HAL
open science

Aspects algorithmiques de la décomposition modulaire

Christophe Paul

To cite this version:

Christophe Paul. Aspects algorithmiques de la décomposition modulaire. Mathématiques [math]. Université Montpellier II - Sciences et Techniques du Languedoc, 2006. tel-00112390

HAL Id: tel-00112390

https://theses.hal.science/tel-00112390

Submitted on 8 Nov 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Christophe Paul

Aspects Algorithmiques de la décomposition modulaire

– Habilitation à Diriger des Recherches –

17 Mai 2006

CNRS - LIRMM - Université Montpellier 2

A Vincent, Emma et Mireille

Remerciements

Mes premiers remerciements s'adressent tout naturellement à mes trois rapporteurs qui ont accepté la lourde tâche que représente la lecture critique d'un tel document. Je remercie aussi l'ensemble des membres du jury pour leur présence et pour l'intérêt qu'ils portent à mes travaux.

Les résultats que je présente dans ce mémoire sont, en bonne partie, dû à l'ensemble des collègues et/ou coauteurs avec qui j'ai eu le loisir de travailler: notamment au LIRMM pendant ma thèse et à nouveau depuis 4 ans, au LaBRI au sein de l'équipe *Graphes et Applications*, à Toronto, à Prague, à Bergen et très bientôt à Montréal... Je les remercie tous pour la chaleur et la richesse des échanges que nous avons partagés.

Ce mémoire a aussi vu le jour avec l'aide du tout nouveau groupe CAGOLE du LIRMM (Emeric, Stphan et Stphane), ses relecteurs pugnaces et ses étudiants (Binh Minh et Christophe) avec qui j'ai la chance de collaborer. Je n'oublie pas Michel, toujours enthousiaste et disponible.

Enfin, je tiens à saluer tout ceux qui m'ont acconpagné dans mes travaux (en particulier Cicou, Pierre, Pascal mais aussi bien d'autres), tout ceux qui répondent toujours présents pour un apéro ou un casse-croute (parmi eux Amit, Isa, Margot et Guilhem, Pip et Dynah,...) J'en oublie bien sûr mais écrire des remerciements exhaustifs est une tâche impossible, pour moi. Je les termine donc en pensant à mes deux mathématiciennes préférées, Mireille et Emma, ainsi qu'à Vincent, petit caïd de la rue d'Aubeterre.

Avant-propos

La rédaction d'une Habilitation à Diriger des Recherches est sans doute pour la plupart d'entre nous un exercice délicat. Il consiste d'abord à trouver une suite logique, un fil conducteur, sur les travaux de ces premières années de recherche. De nombreuses options sont permises parmi lesquelles un document le plus exhaustif possible sur les résultats obtenus, une synthèse sur un domaine de recherche émergeant. J'ai fait un autre choix, celui de présenter les nombreux développements autour du problème de la *décomposition modulaire des graphes* que nous avons connu ces dix dernières années. Sur les aspects combinatoires de la décomposition modulaire, l'article de Möhring et Radermacher [MR84] fait toujours référence même si de nombreuses généralisations ou extensions sont apparues ces 20 dernières années. Mais, il n'existe pas, à ma connaissance, dans la littérature, de document similaire sur les aspects algorithmiques de la décomposition modulaire. En tout cas, aucun qui ne soit à jour des résultats de ces dix dernières années qui sont les plus importants puisqu'il existe désormais des algorithmes linéaires [CH94, MS94b].
Il m'a donc semblé opportun, compte tenu de mes contributions et celles de mes co-auteurs sur la décomposition modulaire, de faire le point sur les *aspects algorithmiques de la décomposition modulaire*. L'Habilitation à Diriger des Recherches en était l'occasion. Ce choix a aussi été motivé par la beauté des algorithmes mis en jeu dans ce domaine. J'espère humblement ainsi participer un peu plus à leur diffusion.

Enfin pour rendre plus digeste ce mémoire, je propose à la fin de chaque chapitre, une recette de tarte à consommer avec un rosé léger ou un blanc vendange tardive ou un muscat sec bien frais.

Table des matières

1 Introduction 1

2 Fondements mathématiques 5
- 2.1 Familles partitives et théorème de décomposition 5
- 2.2 Permutations factorisantes 7
- 2.3 Modules d'un graphe 7
- 2.4 Sur la structure des graphes premiers 13
- 2.5 Quelques familles de graphes 13
  - 2.5.1 Les cographes 13
  - 2.5.2 Les graphes P_4-creux 15
  - 2.5.3 Les graphes de comparabilité 16
- 2.6 Variantes et généralisations 19
  - 2.6.1 Décomposition modulaire des graphes orientés 19
  - 2.6.2 Décomposition en coupes 20
- 2.7 Notes bibliographiques 21

3 Affinage de partition 23
- 3.1 Principe et structure de données 23
- 3.2 Calcul d'une partition modulaire et règle de Hopcroft 25
- 3.3 Notes bibliographiques 28

4 Calcul de l'arbre de décomposition modulaire 31
- 4.1 L'algorithme de Ehrenfeucht et al. 31
  - 4.1.1 Calcul de l'arbre $MD(G_{/\mathcal{M}(G,v)})$ 34
- 4.2 Un algorithme quasi-linéaire 36
  - 4.2.1 Calcul de $\mathcal{M}(G,v)$ 36
  - 4.2.2 Calcul de $MD(G_{/\mathcal{M}(G,v)})$ 37
  - 4.2.3 Mise en œuvre et analyse de complexité 41
- 4.3 Une implémentation simple et sous-quadratique 43
- 4.4 Notes bibliographiques 45

5 Autour des permutations factorisantes 47
5.1 Calcul d'une permutation factorisante 48
5.1.1 Notations 48
5.1.2 Principe de base 49
5.2 Le cas des cographes 50
5.3 De la permutation factorisante à l'arbre de décomposition 54
5.3.1 L'arbre des fractures 54
5.3.2 L'algorithme de Uno et Yagiura revisité 55
5.4 Notes bibliographiques 57

6 Parcours en largeur lexicographique 59
6.1 L'algorithme LexBFS 59
6.1.1 Les couches lexicographiques 61
6.1.2 Règles de tie-break 62
6.2 LexBFS et l'orientation transitive 63
6.3 Reconnaissance des cographes 66
6.3.1 La propriété d'inclusion des voisinages 67
6.3.2 Construction du coarbre 69
6.4 Notes bibliographiques 72

7 Représentation de graphes dynamiques 73
7.1 Le cas général 74
7.1.1 Modification d'arête 74
7.1.2 Modification de sommet 75
7.2 Les cographes dynamiques 76
7.2.1 Modification de sommet 76
7.2.2 Modification d'arête 77
7.3 Les graphes de permutation dynamiques 78
7.3.1 Décomposition modulaire des graphes de permutation. . 79
7.3.2 Modification d'arête 80
7.3.3 Suppression de sommet 80
7.3.4 Ajout de sommet 80
7.4 Notes bibliographiques 85

8 Tri par renversements et décomposition modulaire 87
8.1 Intervalles communs 88
8.1.1 Calcul des intervalles communs 91
8.1.2 Liens avec les graphes de permutation 91
8.2 Tri par renversements et scénarios parfaits 92
8.3 Discussion et notes bibliographiques 96

Bibliographie 99

1

Introduction

Malgré la simplicité de leur définition, les graphes sont des objets combinatoires fondamentaux en informatique. D'une part, ils sont la base de nombreux modèles dans des domaines applicatifs aussi différents que les réseaux, la biologie, la chimie... et plus récemment les sciences sociales. Par ailleurs, ils permettent d'exprimer une grande partie de la théorie de la complexité. Il est donc important de comprendre la structure des graphes. Une approche classique, et commune à la plupart des disciplines scientifiques, avait déjà été suggérée par Descartes en 1637 dans le second principe du *Discours de la méthode* :

> *Le second, de diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait et qu'il serait requis pour mieux les résoudre.*

Ce principe, aussi connu sous le terme de *diviser pour résoudre*, est encore d'actualité comme en témoigne la récente preuve [CRST02] de la conjecture des graphes parfaits [Ber61] pour laquelle de nouvelles méthodes de décomposition sont développées. Ce fut aussi le cas pour la preuve de la conjecture de Wagner [Wag37] résolue par Roberston et Seymour et désormais connu sous le nom de Théorème des mineurs de graphes [RS04]. Une étape importante de cette preuve repose la décomposition arborescente d'un graphe [RS86].

Par ailleurs, de nombreux problèmes difficiles (NP-difficile) en général, deviennent polynomiaux lorsqu'ils sont restreints aux arbres. Il est donc naturel de mettre au point des techniques permettant de réduire les graphes aux arbres. Dans cette optique, des invariants de graphe ont été défini : largeur arborescente [RS86], largeur de branche [RS91], largeur de clique [Cou86]... Si l'un de ces paramètres est borné, alors beaucoup de problèmes difficiles admettent un algorithme de complexité paramétrique polynomial [DF97, Nie02]. Ces décompositions sont aussi à l'origine de méthodes heuristiques efficaces. Elles ont permis d'obtenir des résultats significatifs pour le problème du voyageur de commerce [CC].

Sans entrer dans les détails, les méthodes évoquées ci-dessus relèvent d'un paradigme de séparations successives du graphe en blocs. L'arbre de décomposition peut être compris comme une image de ce processus de séparation. Nous allons, dans ce mémoire, nous intéresser à une autre méthode de décomposition, que l'on pourrait qualifier de décomposition en *parties homogènes*. Il s'agit par exemple de définir des relations d'équivalence sur les sommets d'un graphe. Une décomposition hierarchique est alors obtenue en calculant récursivement les partitions en classes d'équivalence. La théorie des *familles partitives* [MR84, CHM81] entre pleinement dans ce cadre et s'applique à de nombreux objets combinatoires (graphes, hypergraphes, permutations, fonctions booléennes...) En terme de graphes, on parle de *modules*. La *décomposition modulaire* a été utilisée pour la première fois par Gallai [Gal67] pour l'étude des graphes de comparabilité. Il en résulte que les ordres partiels partageant le même graphe de comparabilité ont été complément décrits. Par ailleurs, la décomposition modulaire est à la base de représentations compactes pour différentes familles de graphes d'intersection, telles que les graphes de permutations... [Spi03].

Il existe des articles de synthèse relativement complets sur les aspects théoriques des familles partitives, voir par exemple [MR84], même s'ils commencent un peu à dater. Ce n'est en revanche, pas le cas pour l'algorithmique associée. En particulier, les algorithmes obtenus au cours de ces dix dernières années mériteraient une présentation unifiée (éventuellement simplifiée par rapport aux articles d'origine).

Retraçons un bref historique. Le premier algorithme polynomial pour la décomposition modulaire apparaît début 70 [CJS72]. Après des améliorations successives ([CHM81, MS89] par exemple), les deux premiers algorithmes linéaires n'arrivent qu'en 1994 [CH94, MS94b]. Ces deux algorithmes restent "théoriques" tellement ils sont compliqués à prouver et à mettre en œuvre. De nouveaux algorithmes de décomposition modulaire, plus simples que ceux de 1994, avec éventuellement un léger surcoût de complexité (logarithmique), ont depuis été mis au point avec pour objectif principal la simplicité. Pourtant dans son livre *"Efficient graph representation"* [Spi03], Spinrad écrit :

> *Le nouvel algorithme* [linéaire [MS99]] *est trop compliqué pour être présenté ici* [dans le livre [Spi03]]. *Les premiers algorithmes quadratiques [MS89] étaient également compliqués; j'espère et crois que dans quelques années, les algorihmes linéaires seront simplifiés.*

Je propose donc, dans ce document, de faire le point sur l'ensemble des connaissances accumulées autour de l'algorithmique de la décomposition modulaire avec l'espoir de dégager de nouvelles perspectives pour répondre positivement à la remarque de Spinrad.

La présentation s'organise de la manière suivante. Le Chapitre 2 rappelle des définitions et des résultats connus pour la décomposition modulaire des graphes. Puis nous présentons (Chapitre 3) les techniques algorithmiques de base que nous utiliserons par la suite. Curieusement, ces méthodes, basées sur

l'*affinage de partition*, sont apparues très tôt : en 1971 pour le problème de la minimisation d'automates finis déterministes [Hop71]. Le Chapitre 4 est dédié aux algorithmes récents de décomposition modulaire. Nous ne présenterons pas les deux algorithmes linéaires estimant, comme Spinrad, que c'est inutile pour la clarté de la présentation. Enfin dans les Chapitres 5 et 6 nous discutons de nouveaux algorithmes qui sont, à mon avis, très prometteurs dans la perspective d'un algorithme linéaire simple. Pour terminer, les deux derniers chapitres évoquent des applications à la décomposition modulaires : la représentation de graphes dynamiques et la génomique comparative en bioinformatique.

2

Fondements mathématiques

La décomposition modulaire doit être comprise comme un cas particulier du concept de *famille partitive* dont l'étude théorique remonte au début des années 80 [CE80, CHM81]. Nous commencerons par introduire les définitions et théorèmes principaux de cette théorie. Puis nous discuterons plus en détails de la notion de module d'un graphe et aborderons ses aspects algorithmiques de base. La suite de ce chapitre est dédiée à la présentation de familles de graphes pour lesquelles la décomposition modulaire est particulièrement importante (cographes, graphes de comparabilité...) Finalement nous mentionnerons des variantes ou généralisation de la décomposition modulaire.

Ce chapitre n'est que rappel de résultats connus de longue date. Pour les preuves omises, le lecteur devra se référer à la littérature existante.

2.1 Familles partitives et théorème de décomposition

La différence symétrique de deux ensembles A et B est notée $A \Delta B = (A \setminus B) \cup (B \setminus A)$. Deux sous-ensembles A et B d'un ensemble S *se chevauchent* si $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$ et $B \setminus A \neq \emptyset$. Nous noterons la relation de chevauchement entre A et B par $A \perp B$. Par ailleurs, $\subset$ sera le symbole utilisé pour l'inclusion stricte et $\subseteq$ celui utilisé pour l'inclusion large.

Définition 2.1. *Soit $\mathcal{S} \subseteq 2^S$ une famille de parties de S. $\mathcal{S}$ est* partitive *si*

1. *$S \in \mathcal{S}$, $\emptyset \notin \mathcal{S}$ et pour tout $x \in S$, $\{x\} \in \mathcal{S}$,*
2. *Pour toute paire de parties $A, B \in \mathcal{S}$ telles que $A \perp B$:*
   a) *$A \cap B \in \mathcal{S}$;*
   b) *$A \setminus B \in \mathcal{S}$ et $B \setminus A \in \mathcal{S}$;*
   c) *$A \cup B \in \mathcal{S}$;*
   d) *$A \Delta B \in \mathcal{S}$.*

On parle de famille *faiblement partitive* lorsque la condition (2.d) n'est pas requise. Sauf contre-indication explicite, nous ne considèrerons que des

familles partitives. C'est en particulier le cas pour la suite de ce chapitre. Dans le cadre de familles faiblement partitives, certains résultats et définitions doivent être adaptés.

