi-complement reducible graphs. *Advances in Applied Mathematics*, vol. 18, pages 389–402, 1997. Cité page(s) 15, 98, 101, 103, 109

[GV03] V. GIAKOUMAKIS et J. M. VANHERPE. Linear time recognition and optimizations for weak-bisplit graphs, bi-cographs and bipartite p6-free graphs. *International Journal of Foundations of Computer Science*, vol. 14, n^o 1, pages 107–136, 2003. Cité page(s) 98, 103, 117

[Hab81] M. HABIB. *Substitution des structures combinatoires, théorie et algorithmes*. Thèse d'Ètat, Université Pierre et Marie Curie (Paris VI), 1981. Cité page(s) 12, 23, 28, 131

[HHS95] M. HABIB, M. HUCHARD, et J. SPINRAD. A linear algorithm to decompose inheritance graphs into modules. *Algorithmica*, vol. 13, pages 573–591, 1995. Cité page(s) 58, 59, 131

[Hir] T. HIRAGUCHI. On the dimension of partially ordered sets. Sci. rep. of Kanazawa univ. 1, 77–94. Cité page(s) 52

[HK89] L. HELLERSTEIN et M. KARPINSKI. Learning read-once formulas using membership queries. In *Proceedings of 2nd annual workshop on Computational Learning Theory*. 1989. Cité page(s) 50

[HM79] M. HABIB et M. C. MAURER. On the X-Join decomposition for undirected graphs. *Discrete Applied Math*, vol. 3, pages 198–207, 1979. Cité page(s) 44, 131

[HM91] W.-L. HSU et T.-H. MA. Substitution decomposition on chordal graphs and applications. In *Proceedings of the 2nd ACM-SIGSAM Internationnal Symposium on Symbolic and Algebraic Computation*, n° 557 in LNCS, pages 52–60. Springer-Verlag, 1991. Cité page(s) 138

[HM03] W.-L. HSU et R. M. MCCONNELL. PC-trees and circular-ones arrangements. *Theoretical Computer Science*, vol. 296, pages 99–116, 2003. Cité page(s) 32, 39, 136

[HMP03] M. HABIB, F. DE MONTGOLFIER, et C. PAUL. A simple linear-time modular decomposition algorithm for graphs, using order extending. Rapport Tech. RR03007, LIRMM, 2003. Sumbitted to SODA03 - Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. Cité page(s) 19

[Hoà85] C. HOÀNG. *Perfect Graphs*. Thèse de doctorat, School of Computer Sciences, McGill University, Montreal, 1985. Cité page(s) 51, 129

[Hop71] J. HOPCROFT. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, edité par Z. KOHAVI et A. PAZ, pages 189–196. Academic Press, New York, 1971. Cité page(s) 155, 157

[HP01] M. HABIB et C. PAUL. A simple linear time algorithm for cograph recognition, 2001. Rapport de recherche RR-123800, Labri, Bordeaux. Submitted to Discrete Mathematics. Cité page(s) 131, 138, 155, 160, 168

[HPV99] M. HABIB, C. PAUL, et L. VIENNOT. Partition refinement techniques : an interesting algorithmic toolkit. *International Journal of Foundations of Computer Science*, vol. 10, n^o 2, pages 147–170, 1999. Cité page(s) 16, 130, 131, 141, 155, 156, 157, 160, 161, 239

[HPV01] M. HABIB, C. PAUL, et L. VIENNOT. Linear-time recognition of p_4-indifference graphs. *Discrete Mathematics and Theoretical Computer Sciences*, vol. 4, n^o 2, pages 173–178, 2001. Existe aussi comme rapport de recherche INRIA RR-3779. Cité page(s) 52

[Hsu92] W.-L. HSU. A simple test for interval graphs. *Lecture Notes in Computer Science*, vol. 657, pages 11–16, 1992. Cité page(s) 138

[HT84] D. HAREL et R. TARJAN. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, vol. 13, n^o 2, pages 338–355, 1984. Cité page(s) 144, 234

[Huc92] M. HUCHARD. *Sur quelques questions algorithmiques de l'héritage multiple*. Thèse de doctorat, Université Montpellier II, LIRMM, 1992. Cité page(s) 58, 59