Définition 2.2. *Un élément $F \in \mathcal{S}$ est* fort *s'il ne chevauche aucun autre élément de $\mathcal{S}$. L'ensemble des éléments forts de $\mathcal{S}$ est noté $\mathcal{S}_F$.*

Clairement les éléments triviaux S et $\{x\}$, pour tout $x \in S$, sont des éléments forts. Remarquons que $\mathcal{S}_F$ s'organise en un arbre $T_{\mathcal{S}}$ dont les nœuds sont les éléments de $\mathcal{S}_F$: les feuilles correspondent aux singletons $\{x\}$ tels que $x \in S$, et le père d'un nœud S_1 est l'ensemble minimal $S_2 \in \mathcal{S}_F$ tel que $S_1 \subset S_2$. La racine de $T_{\mathcal{S}}$ est associé à l'ensemble S. En fait $T_{\mathcal{S}}$ est la réduction transitive de l'ordre d'inclusion des éléments de $\mathcal{S}_F$ (voir Figure 2.1). Il en découle que si $|S| = n$, alors $|\mathcal{S}_F| = O(n)$.

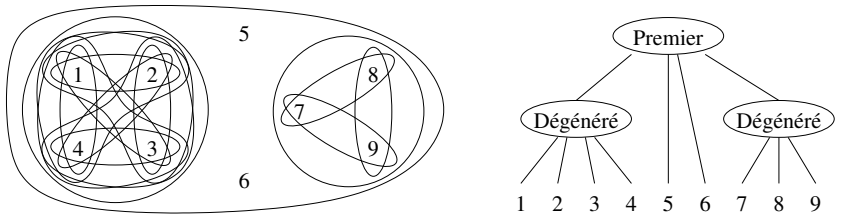

Fig. 2.1. L'arbre d'inclusion des éléments forts de la famille $\mathcal{S} = \{\{1,2,3,4,5,6,7,8,9\}, \{1,2,3,4\}, \{1,2,3\}, \{2,3,4\}, \{1,3,4\}, \{1,2,4\}, \{7,8,9\}, \{1,2\}, \{2,3\}, \{3,4\}, \{4,1\}, \{1,3\}, \{2,4\}, \{7,8\}, \{8,9\}, \{7,9\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}\}$

Définition 2.3. *Soit q un nœud de l'arbre $T_{\mathcal{S}}$ ayant k fils $f_1^q, \ldots, f_k^q$. Le nœud q est* dégénéré *si pour tout $J \subset [1,k]$, $J \neq \emptyset$, $\cup_{j \in J} f_j^q \in \mathcal{S}$. Un nœud non dégénéré est* premier.

Théorème 2.4. *[CHM81] Tout élément $A \in \mathcal{S}$ est soit forte, soit l'union d'un sous-ensemble strict des fils d'un nœud dégénéré de $T_{\mathcal{S}}$.*

Le Théorème 2.4 nous indique que cet arbre $T_{\mathcal{S}}$ est suffisant pour représenter l'ensemble de la famille $\mathcal{S}$ et que les éléments forts de $\mathcal{S}$ forment une "base" de $\mathcal{S}$. Autre conséquence intéressante, toute famille partitive (qui peut être de taille exponentielle) admet une représentation de taille $O(n)$: en effet, il suffit pour cela de coder l'arbre $T_{\mathcal{S}}$ et les ensembles forts associés à chacun de ses nœuds. Un simple parcours de $T_{\mathcal{S}}$ (par exemple un parcours postfixe) permet d'ordonner les éléments de S de sorte de chaque ensemble fort soit codé par un intervalle de cet ordre. La notion de permutation factorisante présentée ci-dessous paraît donc naturelle.

2.2 Permutations factorisantes

Ce n'est que très récemment que le concept de permutation factorisante a été formalisé [Cap96]. Il était pourtant déjà sous-jacent à plusieurs articles [HM91, Hsu92, HHS95]. De manière plus surprenante, nous verrons que dans certaines applications (voir Chapitre 8), la donnée est une permutation factorisante de la famille étudiée. Ce concept est, à mon sens, central dans la simplification des algorithmes de calcul de l'arbre $T_{\mathcal{S}}$. Avant de le définir formellement, posons quelques notions utiles.

Une permutation σ d'un ensemble S à n éléments est définie comme un ordre total sur S ou, de manière équivalente, comme une bijection entre S et $[1, n]$. Soient $i \in [1, n]$ et $x \in S$, alors $\sigma(x)$ désigne le rang i de x dans la permutation σ et $\sigma^{-1}(i)$ désigne le i-ème élément x de S dans σ.

Un ensemble $I \subseteq S$ est un *facteur* ou *intervalle* d'une permutation σ s'il existe $i \in [1, n]$ et $j \in [1, n]$ tel que $I = \{x \ : \ x = \sigma^{-1}(k), i \leqslant k \leqslant j\}$. Moins formellement, I est un ensemble d'éléments consécutifs dans σ.

Définition 2.5. *[Cap96] Soit $\mathcal{P}$ une famille (faiblement) partitive d'un ensemble S et soit $\mathcal{F}$ l'ensemble des éléments forts de $\mathcal{P}$. Une permutation σ de S est une* permutation factorisante *pour $\mathcal{P}$ si pour tout $F \in \mathcal{F}$, F est un facteur de σ.*

Par exemple, $\pi = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9$ mais aussi $\pi_1 = 6\ 5\ 2\ 1\ 4\ 3\ 7\ 9\ 8$ et $\pi_2 = 9\ 8\ 7\ 5\ 1\ 3\ 2\ 4\ 6$ sont des permutations factorisantes pour la famille $\mathcal{S}$ décrite Figure 2.1. Dans ces trois permutations, les deux ensembles forts non-triviaux $\{1, 2, 3, 4\} \in \mathcal{S}_F$ et $\{7, 8, 9\} \in \mathcal{S}_F$ sont effectivement des intervalles.

Observation 2.6 *Si $\mathcal{S}_F$ ne contient que des éléments triviaux, alors toute permutation de S est factorisante.*

2.3 Modules d'un graphe

Sauf indication contraire, nous ne considérons que des graphes non-orientés, simples et sans boucle. Nous utilisons les notations classiques. Le voisinage dans un graphe $G = (V, E)$ d'un sommet x est noté $N_G(x)$ et son non-voisinage $\overline{N}_G(x)$. L'indice G n'est pas utilisé si le contexte est non-ambigü. Le graphe complémentaire de G est noté $\overline{G}$. Etant donné un sous-ensemble de sommets $X \subseteq V$, $G[X]$ désigne le sous-graphe induit par X (toute arête entre deux sommets de X est conservée).

Soient M un ensemble de sommets d'un graphe $G = (V, E)$ et x un sommet de $V \setminus M$. Le sommet x est un *casseur*[1] pour M s'il existe $y \in M$ et $z \in M$ tels que $xy \in E$ et $xz \notin E$. Si x n'est pas un casseur pour M alors M est *uniforme* ou *homogène* par rapport à x.

[1] Le terme anglais utilisé est *splitter*. Nous préfèrerons *casseur* à coupeur pour éviter la confusion avec la notion de coupe (voir Section 2.6.2).

Définition 2.7. *Soit $G = (V, E)$ un graphe. Un ensemble $M \subseteq V$ sommets est un* module *si pour tout $x \notin M$, $M \subseteq N(x)$ ou $M \cap N(x) = \emptyset$.*

Un module est donc un ensemble M uniforme par rapport à tout sommet $x \notin M$.

Lemme 2.8. *[CHM81] La famille $\mathcal{M}$ des modules d'un graphe est une famille partitive.*

Les ensembles V et $\{x\}$, tels que $x \in V$ sont les modules triviaux. Un graphe est *premier* s'il ne possède aucun module non-trivial. Un *module fort* de G est un élément fort de la famille $\mathcal{M}$ des module de G. Rappelons qu'un *module fort* ne chevauche aucun autre module (voir Section 2.1). Ainsi lorsque M et M' sont des modules se chevauchant, alors $M \setminus M'$, $M' \setminus M$, $M \cap M'$, $M \cup M'$ et $M \Delta M'$ sont aussi des modules.

Soient M et M' deux modules disjoints. M et M' sont *adjacents* si tout sommet de M est adjacent à tout sommet de M' et *non-adjacents* si tout sommet de M est non-adjacent à tout sommet de M'.

Observation 2.9 *Toute paire M et M' de modules disjoints est soit adjacente soit non-adjacente.*

Un module M est *maximal* pour un ensemble S de sommets, si $M \subset S$ et s'il n'existe pas de module $M' \subset S$ tel que $M \subset M'$. Par défaut, si l'ensemble S n'est pas spécifié, nous supposerons qu'il s'agit de l'ensemble des sommets V.

Définition 2.10. *Soit $\mathcal{P} = \{M_1, \ldots, M_k\}$ une partition de l'ensemble des sommets d'un graphe $G = (V, E)$. Si pour tout i, $1 \leqslant i \leqslant k$, M_i est un module de G, alors $\mathcal{P}$ est une* partition modulaire *de G.*

Notons que par définition, tout sommet appartient à un unique module fort maximal. Il existe donc une unique partition des sommets en modules forts maximaux.

Définition 2.11. *La* partition modulaire maximale *d'un graphe $G = (V, E)$ est l'unique partition modulaire $\mathcal{P} = \{M_1, \ldots, M_k\}$ ne contenant que des modules forts maximaux.*

Observation 2.12 *Soit $\mathcal{P}$ une partition modulaire d'un graphe G et soit M un module fort de G contenu dans aucun module de $\mathcal{P}$. Alors M est l'union disjointe de modules de $\mathcal{P}$.*

D'après l'Observation 2.9, il est possible de définir un graphe, appelé *graphe quotient*, dont les sommets sont les parties (ou modules) appartenant à la partition modulaire $\mathcal{P}$.

Définition 2.13. *Soit* $\mathcal{P} = \{M_1, \ldots, M_k\}$ *une partition modulaire d'un graphe* $G = (V, E)$*. Le* graphe quotient $G_{/\mathcal{P}}$ *a un ensemble de sommets en bijection avec les modules* $M_i \in \mathcal{P}$*. Deux sommets* v_i *et* v_j *de* $G_{/\mathcal{P}}$ *sont adjacents si les modules correspondants* M_i *et* M_j *sont adjacents dans* G*.*

L'opération inverse au quotient est la *substitution*. Elle consiste à remplacer un sommet x de G par un graphe $H = (V', E')$. Le graphe résultant de l'opération est :

$$G_{x \to H} = ((V \setminus \{x\}) \cup V', (E \setminus \{xy \in E\}) \cup E' \cup \{yz : xy \in E \text{ et } z \in V'\})$$

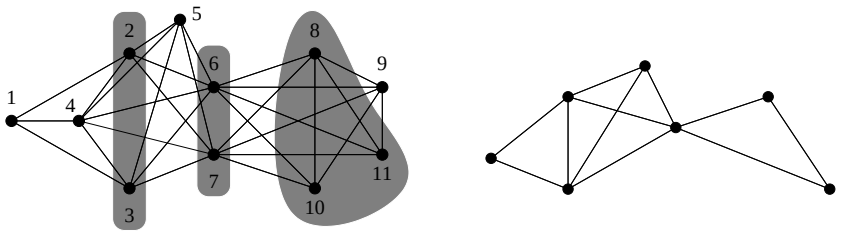

Fig. 2.2. Les ensembles grisés sont des modules du graphe G décrit. $\mathcal{Q} = \{\{1\}, \{2,3\}, \{4\}, \{5\}, \{6,7\}, \{9\}, \{8,10,11\}\}$ est une partition modulaire de G. Le graphe quotient $G_{/\mathcal{Q}}$ est présenté à droite. La partition modulaire maximale de G est $\mathcal{P} = \{\{1\}, \{2,3,4\}, \{5\}, \{6,7\}, \{8,9,10,11\}\}$. Le graphe quotient associé à $\mathcal{P}$ est présenté Figure 2.3.

Remarquons que le graphe quotient $G_{/\mathcal{P}}$ avec $\mathcal{P} = \{M_1, \ldots, M_k\}$ est isomorphe à tout sous-graphe induit $G[V']$ par un sous-ensemble $V' \subseteq V$ de sommets tel que $\forall i \in [1, k]$, $|M_i \cap V'| = 1$. Ainsi plus généralement, quotienter un graphe G par un module M consiste à contracter le module M en un seul sommet. Le graphe résultant est noté $G_{/M}$. Le *graphe représentatif* d'un module M est le graphe quotient $G[M]_{/\mathcal{P}}$ où $\mathcal{P}$ est la partition modulaire maximale de $G[M]$ (c'est donc aussi le sous-graphe obtenu en sélectionnant un sommet par module fort maximal de $G[M]$).

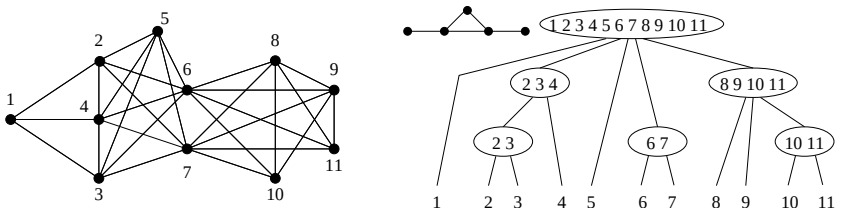

Fig. 2.3. L'arbre $T_{\mathcal{M}}$ des modules forts de G. Le graphe représentatif associé à la racine est $G_{/\mathcal{P}}$ avec $\mathcal{P} = \{\{1\}, \{2,3,4\}, \{5\}, \{6,7\}, \{8,9,10,11\}\}$.

L'arbre $T_{\mathcal{M}}$ dont les nœuds sont les modules forts de G permet de représenter entièrement le graphe G, à condition de lui associer à chacun de ses nœuds le graphe représentatif du module correspondant. Sur la Figure 2.3, les graphes représentatifs des nœuds autres que la racine sont des *cliques* ou des *stables*.[2]

Le problème de la décomposition modulaire d'un graphe consiste à calculer cet arbre $T_{\mathcal{M}}$. Nous verrons que le *théorème de décomposition modulaire* (Théorème 2.16) décrit naturellement un algorithme pour ce problème. Nous présentons préalablement deux propriétés sur les partitions modulaires et les graphes quotients qui permettront d'obtenir des algorithmes plus efficaces (voir Chapitre 4).

Lemme 2.14. *[Möh85b] Soit $\mathcal{P}$ une partition modulaire d'un graphe $G = (V, E)$. Alors $\mathcal{X} \subseteq \mathcal{P}$ est un module de $G_{/\mathcal{P}}$ ssi $\bigcup_{M \in \mathcal{X}} M$ est un module de G.*

Ce lemme peut être renforcé pour observer la même correspondance entre les modules forts de G et ceux de $G_{/\mathcal{P}}$.

Lemme 2.15. *Soit $\mathcal{P}$ une partition modulaire d'un graphe $G = (V, E)$. Alors $\mathcal{X} \subset \mathcal{P}$ est un module* fort *non-trivial de $G_{/\mathcal{P}}$ ssi $\bigcup_{M \in \mathcal{X}} M$ est un module* fort *non-trivial de G.*

Preuve. Soit $\mathcal{X} \subset \mathcal{P}$ un module de $G_{/\mathcal{P}}$. Notons $X = \bigcup_{M \in \mathcal{X}} M$ le module correspondant de G (Lemme 2.14). Par hypothèse $\mathcal{X}$ est non-trivial, il contient donc au moins deux modules de G appartenant à $\mathcal{P}$.

$\Leftarrow$ Supposons que $\mathcal{X}$ ne soit pas un module fort. Il existe donc $\mathcal{Y} \subseteq \mathcal{P}$, module de $G_{/\mathcal{P}}$ tel que $\mathcal{X} \perp \mathcal{Y}$. D'après le Lemme 2.14, $Y = \bigcup_{M' \in \mathcal{Y}} M'$ est un module de G. De plus, $X \perp Y$, donc $X = \bigcup_{M \in \mathcal{X}} M$ n'est pas un module fort de G.

$\Rightarrow$ Supposons que $X = \bigcup_{M \in \mathcal{X}} M$ ne soit pas un module fort de G. D'après le Théorème 2.4, X est l'union d'un sous-ensemble strict de l'ensemble des modules forts maximaux inclus dans un module fort Z dégénéré de G. Notons que $Z \setminus X$ est aussi l'union disjointes de modules de $\mathcal{P}$. Puisque $|\mathcal{X} \cap \mathcal{P}| > 1$, il existe un module $Y \subset Z$ de G tel que : $Y \perp X$, $Y = \bigcup_{M' \in \mathcal{Y}} M'$ avec $\mathcal{Y} \subset \mathcal{P}$ et $\mathcal{X} \perp \mathcal{Y}$. Par conséquent, d'après le Lemme 2.14, $\mathcal{Y}$ est un module de $G_{/\mathcal{P}}$ et donc $\mathcal{X}$ n'est pas un module fort de $G_{/\mathcal{P}}$.

□

Nous pouvons maintenant décrire le théorème de décomposition modulaire.

Théorème 2.16. *[Gal67, CHM81] Soit $G = (V, E)$ un graphe. Alors l'une des trois conditions suivantes est vérifiée:*

1. *G n'est pas connexe;*
2. *$\overline{G}$ n'est pas connexe;*

[2] La *clique* est le graphe complet, alors que le *stabe* est le graphe sans arête.

3. G et $\overline{G}$ sont connexes et le graphe $G_{/\mathcal{P}}$, où $\mathcal{P}$ est la partition modulaire maximale de G, est un graphe premier.

Il est clair que toute union de composantes connexes d'un graphe G est un module. Il en est de même pour toute union de composantes connexes du graphe complémentaire $\overline{G}$. Dans ce cas, le graphe G est dit *dégénéré* (voir Théorème 2.4).

En particulier, lorsque le graphe G n'est pas connexe, la partition modulaire maximale est la partition en composantes connexes et le graphe quotient associé est un *stable*. La racine de l'arbre $T_{\mathcal{M}}$ est alors étiqueté *parallèle*. La *composition parallèle*, ou *union disjointe*, de k graphes connexes $G_1, \ldots, G_k$ résulte en un graphe dont les composantes connexes sont exactement les graphes $G_1, \ldots, G_k$. Cette opération est généralement notée $G_1 \oplus \cdots \oplus G_k$.

Lorsque le graphe complémentaire $\overline{G}$ n'est pas connexe, le graphe quotient associé à la partition en composantes connexes de $\overline{G}$ (qui est la partition modulaire maximale) est une *clique*. [3]. La racine de $T_{\mathcal{M}}$ est alors étiqueté *série*. La *composition série* de k graphes $G_1, \ldots, G_k$ résulte en un graphe dont les composantes co-connexes sont exactement les graphes $G_1, \ldots, G_k$ (pour toute paire de sommets x, y appartenant à des graphes G_i et G_j différents, l'arête xy a été ajoutée). Cette opération est généralement notée $G_1 \otimes \cdots \otimes G_k$.

Algorithme 1 : Naïf

Données : Un graphe $G = (V, E)$
Résultat : L'arbre de décomposition modulaire $MD(G) = (\mathcal{M}, F)$
début
  si $|V| = 1$ **alors** $MD(G)$ ne contient que la feuille $\{v\}$;
  Soit r la racine de $MD(G)$;
  si *G n'est pas connexe* **alors**
    r est étiquetée *parallèle*;
    pour chaque *composante connexe C* **faire**
      $MD(G[C])$ est un sous-arbre de connexe de $MD(G) - r$
  sinon
    si *$\overline{G}$ n'est pas connexe* **alors**
      r est étiquetée *parallèle*;
      pour chaque *composante connexe C de $\overline{G}$* **faire**
        $MD(G[C])$ est un sous-arbre de connexe de $MD(G) - r$
    sinon
      r est étiquetée *premier*;
      pour chaque *module maximal M de G* **faire**
        $MD(G[M])$ est un sous-arbre de connexe de $MD(G) - r$
fin

[3] Une *clique* est un graphe complet.

Dans tous les cas, on peut directement déduire du Théorème 2.16, un algorithme naif pour le calcul de $MD(G)$ pour le calcul de $T_{\mathcal{M}}$ (voir ci-dessus, Algorithme 1): le Théorème 2.16 est appliqué récursivement sur les modules forts maximaux de G. L'arbre $T_{\mathcal{M}}$ dont les nœuds sont étiquetés parallèle, série ou premier, sera noté $MD(G)$ et appelé *arbre de décomposition modulaire*. Notons qu'il n'est pas nécessaire de stocker les graphes quotients des nœuds séries ou parallèles.

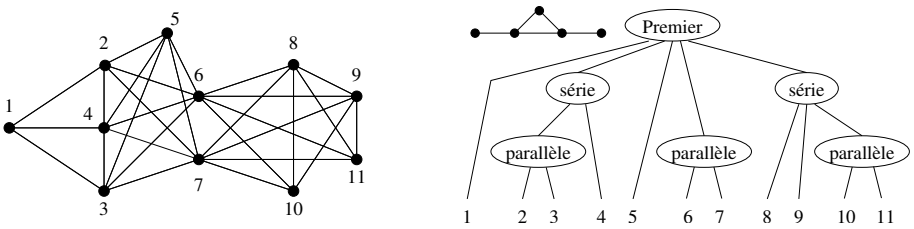

Fig. 2.4. Un graphe et son arbre de décomposition modulaire. Tous les nœuds excepté la racine sont dégénérés. A gauche de la racine est affiché le graphe quotient $G_{/\mathcal{P}}$ avec $\mathcal{P} = \{\{1\}, \{2,3,4\}, \{5\}, \{6,7\}, \{8,9,10,11\}\}$.

Notation 2.17 *Dans la suite, les modules forts correspondants à des nœuds p et q de l'arbre $MD(G)$, seront notés respectivement $M(p)$ et $M(q)$. Pour alléger les notations P et Q pourrons être utilisés.*

Notation 2.18 *Le module fort minimal contenant deux sommets x et y sera noté $m(x,y)$. Par ailleurs, $M(x,\overline{y})$ désignera le module fort maximal contenant le sommet x mais pas le sommet y.*

On constate que le problème principal consiste à calculer la partition modulaire maximale lorsque le graphe et son complémentaire sont connexes. Il n'est pas nécessaire de procéder ainsi. Dès qu'un module non-trivial M est identifié, grâce aux Lemmes 2.14 et 2.15, il suffit de calculer récursivement $MD(G[M])$ et $MD(G_{/M})$ et de reconnecter $MD(G[M])$ sur la feuille de $MD(G_{/M})$ correspondant au sommet représentatif de M. Nous verrons qu'une solution possible pour identifier un module M est de calculer le plus petit module $m(x,y)$ contenant les sommets x et y (voir Théorème 2.28). Ce calcul peut être effectué en $O(n+m)$.

La propriété suivante peut être utile pour l'analyse de la complexité des algorithmes.

Lemme 2.19. *[dM03] Soit $G = (V,E)$ un graphe et soit $\mathcal{F}$ l'ensemble de ses modules forts. Alors*

$$\sum_{F \in \mathcal{F}} |F| \leqslant 2m + 3n \text{ avec } |V| = n, |E| = m$$

2.4 Sur la structure des graphes premiers

La structure des graphes premiers a particulièrement été étudiée. Il est par exemple facile de vérifier que le plus petit graphe premier est le P_4 , le chemin à 4 sommets (voir Figure 2.5). Comme le montre le résultat suivant, les P_4 ont un rôle fondamental dans la structure des graphes premiers.

Lemme 2.20. *[CI98] Soit G un graphe premier. Alors par tout sommet, sauf au plus un, il passe un P_4. Un tel sommet est appelé le "*nez du taureau*" (voir Figure 2.5).*

Fig. 2.5. Le P_4 à gauche et le taureau à droite. Les sommets a et d sont les *extrémités* du P_4, b et c sont ses *milieux*. Le sommet x est le "nez du taureau".

Le Théorème 2.16 peut être étendu en considérant la structure des graphes premiers [JO95b]. Un ensemble de sommets C d'un graphe $G = (V, E)$ est *P-connexe* si pour toute bipartition $\{A, B\}$ de C, il existe un P_4 contenant des sommets de A et de B. Ce n'est par exemple pas le cas du taureau. Une *composante P-connexe* est un ensemble P-connexe maximal. Il faut noter que les composantes P-connexes d'un graphe définissent une partition de l'ensemble de sommets. Une composante P-connexe H est *séparable* si elle admet une bipartition (H_1, H_2) telle que tout P_4 intersectant à la fois H_1 et H_2, a ses extremités dans H_1 et ses milieux dans H_2.

Théorème 2.21. *[JO95b] Soit $G = (V, E)$ un graphe. Alors l'une des quatre conditions suivantes est vérifiée:*

1. *G n'est pas connexe;*
2. *$\overline{G}$ n'est pas connexe;*
3. *G est P-connexe;*
4. *il existe une unique composante P-connexe séparable H avec une bipartition (H_1, H_2) telle que pour tout sommet $x \notin H$, $H_1 \subseteq N(x)$ et $H_2 \cap N(x) = \emptyset$.*

2.5 Quelques familles de graphes

2.5.1 Les cographes

La première famille à considérer pour la décomposition modulaire est certainement la famille $\mathcal{F}$ des graphes totalement décomposables. C'est à dire

les graphes dont l'arbre de décomposition modulaire ne contient pas de nœud premier. Par définition, tout graphe de $\mathcal{F}$ peut être obtenu par compositition série et union disjointe de graphes appartenant à $\mathcal{F}$. Le graphe réduit à un seul sommet est le plus petit graphe de $\mathcal{F}$. Par conséquent, il faut noter que si G appartient à $\mathcal{F}$, alors son complémetaire $\overline{G}$ aussi. Cette famille de graphes s'appelle les *cographes*, pour *complement reducible graphs* [CLSB81, Sum73]. On déduit aussi du théorème de caractérisation suivant que la famille des cographes est *héréditaire* (tout sous-graphe induit d'un cographe est aussi un cographe).

Théorème 2.22. *[Sum73] La famille des cographes est exactement la famille des graphes sans P_4 induit.*

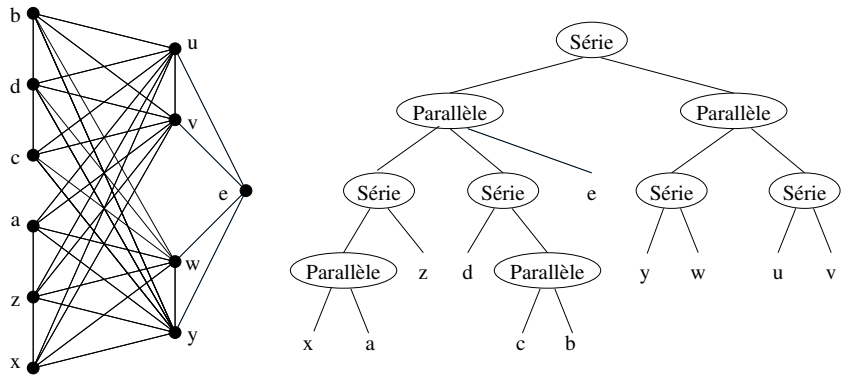

Fig. 2.6. Un cographe et son arbre de décomposition modulaire (ou coarbre).

La famille des cographes a été étudiée et redécouverte dans de nombreux contextes différents (voir [Sum73, CLSB81, CPS85] pour des références). Le premier algorithme linéaire de reconnaissance est dû à Corneil, Perl et Stewart [CPS85]. Cet algorithme construit de manière incrémentale l'arbre de décomposition modulaire, appelé *coarbre* pour les cographes. Probablement à cause du fait que la reconnaissance des cographes constitue le cas de base de la décomposition modulaire, de nombreux autres algorithmes linéaires de reconnaissance [Dah95a, HP05, BCHP03] ont été découverts depuis. Nous en présenterons dans le but de simplifier les explications pour les algorithmes de décomposition modulaire d'un graphe quelconque.

Enonçons quelques propriétés importantes des cographes. On déduit facilement, de la définition des cographes, l'observation suivante.

Observation 2.23 *Soit $MD(G)$ l'arbre de décomposition (ou coarbre) d'un cographe. Alors le père de tout nœud série est parallèle et le père de tout nœud parallèle est série.*

Les propriétés énoncées dans le Lemme suivant sont classiques. Les prouver permet de se familiariser avec la famille des cographes.

Lemme 2.24. *Soient x, y et v trois sommets d'un cographe $G = (V, E)$.*

1. *Si $xv \in E$, $yv \notin E$ et $xy \in E$, alors $m(v, y) \subseteq M(v, \overline{x})$*
2. *Si $xv \in E$, $yv \notin E$ et $xy \notin E$, alors $m(v, x) \subseteq M(v, \overline{y})$*
3. *Si $xv \in E$, $yv \in E$ et $xy \notin E$, alors $M(v, \overline{x}) = M(v, \overline{y})$ et $m(v, x) = m(v, y)$*
4. *Si $xv \notin E$, $yv \notin E$ et $xy \in E$, alors $M(v, \overline{x}) = M(v, \overline{y})$ et $m(v, x) = m(v, y)$*

Preuve. Nous ne ferons que les preuves des propriétés (1) et (3). Les deux autres sont similaires. Notons que par définition, les modules forts $m(v, x)$ et $m(v, y)$ sont comparables pour l'inclusion puisqu'ils s'intersectent. Il en est de même pour $M(v, \overline{x})$ et $M(v, \overline{y})$.

(1) Remarquons tout d'abord que $y \in m(v, x)$ car y casse x et v. Nous avons donc l'inclusion $m(v, y) \subseteq m(v, x)$. Supposons que $m(v, y) = m(v, x)$. Alors x, y et v sont dans trois modules forts maximaux $m(v, x) = m(v, y)$. Le sous-graphe induit par ces trois sommets doit donc être soit une clique, soit un stable. Ce n'est par hypothèse pas le cas : contradiction. Donc nécessairement $m(v, y) \subsetneq m(v, x)$. On en déduit que $m(v, y) \subseteq M(v, \overline{x})$.

(3) C'est un corollaire direct de la propriété (1). Il suffit d'inverser les rôles de x et v pour constater que $y \in M(x, \overline{v})$. De manière symétrique, on a $x \in M(y, \overline{v})$ et donc $M(x, \overline{v}) = M(y, \overline{v})$ et $m(v, x) = m(v, y)$.

□

2.5.2 Les graphes P_4-creux

Les graphes P_4-creux sont une généralisation des cographes : la présence de P_4 est possible mais réduite. Un graphe est P_4-*creux* si tout sous-graphe induit par 5 sommets contient au plus un P_4. On déduit facilement de cette définition un ensemble de sept sous-graphes exclus (voir Figure 2.7).

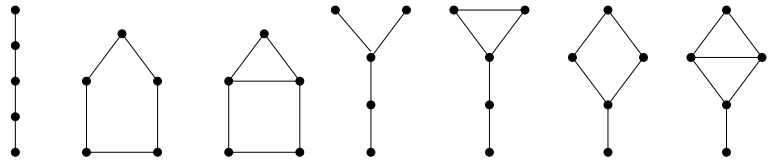

Fig. 2.7. Les 7 sous-graphes exclus des graphes P_4-creux.