[JO89] B. JAMISON et S. OLARIU. P_4-reducible graphs : a class of uniquely tree representable graphs. *Studies in Applied Mathematics*, vol. 81, pages 79–87, 1989. Cité page(s) 51, 129

[JO91] B. JAMISON et S. OLARIU. On a unique tree representation for P_4-extendible graphs. *Discrete Applied Mathematics*, vol. 35, pages 151–164, 1991. Cité page(s) 51

[JO92] B. JAMISON et S. OLARIU. A unique tree representation for P_4-sparse graphs. *Discrete Applied Mathematics*, vol. 35, pages 115–129, 1992. Cité page(s) 51

[JO95] B. JAMISON et S. OLARIU. p-components and the homogeneous decomposition of graphs. *SIAM Journal on Discrete Mathematics*, vol. 8, pages 448–463, 1995. Cité page(s) 52

[Jol73] J. L. JOLIVET. Sur le joint d'une famille de graphes. *Discrete Mathematics*, vol. 5, pages 145–158, 1973. Cité page(s) 45

[Knu73] D. E. KNUTH. *The Art of Computer Programming*, vol. 1. Addison-Wesley, 1973. Cité page(s) 15

[Lan01] J.-M. LANLIGNEL. *Autour de la decomposition en coupes*. Thèse de doctorat, Université Montpellier II, LIRMM, 2001. Cité page(s) 14, 52, 72, 78, 82, 99, 126, 138, 220

[LO91] R. LIN et S. OLARIU. An NC recognition algorithm for cographs. *J. parallel. distrib. comput.*, vol. 13, pages 76–90, 1991. Cité page(s) 131

[Loz02a] V. LOZIN. Bipartite graphs without a skew star. *Discrete Mathematics*, vol. 257, pages 83–100, 2002. Cité page(s) 110

[Loz02b] V. LOZIN. On maximum induced matchings in bipartite graphs. *Information Processing Letters*, vol. 81, pages 7–11, 2002. Cité page(s) 110

[Mau77] M.-C. MAURER. *Joints et décompositions premières dans les graphes*. Thèse de doctorat, Université Pierre et Marie Curie (Paris VI), 1977. Cité page(s) 131

[McC87] C. MCCREARY. *An algorithm for parsing a graph grammar*. Thèse de doctorat, Univerity of Colorado, Boulder, Colorado, 1987. Cité page(s) 131

[McC95] R. MCCONNELL. An $O(n^2)$ incremental algorithm for modular decomposition of graphs and 2-structures. *Algorithmica*, vol. 14, pages 209–227, 1995. Cité page(s) 53, 131, 187, 188

[Möh85a] R. H. MÖHRING. Algorithmic aspects of comparability graphs and interval graphs. In *Graphs and Orders*, edité par I. RIVAL, pages 41–101. D. Reidel Pub. Comp., 1985. Cité page(s) 52

[Möh85b] R. H. MÖHRING. Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions. *Annals of Operations Research*, vol. 6, pages 195–225, 1985. Cité page(s) 44

[Mül97] H. MÜLLER. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics*, vol. 78, pages 189–205, 1997. Cité page(s) 52, 187

[MM03] R. M. MCCONNELL et F. DE MONTGOLFIER. Linear-time modular decomposition of directed graphs, 2003. Accepté pour publication dans *Discrete Applied Mathematics*. Cité page(s) 19, 131, 187

[Mon00] F. DE MONTGOLFIER. Trouver l'arbre de décomposition modulaire d'un graphe à partir d'une permutation factorisante, 2000. Cité page(s) 18

[Mon01] F. DE MONTGOLFIER. Modular decomposition of tournaments, directed graphs and 2-structures : linear algorithms. Rapport Tech. RR01379, LIRMM, 2001. Cité page(s) 7, 18, 223, 224, 226, 228, 230, 232, 234, 236

[Mon03a] F. DE MONTGOLFIER. Décompositions de graphes. Rapport tech., 11èmes Journées des Doctorants de l'école doctorale ISS - Montpellier, 2003. http://www.lirmm.fr/ jaillet/JDOCS2003/index.php. Cité page(s) 19