Théorème 2.25. *[JO92b] Un graphe est P_4-creux ssi il ne possède aucun des graphes de la figure 2.7 comme sous-graphe induit.*

Pourtant ce n'est pas la caractérisation en sous-graphes exclus qui permet de reconnaître les graphes P_4-creux en temps linéaire [JO92a], mais la structure de leur arbre de décomposition modulaire [JO92b] et en particulier des nœuds premiers.

Une *araignée* (voir Figure 2.8) est un graphe $G = (V, E)$ tel que V admet une tripartition $\{S, K, R\}$ (avec R possiblement vide) tel que : i) $|S| = |K| \geqslant 2$ et S est un stable, K est une clique; ii) pour tout sommet $x \in R$, $K \subseteq N(x)$ et $S \cap N(x) = \emptyset$; iii) il existe un couplage $f : S \to K$ tel que soit pour chaque sommet $v \in S$ (cas de l'araignée *fine*), $N(v) = \{f(v)\}$ soit pour chaque sommet $v \in S$, $N(v) = K \setminus \{f(v)\}$ (cas de l'araignée *épaisse*).

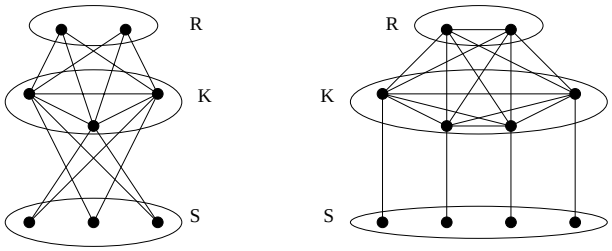

Fig. 2.8. Une araignée *épaisse* à gauche et *fine* à droite.

Théorème 2.26. *[JO92b, JO92a] Un graphe est P_4-creux ssi tout noeud premier p de son arbre de décomposition modulaire $MD(G)$ vérifie:*

1. *le graphe représentatif de p est une araignée dont les sommets se partitionnent en $\{S, K, R\}$;*
2. *tous les fils de p correspondants aux sommets de S et K sont des feuilles de $MD(G)$.*

Donc comme pour les cographes, les graphes P_4-creux se distinguent comme des graphes ayant un arbre de décomposition avec une topologie contrainte : parmi les fils d'un nœud premier, au plus un fils n'est pas une feuille.

Il faut aussi mentionner l'existence de toute une série de familles de graphes généralisant les cographes et les graphes P_4-creux. Les graphes P_4-réductibles [JO95a] forment l'une d'entre elles (voir [BLS99] pour plus d'information à ce sujet).

2.5.3 Les graphes de comparabilité

De nombreuses familles $\mathcal{F}$ de graphes peuvent être caractérisées par la propriété que tout graphe représentatif associé à un nœud premier de l'arbre de décomposition appartient à $\mathcal{F}$. Le problème de la reconnaissance se réduit alors à la reconnaissance des graphes premiers de la famille. C'est par exemple le cas des *graphes de permutation*, des *graphes d'intervalles*, des *distance*

héréditaire... mais aussi des *graphes de comparabilité*(voir [BLS99] pour une présentation de ces familles). Nous présentons ici les forts liens qui unissent les graphes de comparabilité à la décomposition modulaire.

Un graphe $G = (V, E)$ est de *comparabilité*[4] ssi les arêtes de E peuvent être orientées transitivement. Un graphe orienté est *transitif* si pour tout triplet de sommets x, y, z, l'existence des arcs $\overrightarrow{xy}$ et $\overrightarrow{yz}$ implique l'existence de l'arc $\overrightarrow{xz}$.

On définit une relation de forçage Γ sur l'orientation des arêtes d'un graphe $G = (V, E)$ de la manière suivante [Gal67] :

$$\overrightarrow{xy}\ \Gamma\ \overrightarrow{uv} \text{ ssi } \begin{cases} \text{soit } x = u \text{ et } yv \notin E \\ \text{soit } y = v \text{ et } xu \notin E \end{cases} \tag{2.1}$$

La fermeture reflexive et transitive Γ^* de la relation Γ est une relation d'équivalence dont les classes d'équivalence sont appelées *classes de forçage*. Pour une arête xy, nous notons $\Gamma^*(\overrightarrow{xy})$ la classe de forçage contenant l'orientation $\overrightarrow{xy}$ de xy. Soit $\Gamma^{*-1}(\overrightarrow{xy}) = \Gamma^*(\overrightarrow{yx})$. Remarquons que $\Gamma^{*-1}(\overrightarrow{xy}) \cap \Gamma^*(\overrightarrow{xy}) = \emptyset$ ou alors $\Gamma^{*-1}(\overrightarrow{xy}) = \Gamma^*(\overrightarrow{xy})$

L'orientation transitive d'un graphe $G = (V, E)$ s'appuie sur la relation Γ^*. Chaque arête $xy \in E$ peut être orientée soit en $\overrightarrow{xy}$, soit en $\overrightarrow{yx}$. Pour toute arête $yz \in E$ telle que $xz \notin E$, on a alors $\overrightarrow{yz} \in \Gamma^*(\overrightarrow{xy})$ et $\overrightarrow{zy} \in \Gamma^*(\overrightarrow{yx})$. On en déduit la caractérisation suivante des graphes de comparabilité.

Théorème 2.27. *[Gal67] Un graphe $G = (V, E)$ est un graphe de comparabilité ssi pour toute arête $xy \in E$, $\Gamma^*(\overrightarrow{xy}) \cap \Gamma^*(\overrightarrow{yx}) = \emptyset$.*

Ce théorème suggère directement un algorithme de reconnaissance des graphes de comparabilité.

Algorithme 2 : Naïf

Données : Un graphe $G = (V, E)$
Résultat : Vrai ssi G est un graphe de comparabilité
début
  tant que $E \neq \emptyset$ **faire**
    choisir une arête xy quelconque de E;
    si $\Gamma^*(\overrightarrow{xy}) \cap \Gamma^{*-1}(\overrightarrow{xy}) \neq \emptyset$ **alors**
      retourner G *n'est pas de comparabilité*;
    $E \leftarrow E \setminus \{uv \mid \overrightarrow{uv} \in \Gamma^*(\overrightarrow{xy})\}$;
  retourner G *est de comparabilité*;
fin

Le lien le plus intéressant entre les classes de forçage et la décomposition modulaire est sans doute exprimé par le prochain théorème. Soit I une classe de forçage de G. On définit la *couverture* de I comme l'ensemble :

[4] Les graphes de comparabilité sont des graphes non-orientés.

$$C(I) = \{x \in V \mid \exists y \in V \text{ tel que } \overrightarrow{xy} \in C \text{ ou } \overrightarrow{yx} \in C\}$$

Théorème 2.28. *[Gol80] Soit $G = (V, E)$ un graphe. Pour toute classe de forçage I de G, $C(I)$ est un module. De plus, si $I = \Gamma^*(\overrightarrow{xy})$, alors $C(I)$ est le plus petit module $m(x, y)$ contenant x et y.*

Le calcul des classes de forçage d'un graphe $G = (V, E)$ peut donc être utile pour le calcul de l'arbre $MD(G)$ de décomposition modulaire (pas seulement pour les graphes de comparabilité). Par ailleurs, nous avons la propriété suivante.

Lemme 2.29. *[Möh85b] Si un graphe G admet au plus deux classes de forçage, alors tous les modules non-triviaux de G sont des stables.*

Autrement dit les graphes de comparabilité premiers n'admettent que deux orientations transitives dont l'une est obtenue en renversant l'orientation de tous les arcs de l'autre. Finalement les graphes de comparabilités sont caractérisés par les graphes représentatifs de leurs modules premiers.

Théorème 2.30. *[Gal67] Un graphe $G = (V, E)$ est de comparabilité ssi tout graphe représentatif d'un nœud premier de l'arbre de décomposition $MD(G)$ est un graphe de comparabilité.*

Les meilleurs algorithmes pour l'orientation transitive et la reconnaissance des graphes de comparabilité sont basés sur le Théorème 2.30 et le Lemme 2.29. Une étape préliminaire de décomposition modulaire est donc utile [MS94b, MS99]. Cependant, il est possible de simplifier ces algorithmes et d'éviter cette étape préliminaire en utilisant des techniques d'affinage de partition (voir Chapitre 3). Toutefois la complexité devient légèrement moins bonne [HMPV00].

Parmi les graphes de comparabilité remarquables, ceux dont le graphe complémentaire est aussi un graphe de comparabilité, sont importants (des graphes de co-comparabilité). En particulier, restreint à cette famille de graphes, le problème de l'orientation transitive devient linéaire. Cette famille admet une représentation géométrique très intéressante : ce sont les graphes d'intersection de segments compris entre deux droites parallèles. Ils sont appelés *graphes de permutation* à cause de la définition suivante dont découle directement les propriétés discutées ci-dessus. Pour un exposé plus complet, se référer à [Gol80, BLS99, Spi03]. Nous rencontrerons ces graphes au cours des prochains chapitres.

Définition 2.31. *Un graphe $G = (V, E)$ est un* graphe de permutation *ssi il existe deux permutations π et τ de l'ensembles V des sommets telles que $xy \in E$ ssi $\pi(x) < \pi(y)$ et $\tau(y) < \tau(x)$. La paire de permutations (π, τ) est le* réaliseur *de G.*

Par exemple le graphe de la Figure 2.3 est un graphe de permutation dont un réaliseur est donné par les permutations $\pi =$ **1 2 3 4 5 6 7 8 9 10 11** et $\tau =$ **6 7 5 1 4 10 11 9 8 2 3**. Un autre exemple est présenté Figure 2.9

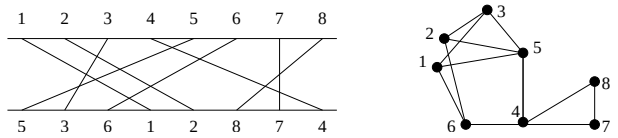

Fig. 2.9. Un graphe de permutation G et son réaliseur (π_1, π_2) avec π_1 l'identitité et $\pi_2 = 4, 2, 5, 1, 6, 3$. Notons que les ensembles $M = \{1, 2\}$ et $M' = \{7, 8\}$ sont des modules. Renverser M ou M' dans π_1 et π_2 produit un autre réaliseur de G.

2.6 Variantes et généralisations

2.6.1 Décomposition modulaire des graphes orientés

La définition de module s'étend naturellement aux graphes orientés. Un ensemble M est un *module orienté* si pour tout $x \notin M$, $M \subseteq N^+(x)$ ou $M \cap N^+(x) = \emptyset$, et pour tout $x \notin M$, $M \subseteq N^-(x)$ ou $M \cap N^-(x) = \emptyset$.

Lemme 2.32. *[MR84] La famille $\mathcal{M}$ des modules orientés d'un graphe orienté est une famille faiblement partitive.*

Un graphe orienté $\overrightarrow{G} = (V, \overrightarrow{E})$ est un *k-ordre* s'il existe une partition de V en $\{V_1, \ldots, V_k\}$ telle que pour tout $x \in V_i$ et $y \in V_j$ tel que $i < j$, alors $\overrightarrow{xy} \in \overrightarrow{E}$ et $\overrightarrow{yx} \notin \overrightarrow{E}$. Les ensembles V_i sont appelées les *composantes ordres* de $\overrightarrow{G}$. Notons que pour tout graphe $\overrightarrow{G}$, il existe un unique k maximum tel que $\overrightarrow{G}$ est un k-ordre.

Observation 2.33 *Soit $\overrightarrow{G} = (V, \overrightarrow{E})$ un k-ordre et $\{V_1, \ldots, V_k\}$ la partition de V en composantes ordres. Alors M est un module ssi il existe i, j tels que $1 \leqslant i < j \leqslant k$, l'ensemble $M = \cup_{h=i}^{j} V_h$.*

Le fait que $\mathcal{M}$ ne soit plus partitive découle de l'observation précédente et implique l'existence d'un quatrième cas pour le théorème de décomposition.

Théorème 2.34. *[MR84] Soit $G = (V, E)$ un graphe. Alors l'une des trois conditions suivantes est vérifiée:*

1. *G n'est pas connexe;*
2. *$\overline{G}$ n'est pas connexe;*
3. *il existe $k > 1$ tel que G est un k-ordre;*
4. *G et $\overline{G}$ sont connexes et le graphe $G_{/\mathcal{P}}$, où $\mathcal{P}$ est la partition en modules maximaux de G est un graphe premier.*

Il existe dans la littérature de nombreuses familles partitives ou faiblement partitives. Une présentation assez exhaustive est proposée dans [dM03].

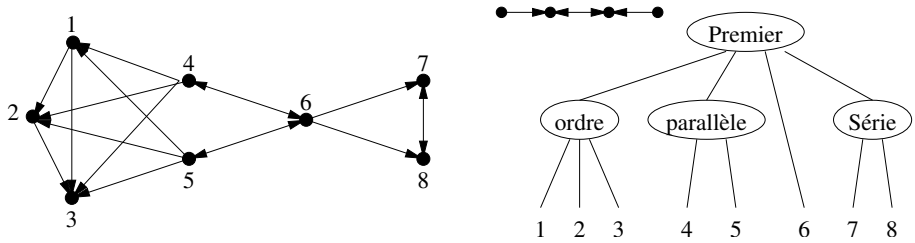

Fig. 2.10. Un graphe orienté et son arbre de décomposition modulaire. Le graphe représentatif de la racine est présenté à sa gauche.

2.6.2 Décomposition en coupes

La décomposition en *coupes complètes* d'un graphe a été introduite [Cun82, CE80] pour les graphes orientés. Mais elle se révèle un outil très puissant pour les graphes non-orientés. En particulier, dans le problème de la reconnaissance des *graphes de cercle* [Bou87, GHS89, MS94a], elle joue un rôle similaire à la décomposition modulaire pour la reconnaissance des graphes de comparabilité. Un graphe de cercle est le graphe d'intersection d'une famille de cordes d'un cercle [BLS99]. En effet, un graphe est un graphe de cercle ssi toutes les composantes de la décomposition en coupes sont des graphes de cercle.

Par ailleurs, la décomposition en coupes complètes joue un rôle primordial dans le problème de la reconnaissance des graphes de *largeur de clique 3* [CHL$^+$00]. La largeur de clique est un paramètre de graphe définissant aussi une décomposition en arbre [CER93].

Définition 2.35. *[Cun82, CE80] Soit $G = (V, E)$ un graphe. Une* coupe complète *de G est une bipartition $\{V_1, V_2\}$ de l'ensemble des sommets V telle que:*

1. $|V_1| \geqslant 2$ *et* $|V_2| \geqslant 2$
2. *soit* $\tilde{V_1} = V_1 \cap N(V_2)$ *et* $\tilde{V_2} = V_2 \cap N(V_1)$*, alors pour tout* $x \in \tilde{V_1}$ *et tout* $y \in \tilde{V_2}$*,* $xy \in E$*.*

Décomposer en coupe complète un graphe consiste à trouver, s'il en existe, une coupe complète $\{V_1, V_2\}$ et décomposer récursivement les deux graphes $G[V_1 \cup \{\tilde{v}_2\}]$, $G[V_2 \cup \{\tilde{v}_1\}]$ où $\tilde{v}_1 \in \tilde{V_1}$ et $\tilde{v}_2 \in \tilde{V_2}$ (voir Figure 2.11).

Clairement le concept de coupe complète peut être vue comme une généralisation de celui de module puisque tout module ayant au moins deux sommets définit une coupe (pour laquelle l'un des deux ensembles $V_1 \setminus \tilde{V_1}$, $V_2 \setminus \tilde{V_2}$ est vide). [5]

Actuellement le meilleur algorithme pour la décomposition en coupe complète est linéaire [Dah00], mais il reste très (trop!) compliqué. L'algorithme

[5] La décomposition en coupes complètes, même si on peut la comprendre comme un généralisation de la décomposition modulaire, relève peut-être plus du paradigme de décomposition par séparation succesives discutée dans l'introduction.

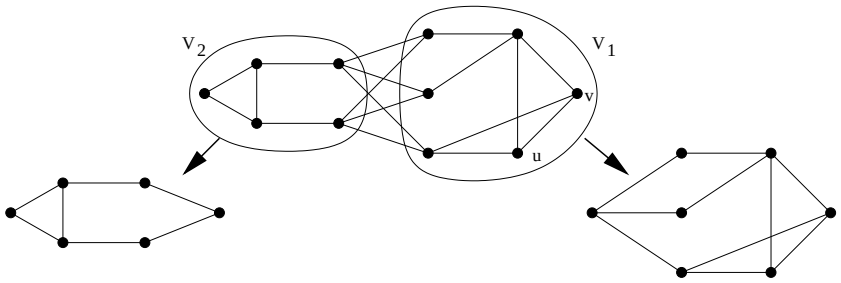

Fig. 2.11. Une coupe $\{V_1, V_2\}$. Notons que $\{u, v\}$ est un module et donc $\{V \setminus \{u, v\}, \{u, v\}\}$ est aussi une coupe complète. La décomposition se poursuit récursivement sur les deux graphes du bas de la figure.

précédent était quadratique [Spi94]. Autant, je pense que l'algorithmique de la décomposition modulaire est aujourd'hui bien comprise (en témoigne le nombre d'algorithmes linéaires), autant ce n'est pas le cas pour la décomposition en coupes complètes. L'algorithme linéaire actuel n'est en aucun cas satisfaisant.

Problème 2.36. Trouver un algorithme simple linéaire, ou sous-quadratique, pour le calcul de la décomposition en coupes complètes.

2.7 Notes bibliographiques

L'article fondateur de la décomposition modulaire des graphes est sans doute celui de Gallai [Gal67] sur l'orientation transitive. A ma connaissance le seul article de synthèse existant sur la décomposition modulaire et plus généralement les familles partitives est celui de Möhring et Radermacher [MR84]. Plus récemment, Ehrenfeucht, Harju et Rozenberg [EHR99] ont publié un livre sur la décomposition des 2-structures (une généralisation des graphes) et proposent une présentation de la décomposition modulaire dans un cadre plus général. Concernant les familles de graphes (graphes de comparabilité, d'intervalle, de permutation, cographes...) se structurant autour de la décomposition modulaire, il faut se référer aux livres de Golumbic [Gol80] et plus récemment à [BLS99, Spi03]. L'aspect algorithme est particulièrement présent dans [Gol80, Spi03].

Tarte au poivron et à l'ail

Ingrédients : 3-4 poivrons rouges, 8 gousses d'ail, coriande fraîche, huile d'olive, une pâte à pizza ou une pâte feuilleté.

Découper les poivrons rouges en large lamelles et les passer au grill (250, environ 10 minutes) et les éplucher. Faire revenir 8 gousses d'ail dans leur peau dans un fond d'huile d'olive (5 minutes). Couvrir le plat à tarte d'une feuille d'alu. Tapisser le fond du plat avec les poivrons. Répartir les gousses d'ail épluchées entre les lamelles de poivrons. Recouvrir la préparation avec la pâte. Mettre au four chaud pendant 25 minutes (250). Démouler la tarte et sopoudrer la coriande coupée finement.

3

Affinage de partition

L'affinage de partition est une technique algorithmique utilisée dans de nombreuses applications parmi lesquelles on peut citer la minimisation d'automates finis déterministes [Hop71]. En 1987, Paigue et Tarjan ont publié un premier article de synthèse sur cette technique [PT87]. Elle a depuis été maintes fois appliquée et mériterait sans doute d'être présente dans les livres d'introduction à l'algorithmique de graphes. Comme nous allons le voir, elle joue un rôle primordial pour la décomposition modulaire. Ce chapitre présente cette technique et l'illustre avec l'exemple du calcul d'une partition modulaire maximale.

3.1 Principe et structure de données

Le principe de l'affinage de partition est très simple, il s'agit de répéter autant de fois que nécessaire une opération, appelé *affinage* : étant donnés une partition $\mathcal{P}$ d'un ensemble V et un sous-ensemble $S \subseteq V$, appelé *ensemble pivot*, il s'agit de calculer la partition maximale $\mathcal{P}'$ plus *fine* que $\mathcal{P}$ *stable* pour S. Soient $\mathcal{P}$ et $\mathcal{P}'$ deux partitions d'un ensemble V, $\mathcal{P}'$ est plus *fine* que $\mathcal{P}$ si toute partie de $\mathcal{P}'$ est contenue dans une partie de $\mathcal{P}$. Si aucune partie de la partition $\mathcal{P}'$ ne chevauche $S \subseteq V$, on dit que $\mathcal{P}$ est *stable* pour S.

Algorithme 3 : *Affiner($\mathcal{P}$, S)*

Données : Une partition $\mathcal{P}$ d'un ensemble V et $S \subseteq V$
Résultat : La partition maximale plus fine que $\mathcal{P}$ stable pour S
début
  pour chaque *partie* $\mathcal{X} \in \mathcal{P}$ **faire**
    si $\mathcal{X} \cap S \neq \emptyset$ *et* $\mathcal{X} \cap S \neq \mathcal{X}$ **alors** remplacer $\mathcal{X}$ par $\mathcal{X} \cap S$ et $\mathcal{X} \setminus S$;
fin

Dans de nombreux algorithmes de graphes, l'ensemble partitionné est l'ensemble des sommets du graphe et les ensembles pivots sont des voisinages de sommets. En général toute la difficulté de l'élaboration d'un algorithme d'affinage de partition réside dans le choix de la partition initiale et des ensembles pivots à utiliser.

Décrivons brièvement la structure de données (voir Figure 3.1). Les éléments de l'ensemble V à partitionner sont stockés dans une liste doublement chaînée. Chaque élément de V possède un pointeur vers la partie qui le contient. Les éléments de chaque partie $\mathcal{X}$ sont consécutifs dans cette liste (ils forment un intervalle). Chaque partie maintient un pointeur vers son premier et son dernier élément dans la liste.

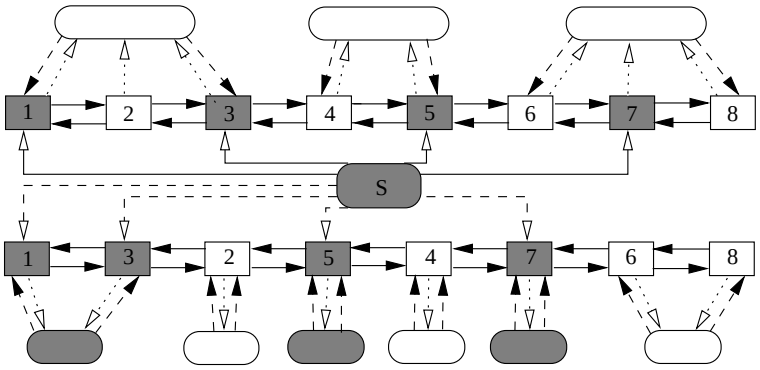

Fig. 3.1. $\mathcal{P}' =$*Affiner*$(\mathcal{P}, S)$.

Notation 3.1 *Cette structure de données décrite manipule donc implicitement des* partitions ordonnées *: les parties sont totalement ordonnées. Cet aspect est ou non important selon les applications. Pour distinguer ces deux cas, une partition ordonnée sera notée* $\mathcal{P} = [\mathcal{X}_1, \ldots, \mathcal{X}_k]$ *et une partition non-ordonnée* $\mathcal{P} = \{\mathcal{X}_1, \ldots, \mathcal{X}_k\}$*. Il pourra être pratique de considérer une partition ordonnée* $\mathcal{P}$ *comme un ordre partiel : chaque partie est une antichaîne et toute paire d'éléments appartenant à deux parties différentes est ordonnée.*

Remarquons qu'il est équivalent d'affiner une partition par un ensemble S ou par son complémentaire $V \setminus S$: *Affiner*$(\mathcal{P}, S)$=*Affiner*$(\mathcal{P}, V \setminus S)$. Il est donc possible de manipuler le graphe complémentaire sans en avoir explicitement stocké les arêtes. Il faut néanmoins, dans le cas des partitions ordonnées, faire attention à l'ordre entre les parties. Ce dernier aspect pourra être spécifié selon les applications : à savoir $\mathcal{X} \cap S$ est placé à gauche ou à droite de $\mathcal{X} \setminus S$.

Lemme 3.2. *La complexité de l'opération* Affiner$(\mathcal{P}, S)$=Affiner$(\mathcal{P}, V \setminus S)$ *est* $O(|S|)$.

Ainsi lorsqu'il est possible de n'utiliser qu'une seule fois le voisinage de chaque sommet comme pivot, on obtient une complexité linéaire, $O(n+m)$, pour l'algorithme. Pour une présentation plus complète de la technique d'affinage de partition et de la structure de donnée nécessaire, il est possible de se référer à [Pau98, HPV98, HPV99, HMPV00].

Lorsque les partitions manipulées sont ordonnées, la manière de placer deux nouvelles sous-parties $\mathcal{X}_a$ et $\mathcal{X}_b$ issue de l'affinage d'une partie $\mathcal{X}$ peut être importante. Nous devons décider selon la situation de leur ordre relatif.

Terminons cette brève présentation par une remarque portant sur l'affinage de partition appliquée aux graphes. Il s'agit souvent de partitionner les sommets pour obtenir soit un ordre total, soit une partition en classe d'équivalence (par exemple chaque partie peut contenir les ensembles de sommets deux à deux jumeaux). Pour cela, les ensembles pivots utilisés sont les voisinages des sommets. En enrichissant la structure de données, McConnell et Spinrad [MS00] montrent comment il est possible à chaque étape d'affinage, sans surcoût de complexité, d'extraire les arêtes incidentes à des sommets n'appartenant pas à la même partie. Pour cela, un tri par "paquets" sur les arêtes est nécessaire. Si m' est le nombre des arêtes extraites, le coût de l'opération est $O(m')$. Cette extraction permettra notamment de calculer efficacement le graphe quotient associé à une partition modulaire.

3.2 Calcul d'une partition modulaire et règle de Hopcroft

L'affinage de partition est l'outil naturel pour le calcul d'une partition modulaire. Nous allons montrer dans la suite que certaines partitions modulaires sont d'une grande utilité pour obtenir des algorithmes efficaces de décomposition modulaire. Dans un premier temps, nous nous intéressons au problème du calcul de la *partition modulaire maximale* (voir Definition 3.3). Nous présentons un algorithme de complexité $O(n+m\log n)$. Cet algorithme qui nous permet d'introduire la *règle de Hopcroft*. Il sera la base d'algorithmes de décomposition modulaire quasi-linéaires très simples (Voir Chapitre 4 et Chapitre 5).

Définition 3.3. *Soit une partition $\mathcal{P}$ des sommets d'un graphe $G=(V,E)$. La* partition modulaire maximale pour $\mathcal{P}$ et G *est la partition $\mathcal{Q}$ plus fine que $\mathcal{P}$ dont toute partie $\mathcal{X}$ est un module de G ne chevauchant aucune partie de $\mathcal{P}$ et maximal pour cela.*

L'idée principale est la suivante : tant qu'il est possible de trouver une partie $\mathcal{X}$ qui n'est pas uniforme par rapport à un sommet $x \notin \mathcal{X}$, la partition courante $\mathcal{P}$ est affinée avec le voisinage $N(x)$. L'algorithme terminera donc lorsque toute partie est un module. La difficulté consiste donc à trouver à chaque étape un sommet x suceptible de casser une partie de $\mathcal{P}$. C'est un

problème difficile si on souhaite obtenir une complexité linéaire, car il ne faut utiliser qu'un nombre constant de fois le voisinage de chaque sommet. Ce premier algorithme garantit que chaque voisinage intervient au plus $\log n$ fois. Pour cela, nous utilisons la propriété suivante découlant directement de la définition d'un module.

Lemme 3.4. *Soient $\mathcal{P}$ une partition des sommets du graphe $G = (V, E)$ et $\mathcal{X}$ une partie de $\mathcal{P}$. Si pour tout sommet $y \notin \mathcal{X}$, $\mathcal{P}$ est stable pour $N(y)$, alors $\mathcal{X}$ est un module de G et pour $x \in \mathcal{X}$ quelconque, la partition $\mathcal{Q} = Affiner(\mathcal{P}, N(x))$ est stable pour $N(x')$, $\forall x' \in \mathcal{X}$.*

Il est donc possible de n'utiliser que les sommets de toutes les parties de $\mathcal{P}$ sauf une. Il est avantageux de choisir la plus grande partie pour cette dernière. De même, lorsque les sommets d'une partie $\mathcal{X}$ ont été utilisés, il est possible que $\mathcal{X}$ soit coupée en deux. Il faut donc poursuivre le travail localement sur le sous-graphe induit par $\mathcal{X}$. De même, la plus grande sous-partie de $\mathcal{X}$ peut être mise de côté pour n'utiliser qu'un seul de ses sommets. C'est la **règle de Hopcroft** [1], aussi appelée *règle de la plus petite moitié*.

Pour gérer ces deux "catégories" de parties, nous utiliserons deux listes K et L. Les voisinages de tous les sommets d'une partie extraite de L seront utilisés, alors que le voisinage d'un unique sommet sera nécessaire pour les familles extraites de K. La gestion de la liste K sera FIFO, c'est à dire que l'extraction se fait en tête de liste et l'ajout en fin de liste. Ainsi seulement nous pourrons garantir que la partie extraite est un module.