[Mon03b] F. DE MONTGOLFIER. A twomodular graph decomposition theory. Rapport tech., soumis à STACS 2004, 2003. Cité page(s) 19

[MR84] R. H. MÖHRING et F. J. RADERMACHER. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Mathematics*, vol. 19, pages 257–356, 1984. Cité page(s) 24, 44, 46, 51, 55, 219

[MS89] J. H. MÜLLER et J. SPINRAD. Incremental modular decomposition. *Journal of the association for Computing Machinery*, vol. 1, pages 1–19, 1989. Cité page(s) 131

[MS94a] T. MA et J. P. SPINRAD. An $O(n^2)$ algorithm for undirected split decomposition. *Journal of Algorithms*, vol. 16, n^o 1, pages 145–160, 1994. Cité page(s) 72

[MS94b] R. MCCONNELL et J. SPINRAD. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (Arlington, VA), pages 536–545. ACM, New York, 1994. Cité page(s) 44, 131, 195

[MS99] R. MCCONNELL et J. SPINRAD. Modular decomposition and transitive orientation. *Discrete Mathematics*, vol. 201, pages 189–241, 1999. Cité page(s) 16, 44, 49, 81, 130, 131, 138, 194, 195, 221

[MS00] R. M. MCCONNEL et J. SPINRAD. Ordered vertex partitioning. *Discrete Mathematics and Theoretical Computer Sciences*, vol. 4, pages 45–60, 2000. Cité page(s) 130, 131

[MV95] M. MORVAN et L. VIENNOT. Parallel comparability graph recognition and modular decomposition. In *Graph-Theoretic Concepts in Computer Science, WG'95*, vol. 1017 de *LNCS*. 1995. Cité page(s) 131

[Ola88] S. OLARIU. No antitwins in minimal imperfect graphs. *J. Combin. Theory Ser.*, vol. B 45, pages 255–257, 1988. Cité page(s) 73

[Pau98] C. PAUL. *Parcours en Largeur Lexicographique : Un Algorithme de Partitionnement, Application aux Graphes et Généralisations*. Thèse de doctorat, Université Montpellier II, LIRMM, janvier 1998. Cité page(s) 44, 138, 141, 155, 229

[PT87] R. PAIGE et R. TARJAN. Three partition refinement algorithms. *SIAM Journal on Computing*, vol. 16, n^o 6, pages 973–989, 1987. Cité page(s) 140, 155

[Riz01] R. RIZZI. On the recognition of P_4-indiffernt graphs. *Discrete Mathematics*, vol. 239, pages 161–169, 2001. Cité page(s) 52

[RRT99] F. ROUSSEL, I. RUSU, et H. THUILLIER. On graphs with limited number of p_4-partners. *International Journal of Fundations of COmputer Science*, vol. 10, pages 103–121, 1999. Cité page(s) 51

[SA96] R. G. SEIDEL et C. R. ARAGON. Randomized search trees. *Algorithmica*, vol. 16, pages 464–497, 1996. Cité page(s) 144

[Sab59] J. SABIDUSSI. The composition of graphs. *Duke Math. Journal*, vol. 26, pages 693–696, 1959. Cité page(s) 45

[Sch81] J. SCHMERL. Arborescent structures. ii : interpretability in the theory of trees. *Transactions of the A.M.S.*, vol. 266, pages 629–643, 1981. Cité page(s) 44

[SF70] L. SHEVRINE et N. FILIPOF. Partially ordered sets and their comparability graphs. *Siberian Mathematics*, vol. 11, pages 497–509, 1970. Cité page(s) 44

[She86] S. SHEW. A cograph approach to examination sheduling. M. Sc. thesis, Dept. of computer sciences, Univerity of Toronto, 1986. Cité page(s) 50

[Sid86] J. SIDNEY. Optimal sequenting by modular decomposition. *Oper. Res.*, vol. 34, 1986. Cité page(s) 52

[SM95] F.-S. SHIEH et C. L. MCCREARY. Directed graphs drawing by clan-based decomposition. In *Graph Drawing*, edité par F.-J. BRANDENBURG, Lecture Notes in Computer Science, pages 472–482. 1995. Cité page(s) 42, 52

[Spi85] J. SPINRAD. On comparability and permutation graphs. *SIAM Journal of Computing*, vol. 14, n^o 3, pages 658–670, 1985. Cité page(s) 52