Théorème 3.5. *Soit $\mathcal{P}$ une partition des sommets d'un graphe $G = (V, E)$. L'algorithme 4 calcule la partition modulaire pour G et $\mathcal{P}$ en temps $O(n + m \log n)$.*

Preuve. Pour démontrer la correction de l'algorithme, nous prouvons les invariants suivants :

- *Si M est un module de G contenu dans une partie $\mathcal{X} \in \mathcal{P}$, alors à chaque étape, il existe une partie $\mathcal{Y}$ de la partition courante contenant M :* Par contradiction. Considérons la première étape à laquelle M n'est pas contenu dans une partie de la partition. Donc M intersecte deux parties $\mathcal{Y}_1$ et $\mathcal{Y}_2$ résultantes de l'affinage d'une partie $\mathcal{Y}$ par le voisinage $N(x)$ d'un sommet $x \notin \mathcal{Y}$. Puisque $M \subseteq \mathcal{Y}$, $x \notin M$ et donc x est un casseur pour M qui ne peut pas être un module.
- *Si $L = \emptyset$, alors la partie $\mathcal{Y}$ en tête de K est un module* : Par induction. Supposons que $\mathcal{Y}_1$ soit la première partie extraite de K. Notons que $\mathcal{Y}_1$ est incluse dans $\mathcal{Z}$, la plus grande partie de la partition $\mathcal{P}$. Soit $\mathcal{X} \neq \mathcal{Y}_1$ une partie de la partition courante. Que $\mathcal{X}$ soit ou non incluse dans $\mathcal{Z}$,

[1] La règle de Hopcroft est ainsi appelée car elle a été, à notre connaissance, utilisée pour la première fois par Hopcroft dans l'algorithme de minimisation d'automates finis déterministes [Hop71]

Algorithme 4 : Partition modulaire

Données : Une partition $\mathcal{P}$ de l'ensemble V des sommets d'un graphe G
Résultat : La partition modulaire maximale $\mathcal{Q}$ plus fine que $\mathcal{P}$
début
  $\mathcal{Q} \leftarrow \mathcal{P}$;
  Soit $\mathcal{Z}$ la plus grande partie de $\mathcal{P}$;
  $K \leftarrow \{\mathcal{Z}\}$;
  $L \leftarrow \{\mathcal{X} \mid \mathcal{X} \neq \mathcal{Y}, \mathcal{X} \in \mathcal{P}\}$;
  tant que $L \cup K \neq \emptyset$ **faire**
    si *il existe* $\mathcal{X} \in L$ **alors** $S \leftarrow \mathcal{X}$ et $L \leftarrow L \setminus \{\mathcal{X}\}$;
    sinon
1 Soient $\mathcal{X}$ la première partie de K et $x \in \mathcal{X}$;
      $S \leftarrow \{x\}$ et $K \leftarrow K \setminus \{\mathcal{X}\}$;
    pour chaque *sommet* $x \in S$ **faire**
      pour chaque *partie* $\mathcal{Y} \neq \mathcal{X}$ *telle que* $N(x) \perp \mathcal{Y}$ **faire**
2 Remplacer dans $\mathcal{Q}$, $\mathcal{Y}$ par $\mathcal{Y}_1 = \mathcal{Y} \cap N(x)$ et $\mathcal{Y}_2 = \mathcal{Y} \setminus N(x)$;
        Soit $\mathcal{Y}_{min}$ (resp. $\mathcal{Y}_{max}$) la partie de plus petite (resp. grande) taille parmi $\mathcal{Y}_1$ et $\mathcal{Y}_2$;
        si $\mathcal{Y} \in L$ **alors** $L \leftarrow L \cup \{\mathcal{Y}_{min}, \mathcal{Y}_{max}\} \setminus \{\mathcal{Y}\}$;
        sinon
          $L \leftarrow L \cup \{\mathcal{Y}_{min}\}$;
          si $\mathcal{Y} \in K$ **alors** Remplacer $\mathcal{Y}$ par $\mathcal{Y}_{max}$ dans K;
          sinon Ajouter $\mathcal{Y}_{max}$ à la fin de K;
fin

L a contenu une partie $\tilde{\mathcal{X}} \supseteq \mathcal{X}$. Comme $L = \emptyset$, pour tout sommet $x \notin \mathcal{Y}_1$, la partition courante est stable pour $N(x)$. Donc $\mathcal{Y}_1$ est un module de G. Supposons l'invariant vrai pour les i premières parties $\mathcal{Y}_1, \ldots, \mathcal{Y}_i$ extraites de K et montrons que la partie extraite $\mathcal{Y}$ suivante est aussi un module. Notons tout d'abord que puisque, pour tout $j \in [1, i]$, $\mathcal{Y}_j$ est un module, la partition courante est stable pour $N(y_j)$, pour tout $y_j \in \mathcal{Y}_j$. L'argument utilisé pour $\mathcal{Y}_1$ s'applique donc récursivement pour $\mathcal{Y}$ et la partition courante privé des parties $\mathcal{Y}_1, \ldots, \mathcal{Y}_i$.

- *Si la partition courante contient une partie $\mathcal{X}$ qui n'est pas un module, alors il existe $\mathcal{Y} \in L \cup K$, différente de $\mathcal{X}$, contenant un casseur y pour $\mathcal{X}$* : Supposons par contradiction que $\mathcal{Y} \notin L \cup K$. Alors la dernière fois que $N(y)$ a été utilisé pour affiner la partition, $\mathcal{X}$ et $\mathcal{Y}$ étaient contenues dans une même partie $\mathcal{W}$. Or lorsque $\mathcal{X}$ et $\mathcal{Y}$ ont été séparées, elles ont été ajoutées à $L \cup K$. Puisque $\mathcal{Y} \notin L \cup K$, le voisinage de y (ou d'un sommet y' tel que $N(y) \cap \mathcal{X} = N(y') \cap \mathcal{X}$) a été utilisé pour affiner la partition. Donc $\mathcal{X}$ n'appartient plus à la partition courante.

Le premier invariant garantit qu'aucun module contenu dans une classe de $\mathcal{P}$ n'est coupé. Le troisième assure que la partition modulaire $\mathcal{Q}$ retournée par l'algorithme ne contient que des modules.

Sachant que l'opération d'affinage par le voisinage d'un sommet x nécessite un coût $O(d(x))$, l'analyse de la complexité de l'Algorithme 4 se réduit à estimer le nombre de fois que le voisinage d'un sommet peut être utilisé, le coût global de gestion des listes L et K étant $O(n)$. La première observation est que lorsqu'un sommet x d'une partie $\mathcal{X}$ issue de K est utilisé, ni x ni aucun autre sommet de $\mathcal{X}$ ne sera plus utilisé. En effet, les parties extraites de K étant des modules, elles ne sont jamais modifiées dans la suite de l'algorithme. Il reste à estimer le nombre de fois qu'un sommet peut appartenir à une partie de L. Lorsqu'une partie n'appartenant pas à L est coupée, seulement la plus petite moitié est ajoutée à L : un sommet n'apparaît donc au plus que $\log n$ fois dans une partie de L. On en déduit que le voisinage de chaque sommet n'est utilisé qu'au plus $\log n$ fois pour affiner la partition. La complexité de l'Algorithme 4 est donc $O(n + \log n. \sum_{x \in V} d(x)) = O(n + m \log n)$. $\square$

3.3 Notes bibliographiques

Comme nous l'avons mentionné en introduction de ce chapitre, la première utilisation des techniques d'affinage de partition remonte à 1971 pour le problème de la minimisation d'automate fini déterministe [Hop71]. En 1987, Paige et Tarjan [PT87] s'en servent pour trois problèmes : la partition fonctionelle, la partition relationelle maximale, et les ordres doublement lexicographique des matrices. Ce n'est que fin 90 que son utilisation se systématise dans le cadre de la décomposition modulaire et de l'orientation transitive des graphes ([MS00, HMPV00] par exemple). Par ailleurs, cette technique permet d'obtenir pour l'algorithme du parcours en largeur lexicographique, une implémentation extrèmement simple qui, nous le verrons Chapitre 6, autorise de nombreuses variantes.

Il faut toutefois modérer notre enthousiasme. L'affinage de partition couplée à la règle de Hopcroft a permis d'obtenir de très nombreux algorithmes simples et efficaces. Par exemple, dans le cas de la décomposition modulaire, nous le verrons Chapitre 4 et Chapitre 5, elle permet de concevoir sans trop de peine des algorithmes de complexité $O(n+m \log n)$. Pour autant, ces algorithmes restent très loin d'un algorithme linéaire. L'énorme difficulté réside dans le choix du sommet pivot. Pour chaque problème, il faut tout reprendre à zéro et trouver à chaque fois une nouvelle astuce. De ce point de vue, l'affinage de partition reste très "artisanal".

Soupe de melon au basilic

Ingrédients : 2 melons, basilic, poivre, mozzarella et chiffonade de jambon de parme bien sec.

Vider les melons et mixer la pulpe. Découper les feuilles de basilic finement. Ajouter le poivre et le basilic dans la soupe. La mettre au frigo au moins pendant 2 heures. Servir frais avec des petites boules de mozzarela enroulées de jambon de parme.

4

Calcul de l'arbre de décomposition modulaire

En 1994, Ehrenfeucht, Gabow, McConnell et Sullivan [EGMS94] présentent un algorithme quadratique pour la décomposition modulaire des 2-structures. Les 2-structures peuvent être vues comme des graphes orientés complets dont les arcs sont colorés. Cet algorithme est donc linéaire en la taille d'une 2-structure. Par contre, il ne l'est plus lorsqu'on l'utilise pour la décomposition modulaire des graphes. Nous verrons dans ce chapitre que le principe de cet algorithme est pourtant la base de nombreux autres algorithmes efficaces (sous-quadratiques) de décomposition modulaire de graphes [MS00, DGM01]. On peut donc abusivement considérer que cette série d'algorithmes ne constitue qu'une série d'implémentations différentes de l'algorithme de Ehrenfeucht et al., permettant l'amélioration de sa complexité : $O(n + m.\alpha(n, m))$ puis $O(n + m)$ [DGM01], et $O(n + m \log n)$ [MS00].

L'objet de ce chapitre est la présentation de l'algorithmique de Ehrenfeucht et al. et de ses variantes. Dans un premier temps, nous en décrirons le principe sans entrer dans les considérations de complexité. Nous montrerons ensuite comment Dahlhaus, Gustedt et McConnell [DGM01] ont pu rendre cet algorithme linéaire. En fait, pour des raisons de lisibilité, seule leur version en $O(n + m.\alpha(n, m))$ sera détaillée. Enfin nous nous intéresserons à l'alternative proposée par McConnell et Spinrad [MS00] en $O(n + m \log n)$, beaucoup plus simple grâce à l'utilisation de l'affinage de partition.

4.1 L'algorithme de Ehrenfeucht et al.

Commençons par une remarque préliminaire à la description du principe de l'algorithme de Ehrenfeucht et al. Il ne calculera pas exactement $MD(G)$ mais une forme non réduite T de l'arbre $MD(G)$. C'est à dire que T pourra contenir des nœuds séries fils de nœuds séries et des nœuds parallèles fils de nœuds parallèles. L'ensemble des algorithmes décrits dans ce chapitre procèderons de même. Cela n'a aucune conséquence sur la complexité globale puisqu'un simple parcours de cet arbre permettra de le "nettoyer" en temps $O(n)$ en

supprimant les nœuds inutiles. Dans la suite, nous dénommerons abusivement par $MD(G)$ l'arbre calculé par l'algorithme.

L'idée principale utilisée par Ehrenfeucht et al. [EGMS94] est de trouver une sorte de "colonne vertébrale" de l'arbre de décomposition modulaire $MD(G)$: à savoir un chemin entre une feuille de l'arbre de décomposition et sa racine, avec pour chaque nœud interne l'ensemble de ses fils. Ainsi, pour un sommet v fixé, l'algorithme calcule d'une part l'ensemble des modules forts contenant v et d'autre part l'ensemble des modules maximaux ne contenant pas v. Il est alors possible de définir la branche de l'arbre $MD(G)$ comprise entre la racine et la feuille v (voir Figure 4.1). L'algorithme est réitéré récursivement dans chacun des modules non-triviaux ne contenant pas v, ces modules correspondent aux fils des nœuds internes ancêtres de v (voir Algorithme 5).

Définition 4.1. *Soit v un sommet quelconque d'un graphe $G = (V, E)$. Alors on définit la partition modulaire*

$\mathcal{M}(G, v) = \{v\} \cup \{M \mid M$ *est un module maximal ne contenant pas* $v\}$

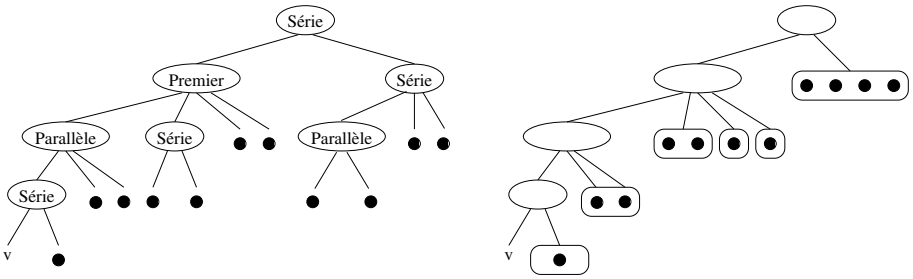

Fig. 4.1. Un arbre de décomposition modulaire $MD(G)$ à gauche; la partition modulaire $\mathcal{M}(G, v)$ et la branche correspondante entre v et la racine de $MD(G)$ à droite.

Pour faire suite à la remarque préliminaire, notons que tout module dégénéré (série ou parallèle) contenant v sera représenté par un nœud binaire dans l'arbre construit par l'algorithme. Le test de la ligne 1 permet la suppression "à la volée" de ces nœuds binaires.

La correction de l'algorithme repose sur la propriété suviante.

Lemme 4.2. *[EGMS94] Soient v un sommet d'un graphe $G = (V, E)$ et $\mathcal{M}(G, v)$ la partition modulaire associée. Alors :*

1. *tout module non-trivial de $G_{/\mathcal{M}(G,v)}$ contient v;*
2. *un ensemble $\mathcal{X} \subset \mathcal{M}(G, v)$ est un module fort non-trivial de $G_{/\mathcal{M}(G,v)}$ ssi $\bigcup_{M \in \mathcal{X}} M$ est un ancêtre de v dans $MD(G)$;*
3. *tout module ne contenant pas v est un sous-ensemble d'une partie $M \in \mathcal{M}(G, v)$.*

Algorithme 5 : Ehrenfeucht et al. [EGMS94]

Données : Un sommet v quelconque de $G = (V, E)$, $T = MD(G_{/\mathcal{M}(G,v)})$ et $\{MD(G[X]) \mid X \in \mathcal{M}(G, v)\}$
Résultat : L'arbre de décomposition modulaire $MD(G) = (\mathcal{M}, F)$
début
  pour chaque *feuille X de T* **faire**
    Soit $T_X = MD(G[X])$ et $p(X)$ le père de X dans T;
    Remplacer X par T_X dans T;
1 **si** *la racine $r(T_X)$ et $p(X)$ sont tous les deux parallèles ou séries*
    alors
      Supprimer $r(T_X)$ et connecter les fils des $r(T_X)$ à $p(X)$
fin

Preuve. 1. Supposons qu'un module non-trivial M de $G_{/\mathcal{M}(G,v)}$ ne contienne pas v. A chaque sommet $x \in M$ correspond un module $M(x) \in \mathcal{M}(G, v)$. D'après le Lemme 2.14, l'ensemble $\cup_{x \in M} M(x)$ est un module de G. Puisque M est non-trivial, il contient au moins deux sommets x et x'. Il existe donc $M(x)$ et $M(x')$ deux modules de $\mathcal{M}(G, v)$ qui sont strictement contenus dans un module de G ne contenant pas v: contradiction.

2. Cette seconde propriété est une conséquence directe de la première et du Lemme 2.15.

3. Soit M un module ne contenant pas v. Par définition de $\mathcal{M}(G, v)$, M ne peut contenir aucun module $M' \in \mathcal{M}(G, v)$. Supposons qu'il existe $M' \in \mathcal{M}(G, v)$ tel que $M \perp M'$. Alors $M \cup M'$ est un module de G ne contenant pas v et $M' \subset M \cup M'$, ce qui contredit la maximalité de $M' \in \mathcal{M}(G, v)$. Donc M est contenu dans un module de $\mathcal{M}(G, v)$.

□

Il s'agit donc de calculer : 1) la partition modulaire $\mathcal{M}(G, v)$ et 2) l'arbre de décomposition du graphe $G_{/\mathcal{M}(G,v)}$. En utilisant l'Algorithme 4 du Chapitre 3, $\mathcal{M}(G, v)$ peut être calculé en temps $O(n + m \log n)$ grâce à l'affinage de partition. Le calcul de $MD(G_{/\mathcal{M}(G,v)})$ est plus délicat. Rappelons que tout au long de l'affinage de partition, il est possible d'extraire les arêtes comprises entre des parties différentes de la partition obtenues. Soit m' le nombre de ces arêtes. Il est alors possible à l'aide d'un tri par paquet de calculer le graphe quotient $G_{/\mathcal{M}(G,v)}$ en temps $O(m')$ (voir [MS00] ou [DGM01] pour les détails de cette opération).

Le reste de cette section est dédié au calcul de $MD(G_{/\mathcal{M}(G,v)})$. Cette étape constitue le verrou de complexité pour l'algorithme qui implique un temps global quadratique. Nous verrons par la suite que c'est précisement ce point qu'améliorent Dahlhaus et al. [DGM01] pour obtenir une complexité sous-quadratique.

4.1.1 Calcul de l'arbre $MD(G_{/\mathcal{M}(G,v)})$

Définition 4.3. *Un graphe G est* emboîté *s'il existe un sommet v contenu dans tous ses modules non-triviaux. Ce sommet v sera appel le sommet* intérieur*.*

Le Lemme suivant est une reformulation de la première propriété du Lemme 4.2 :

Lemme 4.4. *Pour tout graphe $G = (V, E)$ et tout sommet $v \in V$, le graphe quotient $G_{/\mathcal{M}(G,v)}$ est un graphe emboîté dont le sommet intérieur est v.*

Corollaire 4.5. *Les nœuds dégénérés de $MD(G_{/\mathcal{M}(G,v)})$ sont binaires.*

Il s'agit donc de calculer les modules de $MD(G_{/\mathcal{M}(G,v)})$ qui contiennent v (il n'y en a pas d'autre). Pour cela, il est possible d'utiliser le *graphe de forçage* introduit par Ehrenfeucht et al. [EGMS94].

Définition 4.6. [1] *Soit v un sommet quelconque d'un graphe G. Le* graphe de forçage *$\mathcal{F}(G, v)$ est un graphe orienté ayant pour ensemble de sommets $V \setminus \{v\}$. L'arc $\overrightarrow{xy}$ existe si y est un casseur pour $\{x, v\}$.*

Notons que par définition, si $\overrightarrow{xy}$ existe alors tout module M contenant v et x doit contenir le sommet y.

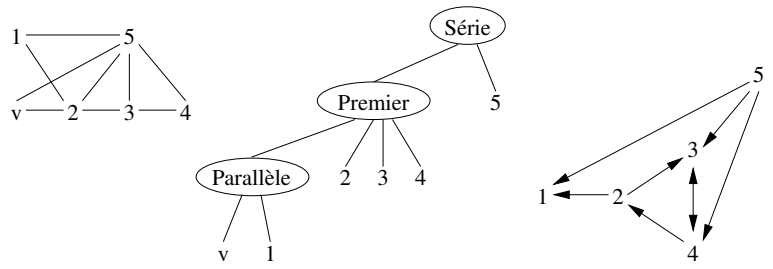

Fig. 4.2. Un graphe G avec son arbre de décomposition modulaire $MD(G)$ et à droite le graphe de forcage $\mathcal{F}(G, v)$. Les compasantes fortement connexes de $\mathcal{F}(G, v)$ sont $\{1\}, \{2, 3, 4\}, \{5\}$. Tout module de G contenant 3 et v doit aussi contenir $\{1, 2, 4\}$, les sommets atteignables dans $\mathcal{F}(G, v)$ depuis 3.

Lemme 4.7. *[EGMS94] Soit X l'ensemble des sommets atteignables depuis x dans le graphe $\mathcal{F}(G, v)$. Alors $\{v\} \cup X$ est le module minimal de G contenant v et x.*

[1] Nous avons adapté la définition proposée par [EGMS94] afin de faciliter le parallèle avec les travaux de [DGM01].

Preuve. Soit M un module contenant v et x. Montrons par induction que $X \subset M$. Soit y un sommet de X. Si $\overrightarrow{xy}$ est un arc de $\mathcal{F}(G, v)$, $y \in M$ par définition de $\mathcal{F}(G, v)$ (car y sépare x et v). Supposons que tout y, atteignable depuis x dans $\mathcal{F}(G, v)$ par un chemin orienté de longueur k, appartienne à X. Soit z atteignable depuis x par un chemin de longueur $k + 1$ et soit y son prédecesseur. Par hypothèse, $y \in M$. Or l'existence de $\overrightarrow{yz}$ implique que z sépare y et v, donc $z \in M$.

Pour conclure, il reste à montrer que M est uniforme par rapport à tout sommet $w \notin X$. Supposons qu'il existe $w \notin X$ et $y \in X$, $z \in X$ tel que $wy \in E$ et $wz \notin E$. Donc soit w sépare y et v soit w sépare z et v. Autrement dit, soit $\overrightarrow{yw}$ soit $\overrightarrow{zw}$ est un arc de $\mathcal{F}(G, v)$ et donc w est atteignable depuis x : contradiction. □

Dans la suite de la section, nous ne considérerons plus que le graphe $G_{/\mathcal{M}(G,v)}$ et noterons le graphe de forçage associé $\mathcal{F}(\mathcal{M}(G, v))$ (dont les sommets sont $\mathcal{M}(G, v) \setminus \{v\}$). En appliquant le Lemme 4.7 à $\mathcal{F}(\mathcal{M}(G, v))$, on obtient comme conséquence directe du Lemme 4.4 et du Corollaire 4.5 :

Corollaire 4.8. *[EGMS94] Soit M_x le module de $\mathcal{M}(G, v)$ contenant le sommet x. Soit $\mathcal{X}$ l'ensemble des modules atteignables depuis M_x dans $\mathcal{F}(\mathcal{M}(G, v))$. Alors $\bigcup_{M \in \mathcal{X}} M$ est le module fort minimal de G contenant v et x.*

Les composantes fortement connexes de $\mathcal{F}(\mathcal{M}(G, v))$ seront appelées les *blocs* de (G, v). Nous pouvons alors définir $\mathcal{B}(G, v)$ le *graphe des blocs* de $\mathcal{F}(\mathcal{M}(G, v))$ (voir par exemple [CLR90]) dont les sommets sont les blocs de (G, v). Un arc de $\mathcal{B}(G, v)$ relie le bloc B au bloc B' si les sommets de B' sont atteignables dans $\mathcal{F}(\mathcal{M}(G, v))$ depuis les sommets de B.

Lemme 4.9. *[EGMS94] La réduction transitive du graphe des blocs $\mathcal{B}(G, v)$ est une chaîne.*

Preuve. Supposons que la réduction transitive de $\mathcal{B}(G, v)$ ne soit pas une chaîne. Il existe donc deux sommets (ou modules de $\mathcal{M}(G, v)$) m_1 et m_2 non atteignables l'un depuis l'autre dans $\mathcal{F}(\mathcal{M}(G, v))$. Soit $\mathcal{X}_1$ et $\mathcal{X}_2$ les ensembles respectivement atteignables depuis m_1 et m_2. D'après le Corollaire 4.8, les ensembles $\mathcal{X}_1$ et $\mathcal{X}_2$ correspondent à des modules de $G_{/\mathcal{M}(G,v)}$ contenant v. Notons que ces deux modules sont incomparables pour l'inclusion: contradiction, le graphe $G_{/\mathcal{M}(G,v)}$ est emboîté (Lemme 4.4). □

Un ensemble de sommets d'un graphe orienté est un *puits* s'il ne possède aucun voisin sortant. D'après le Lemme 4.9, tout ensemble puits de $\mathcal{F}(\mathcal{M}(G, v))$ est donc une union de blocs suffixe de la réduction transitive de $\mathcal{B}(G, v)$. Chaque ensemble puits correspond en fait à un module de $G_{/\mathcal{M}(G,v)}$.

Corollaire 4.10. *[EGMS94] Soit $v \in V$ un sommet d'un graphe $G = (V, E)$. Un ensemble M de sommets contenant v est un module de $G_{/\mathcal{M}(G,v)}$ ssi M est l'union de $\{v\}$ et des modules de $\mathcal{M}(G, v)$ appartenant à un esemble X puits de $\mathcal{B}(G, v)$.*

Le graphe des blocs $\mathcal{B}(G, v)$ permet donc de calculer les modules de $G_{/\mathcal{M}(G,v)}$. Le Lemme 4.4 et le Corollaire 4.5 permettent la construction de $MD(G_{/\mathcal{M}(G,v)})$. Finalement $MD(G)$ est reconstitué récursivement grâce au Lemme 4.2.

4.2 Un algorithme quasi-linéaire

En 2001, Dahlhaus, Gustedt et McConnell [DGM01] améliorent la complexité de l'algorithme de Ehrenfeucht et al. Deux "implémentations" sont proposées, la complexité de la première est $O(n + m.\alpha(n, m))$ alors que celle de la seconde est linéaire. Dans l'algorithme de Ehrenfeucht et al., les deux principaux verrous de complexité étaient le calcul de la partition $\mathcal{M}(G, v)$ et le coût de la construction $MD(G_{/\mathcal{M}(G,v)})$.

Nous présentons la version quasi-linéaire dans cette section. Son principe est le même que celui de l'algorithme de Ehrenfeucht et al., à savoir : 1) calcul d'une partition modulaire $\mathcal{M}(G, v)$; 2) calcul de $MD(G_{/\mathcal{M}(G,v)})$; 3) pour chaque $\mathcal{X} \in \mathcal{M}(G, v)$, calcul récursif de $MD(G[\mathcal{X}])$; et 4) fusion des arbres $MD(G[\mathcal{X}])$ et $MD(G_{/\mathcal{M}(G,v)})$ pour obtenir $MD(G)$.

Pour éviter les deux verrous de complexité mentionés, une stratégie inspirée de Dahlhaus [Dah95b] est utilisée. L'idée est de résoudre le problème récursivement sur les sous-graphes induits $G[N(v)]$ et $G[\overline{N}(v)]$, et de recomposer $\mathcal{M}(G, v)$ et $MD(G_{/\mathcal{M}(G,v)})$ à partir des arbres de décomposition modulaire $MD(G[N(v)])$ et $MD(G[\overline{N}(v)])$. Pour assurer une faible complexité, cette recomposition doit se faire en utilisant seulement les arêtes comprises entre les ensembles $\{v\}$, $N(v)$ et $\overline{N}(v)$. Ces arêtes seront appelées les *arêtes actives*. Clairement un coût linéaire en le nombre d'arêtes actives permet d'obtenir une complexité globale linéaire.

En faisant appel à une analyse et des structures de données très fines (*union-find* [Tar83, CLR90] et plus petit ancêtre commun [HT84]), [DGM01] obtiennent un algorithme linéaire. Nous ne présenterons ici que la version simple dont la complexité est $O(n+m.\alpha(n, m))$. L'unique différence entre ces deux implémentations réside dans un prétraitement du graphe et une analyse très fine de la complexité. Nous ne détaillerons pas ces différences.

4.2.1 Calcul de $\mathcal{M}(G, v)$

Les deux observations suivantes sont des conséquences directes de la définition de module. Elles nous permettent de déduire un algorithme qui, étant donné $MD(G[N(v)])$ (resp. $MD(G[\overline{N}(v)])$), nous permet d'obtenir non seulement les modules $\mathcal{X} \in \mathcal{M}(G, v)$ inclus dans $N(v)$ (resp. $\overline{N}(v)$), mais aussi les arbres $MD(G[\mathcal{X}])$.

Lemme 4.11. *Si $\mathcal{X}$ est un module de $\mathcal{M}(G, v)$, alors $\mathcal{X}$ est soit un module de $G[N(v)]$ ou un module de $G[\overline{N}(v)]$.*

Algorithme 6 : Dahlhaus [Dah95a]

Données : Un graphe $G = (V, E)$
Résultat : $MD(G)$
début
  si *G ne possède qu'un seul sommet v* **alors**
    $MD(G)$ est un arbre avec un seul sommet $\{v\}$;
  sinon
    Soit v un sommet quelconque de G;
    Calculer récursivement $MD(G[N(v)])$ et $MD(G[\overline{N}(v)])$;
    Extraire $\mathcal{M}(G, v)$ de $MD(G[N(v)])$ et $MD(G[\overline{N}(v)])$;
    pour chaque $X \in \mathcal{M}(G, v)$ **faire**
      Extraire $MD(G[X])$ de $MD(G[N(v)])$ ou $MD(G[\overline{N}(v)])$;
    Calculer $MD(G_{/\mathcal{M}(G,v)})$;
    Calculer $MD(G)$ à partir de $MD(G_{/\mathcal{M}(G,v)})$ et l'ensemble $\{MD(G[X]) \mid X \in \mathcal{M}(G, v)\}$ (voir Algorithme 5);
fin

Lemme 4.12. *Un ensemble $M \subseteq S \subset V$ est un module de $G = (V, E)$ ssi M est un module de $G[S]$ et M est uniforme par rapport à tout sommet $x \in V \setminus S$.*

L'algorithme d'extraction des modules maximaux $\mathcal{X} \in \mathcal{M}(G, v)$ inclus dans $N(v)$ (resp. $\overline{N}(v)$) est un marquage de l'arbre $MD(G[N(v)])$ (resp. $MD(G[\overline{N}(v)])$) des feuilles vers la racine. Pour simplifier sa description, nous le paramétrons par un ensemble S qui sera soit $N(v)$ soit $\overline{N}(v)$. Chaque nœud reçoit le type *supprimé* ou *copié* selon que ses fils aient ou non le même voisinage dans $V \setminus S$.

Lemme 4.13. *[DGM01] L'Algorithme 7 appliqué successivement à $N(v)$ et à $\overline{N}(v)$ (ie. Extraction$(G, N(v))$ et Extraction$(G, \overline{N}(v))$) permet de calculer $\mathcal{M}(G, v)$ et pour chaque module $\mathcal{X} \in \mathcal{M}(G, v)$, l'arbre de décomposition modulaire $MD(G[\mathcal{X}])$.*

La preuve du lemme précedent, et donc la correction de l'Algorithme 7, découle des Lemmes 4.11 et 4.12.

4.2.2 Calcul de $MD(G_{/\mathcal{M}(G,v)})$

De la même manière que dans l'algorithme de Ehrenfeucht et al., un graphe forçage est utilisé pour construire l'arbre de décomposition modulaire d'un graphe emboîté. Celui défini par Dahlhaus et al. est toutefois différent. Ses sommets ne sont plus les modules de $\mathcal{M}(G, v) \setminus \{v\}$, mais les classes d'équivalence d'une relation d'équivalence $\mathcal{K}(G, v)$ définie sur les sommets de $V \setminus \{v\}$. Un graphe de forçage, $\mathcal{F}(\mathcal{K}(G, v))$, sur les classes d'équivalence sera défini de manière analogue au graphe $\mathcal{F}(\mathcal{M}(G, v))$ (voir Définition 4.6).