[Spi92] J. SPINRAD. P4-trees and substitution decomposition. *Discrete Applied Mathematics*, vol. 39, pages 263–291, 1992. Cité page(s) 130, 131, 149, 239

[Spi94] J. SPINRAD. Recognition of circle graphs. *Journal of Algorithms*, vol. 16, pages 264–282, 1994. Cité page(s) 72

[SS03] R. SHAMIR et R. SHARAN. A fully-dynamic algorithm for modular decomposition and recognition of cographs, 2003. Accepté pour publication dans *Discrete Applief Mathematics*. Cité page(s) 131

[Sum73] D. P. SUMNER. Graphs indecomposable with respect to the X-join. *Discrete Mathematics*, vol. 6, pages 281–298, 1973. Cité page(s) 44, 49

[Tut66] W. TUTTE. *Connectivity in graphs*. University of Toronto Press, Toronto, 1966. Cité page(s) 39

[Van99] J.-M. VANHERPE. *Décomposition et algorithmes efficaces sur les graphes*. Thèse de doctorat, Université de Picardie Jules Vernes, Amiens, 1999. Cité page(s) 15, 98, 101, 102, 103, 109, 110, 117, 120

[VLT82] J. VALDES, E. LAWLER, et R. TARJAN. The recognition of serie parallel digraphs. *SIAM J. Comput.*, vol. 11, pages 289–313, 1982. Cité page(s) 51

[Wag90] D. WAGNER. Decomposition of k-ary relations. *Discrete Mathematics*, vol. 81, pages 303–322, 1990. Cité page(s) 72

[Wes96] D. B. WEST. *Introduction to graph theory*. Prentice Hall, 1996. Cité page(s) 41, 43

Résumé

La décomposition modulaire des graphes est le sujet principal de ce mémoire, qui est divisée en deux parties. La première, théorique, commence par présenter des familles de parties d'un ensemble possédant les même propriétés que les modules d'un graphes : les familles partitives, et trois variantes. Elle se poursuit par des généralisations, aux graphes, de la décomposition modulaire : le lecteur trouvera la preuve que la famille des *2-modules* est « presque » partitive, et verra exposées la décomposition en 2-joints, en sesquimodules, ainsi que la décomposition bimodulaire des graphes bipartis. Cette dernière est un parallèle presque complet dans le cas biparti des modules d'un graphe orienté usuel, et se calcule polynômialement.

Enfin la seconde partie, pratique, présente des algorithmes de décomposition modulaire en $O(n+m)$ pour le cas des graphes non-orientés, des tournois, et des graphes orientés, et en $O(n)$ pour les graphes d'intervalles et les graphes de permutation (codés sous forme de leur modèle d'intervalles ou de permutation). Tous ces algorithmes sont linéaires en la taille de la donnée ; les deux premiers sont plus simples que les algorithme existants et les trois derniers atteignent cette borne pour la première fois.

Mots clés

graphes – algorithmes – décomposition modulaire – familles partitives – 2-modules
2-joints – graphes bipartis – graphes d'intervalles – graphes de permutation

Abstract

This thesis is about modular decomposition of graphs. The first part of the manuscript is devoted to theory. First it is question of set decompositions that have the same properties than modular decomposition : the partitive families, and three variations. Then are presented some graph decompositions that extend modular decomposition, using the notion of *2-modules* (homogeneous pairs). This is the case for the split decomposition of Cunningham, and also for the new *2-joins* and *sesquimodules* decompositions. A analog of modules for bipartites graphs, the *bimodule*, leads to a decomposition theory very near the modular decomposition of directed graphs.

The second part is practical and presents modular decompositions algorithms for undirected graphs, tournaments, directed graphs, interval graphs and permutation graphs. All these algorithms run in linear-time from the input : $O(n+m)$ for the three first classes, $O(n)$ for the two last classes, in the graph is coded with an interval or permutation model.

Keywords

graphs – algorithms – modular decomposition – partitive set families – 2-modules
homogeneous pairs – 2-joins – bipartite graphs – interval graphs – permutation graphs

LIRMM — 161 rue Ada — 34392 Montpellier Cedex 5 — France