Algorithme 7 : Extraction(G, S)

Données : Le graphe G et $MD(G[S])$
Résultat : L'ensemble des arbres $MD(G[\mathcal{X}])$ tels que $\mathcal{X} \in \mathcal{M}(G, v)$ et $\mathcal{X} \subseteq S$
début
  pour chaque *feuille x de $MD(G[S])$* **faire**
    Soit $L(x) = N(x) \cap (V \setminus S)$;
  tant que *il existe un nœud N non marqué dont les fils sont marqués* **faire**
    si *N possède un fils marqué* SUPPRIMÉ **alors**
      N est marqué SUPPRIMÉ;
    sinon
      si *tous les fils de N ont la même liste de voisins* **alors**
        N est marqué COPIÉ;
        $L(N) = L(N')$ avec N' l'un des fils de N;
      sinon N est marqué SUPPRIMÉ;
  pour chaque *nœud N* COPIÉ *dont le père P est* SUPPRIMÉ **faire**
    si *P est dégénéré* **alors**
      Créer un nouveau nœud X, dégénéré, dont les fils sont les frères COPIÉ de N ayant $L(N)$ pour liste de voisins;
      $X \in \mathcal{M}(G, v)$ et $MD(G[X])$ est le sous-arbre enraciné en X;
    sinon
      $N \in \mathcal{M}(G, v)$ et $MD(G[N])$ est le sous-arbre enraciné en N;
fin

Définition 4.14. *[DGM01] Soient x et y deux sommets distincts de $\overline{N}(v)$ (resp. $N(v)$), alors $x\mathcal{K}y$ ssi :*

- *x et y appartiennent à la même composante connexe de $G[\overline{N}(v)]$ (resp. $\overline{G}[N(v)]$), ou*
- *la composante connexe de $G[\overline{N}(v)]$ (resp. $G[N(v)]$) contenant x et celle contenant y sont incluses dans un module M de G tel que $M \subset \overline{N}(v)$ (resp. $M \subset N(v)$).*

Puisqu'un module ne peut chevaucher une composante connexe, les classes d'équivalence sont des union de modules de composantes connexes de $G[\overline{N}(v)]$ ou $\overline{G}[N(v)]$. Notons que le premier intérêt de cette définition est que les classes d'équivalence de $\mathcal{K}(G, v)$ peuvent être calculées à partir de celles issues de $\mathcal{K}(G[N(v)], u)$, avec $u \in N(v)$, et de $\mathcal{K}(G[\overline{N}(v)], w)$, avec $w \in \overline{N}(v)$, en n'utilisant que les arêtes actives[2]. Il suffit pour cela de calculer l'union de composantes connexes (ou composantes co-connexes). Ce n'était pas le cas des modules de $\mathcal{M}(G, v)$.

Commençons par montrer les liens entre les classes d'équivalence de $\mathcal{K}(G, v)$ et modules de $\mathcal{M}(G, v)$.

[2] Rappelons que les arêtes actives sont les arêtes comprises entre les ensembles $\{v\}$, $N(v)$ et $\overline{N}(v)$.

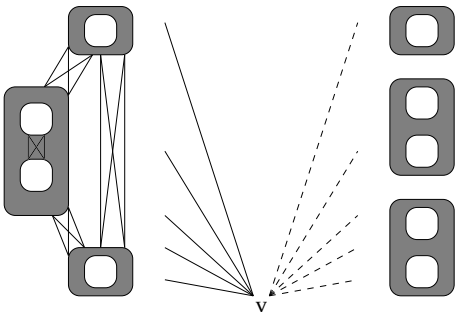

Fig. 4.3. Les classes d'équivalence de $\mathcal{K}(G, v)$ apparaîssent en grisé. Les parties blanches correspondent respectivement aux composantes connexes de $G[\overline{N}(v)]$ et $\overline{G}[N(v)]$.

Lemme 4.15. *Toute classe d'équivalence K de $\mathcal{K}(G, v)$ est l'union de modules de $\mathcal{M}(G, v)$. De plus $\forall x, y \in K$ tels que $x \neq y$, nous avons $m(x, v) = m(y, v)$.*

Preuve. Considérons une classe d'équivalence K contenue dans $\overline{N}(v)$. Une preuve similaire s'applique lorsque K est incluse de $N(x)$.

Si K est non-connexe, alors c'est, par définition, un module M de $\mathcal{M}(G, v)$. Si M n'est pas un module fort de G, alors M est l'union de modules forts fils d'un nœud parallèle ancêtre de v dans $MD(G)$ (voir Théorème 2.4). Nous avons donc $m(x, v) = m(y, v)$.

Supposons K connexe. S'il existe un module M tel que $K \perp M$, alors K contient deux sommets adjacents $u \notin M$ et $v \in M$. Or par définition v est adjacent à l'ensemble des sommets de M qui est donc un sous-ensemble de K : contradiction. Aucun module ne chevauche donc K. K est donc une union de modules ne contenant pas v. Par maximalité d'une classe d'équivalence, il n'existe pas de modules contenant K. K est bien une union de modules appartenant à $\mathcal{M}(G, v)$. Pour terminer la preuve, il faut montrer que pour tous sommets distincts $x, y \in K$, $m(v, x) = m(v, y)$, où $m(v, x)$ est le plus petit module fort contenant v et x. Supposons par contradiction que $m(v, x) \subset m(v, y)$. Puisque K est connexe, il existe un chemin sans corde dans $G[K]$ entre x et y. Soit z le premier sommet sur ce chemin n'appartenant pas à $m(v, x)$ (z peut être égal à y). Mais z possède un voisin dans $m(v, x)$ (son prédécesseur sur le chemin) et un non-voisin (le sommet v) : contradiction $m(v, x)$ est un module. $\square$

Autrement dit, les modules contenus dans une classe d'équivalence K sont tous fils d'un même ancêtre de v dans $MD(G)$ (voir Figure 4.4). Le Lemme 4.15 indique donc que les classes d'équivalence de $\mathcal{K}(G, v)$ définissent une partition plus large que $\mathcal{M}(G, v)$, mais qu'elle doit aussi permettre de retrouver l'arbre de décomposition modulaire $MD(G_{/\mathcal{M}(G,v)})$. Le Lemme 4.16 est similaire à la propriété 2 du Lemme 4.2.

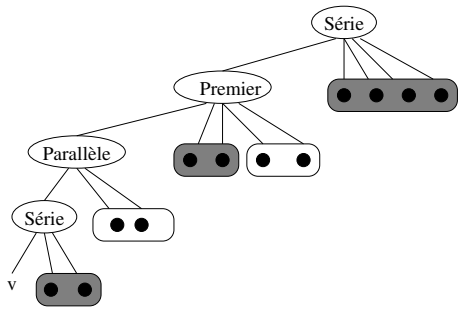

Fig. 4.4. Les classes d'équivalence de $\mathcal{K}(G, v)$ dans l'arbre de décomposition modulaire $MD(G)$. En grisé, celles issues de $N(v)$.

Lemme 4.16. *Tout module fort de G contenant v est l'union de $\{v\}$ et d'un ensemble de classes d'équivalence de $\mathcal{K}(G, v)$.*

Preuve. D'après le Lemme 4.2, tout module fort de G contenant v est l'union de modules de $\mathcal{M}(G, v)$. Le résultat découle du Lemme 4.15 puisque toute classe d'équivalence de $\mathcal{K}(G, v)$ est une union de modules de $\mathcal{M}(G, v)$ fils du même ancêtre de v. □

Ainsi, chaque nœud de $MD(G)$ ancêtre de v (ou module fort contenant v) est associé à une ou plusieurs classes d'équivalence : une seule si c'est un nœud dégénéré, éventuellement plusieurs sinon. Nous pouvons maintenant définir le graphe de forçage sur les classes d'équivalence de $\mathcal{K}(G, v)$.

Définition 4.17. *[DGM01] Soit v un sommet quelconque d'un graphe G. Le* graphe de forçage *$\mathcal{F}(\mathcal{K}(G, v))$ est un graphe orienté ayant pour ensemble de sommets les classes d'équivalence de $\mathcal{K}(G, v)$. Soit K et K' deux classes d'équivalence de $\mathcal{K}(G, v)$, alors l'arc $\overrightarrow{KK'}$ existe ssi il existe $x \in K$ et $y \in K'$ tel que y est un casseur pour x et v.*

Intéressons nous aux composantes fortement connexes de $\mathcal{F}(\mathcal{K}(G, v))$, que nous appellerons *$\mathcal{K}$-blocs*. Le graphe des $\mathcal{K}$-blocs $\mathcal{B}'(G, v)$ est définit de la même manière que $\mathcal{B}(G, v)$. Les Lemmes 4.15 et 4.16 permettent de déduire des résultats similaires au Corollaire 4.8 et au Lemme 4.9. Les preuves étant de même nature, nous les omettons ici.

Lemme 4.18. *[DGM01] Soit x un sommet dont la classe d'équivalence est K_x. Soit $\mathcal{X}$ l'ensemble des classes d'équivalence atteignables depuis K_x dans $\mathcal{B}'(G, v)$, alors $\{v\} \cup_{X \in \mathcal{X}} X$ est le module fort minimal de G contenant v et x.*

Lemme 4.19. *[DGM01] Le graphe des $\mathcal{K}$-blocs $\mathcal{B}'(G, v)$ est une chaîne.*

Corollaire 4.20. *[DGM01] Soit $v \in V$ un sommet d'un graphe $G = (V, E)$. Un ensemble M de sommets contenant v est un module de $G_{/\mathcal{M}(G,v)}$ ssi $M \setminus \{v\} = \bigcup_{K \in \mathcal{X}} K$ où $\mathcal{X}$ est un ensemble puits*[3].

Il est donc possible d'obtenir les modules forts de G contenant v à l'aide d'un tri topologique de $\mathcal{B}'(G, v)$. Pour résumer, l'algorithme ci-dessous décrit les grandes lignes du calcul de l'arbre de décomposition modulaire de $G_{\mathcal{M}(G,v)}$:

Algorithme 8 : Calcul de $MD(G_{/\mathcal{M}(G,v)})$ [DGM01]

Données : Un graphe G, un sommet v et $MD(G[N(v)])$, $MD(G[\overline{N}(v)])$
Résultat : $MD(G_{/\mathcal{M}(G,v)})$
début
  Calcul des graphes $\mathcal{F}(\mathcal{K}(G, v))$ et $\mathcal{B}'(G, v)$;
  Construction de $MD(G_{/\mathcal{M}(G,v)})$ à partir d'un tri topologique de $\mathcal{B}'(G, v)$;
fin

4.2.3 Mise en œuvre et analyse de complexité

Nous ne décrirons pas dans les détails l'analyse de l'algorithme de Dahlhaus et al. [DGM01]. Le lecteur intéressé est invité à se référer à l'article d'origine. Nous allons en revanche indiquer quels sont les outils nécessaires à sa mise en œuvre, en espérant ainsi donner une intuition qu'il est possible d'obtenir une complexité en $O(n + m.\alpha(n, m))$.

Précalcul de l'arbre de récursion

L'idée principale est lors de chaque appel récursif, de n'utiliser que les arêtes actives. Il faut pour cela précalculer un arbre de récursion à partir du quel une partition des arêtes sera calculée. À chaque sommet v on associe une liste d'arête $Actives(v)$.

La méthode pour le calcul de l'arbre est simple, elle peut s'assimiler à un algorithme de partitionnement. Sa complexité est clairement $O(n + m)$.

Il s'agit maintenant de calculer les arêtes actives de chaque sommet. L'arête xy appartient à $Actives(v)$ ssi $lca(x, y)$ est le père de la feuille étiquetée par $\{v\}$ dans l'arbre de récursion. En utilisant des requêtes de plus petit ancêtre commun, il est alors possible de calculer pour chaque sommet v, l'ensemble $Actives(v)$. Cela peut se faire avec un coût $O(n+m.\alpha(n, m))$ en utilisant une structure de donnée décrite dans [CLR90].

[3] Les ensembles $K \in \mathcal{X}$ sont les classes d'équivalence de $\mathcal{K}(G, v)$

Algorithme 9 : Précalcul de l'arbre de récursion [DGM01]

Données : Un graphe $G = (V, E)$
Résultat : Un arbre de récursion T et pour chaque sommet, les arêtes actives associées
début
  si *G ne possède qu'un sommet v* **alors**
    T n'a qu'un nœud étiqueté par $\{v\}$;
  sinon
    Soit v un sommet de V;
    La racine de T étiquetée par V possède 3 fils $T(G[\{v\}])$, $T(G[N(v)])$ et $T(G[\overline{N}(v)])$;
fin

Extraction des arbres $MD(G[X])$ tels que $X \in \mathcal{M}(G, v)$

En utilisant une structure de donnée adéquate pour représenter un arbre de décomposition modulaire (voir [DGM01] plus les détails), l'Algorithme 7 appliqué respectivement à $N(v)$ et $\overline{N}(v)$ (ie. respectivement Extraction$(G, N(v))$ et Extraction$(G, \overline{N}(v))$) peut être implémenté en temps $O(|Actives(v)|)$.

En plus de la représentation de l'arbre de décomposition modulaire, une structure d'*union-find* est maintenue pour représenter les composantes connexes du graphe. Elle permet notamment d'identifier, lors de la phase d'extraction, les classes de la relation d'équivalence $K(G, v)$ sans surcoût de complexité.

Calcul de $MD(G_{/\mathcal{M}(G,v)})$

Nous avons vu que $MD(G_{/\mathcal{M}(G,v)})$ pouvait s'obtenir à partir du graphe de forçage $\mathcal{F}(\mathcal{K}(G, v))$ avec un tri topologique du graphe des blocs $\mathcal{B}'(G, v)$. Remarquons qu'il existe au plus une classe d'équivalence de $\mathcal{K}(G, v)$ dont aucun sommet n'est incident à une arête de $Actives(v)$. On en déduit que le graphe $\mathcal{F}(\mathcal{K}(G, v))$ est de taille $O(|Actives(v)|)$. En utilisant un résultat de Dahlaus et al. [DGM02], il possible de calculer un tri topologique des composantes fortement connexes de $\mathcal{F}(\mathcal{K}(G, v))$ en temps linéaire, (ie. $O(|Actives(v)|)$).

Calcul de $MD(G)$

L'unique travail supplémentaire à faire par rapport à l'Algorithme 5, est de retrouver les composantes connexes de G à partir de celles de $G[N(v)]$ et de $G[\overline{N}(v)]$. C'est le rôle de la structure d'*union-find*. Le coût global de l'ensemble des opérations d'union sera $O(n + m.\alpha(n, m))$.

4.3 Une implémentation simple et sous-quadratique

On peut lire en conclusion de [MS00] le commentaire suivant sur les deux implémentations de l'algorithme de Ehrenfeucht et al. proposées par Dahlhaus et al. [DGM01] :

"Up until now, turning [Dalhlaus et al.'s] idea into an $O(n + m.\alpha(n, m))$ algorithm requires conceptually difficult tricks, [...]. It also requires careful charging arguments to prove the time bound. The linear variant uses sophisticated union-find data structures."

Cet section présente l'implémentation de l'algorithme de Ehrenfeucht et al. proposée par McConnell et Spinrad [MS00]. Sa complexité est en $O(n + m \log n)$. Elle repose fortement sur l'algorithme 4 (dont l'analyse de complexité a été présentée Chapitre 3) d'une part pour le calcul de la partition modulaire $\mathcal{M}(G, v)$ mais aussi pour obtenir $MD(G_{/\mathcal{M}(G,v)})$. Pour ce dernier aspect, l'ordre entre les parties de la partition à affiner est important. On parle dans la littérature d'*affinage de partition ordonnée* [HPV98, MS00]. En effet, nous allons montrer que $MD(G_{/\mathcal{M}(G,v)})$ peut être reconstruit à partir de l'algorithme 4 s'il utilise la règle d'affinage qui lorsqu'une partie $\mathcal{X}$ est coupée par $N(x)$, alors $N(x) \cap \mathcal{X}$ est placé après $\overline{N}(x) \cap \mathcal{X}$ ssi le sommet x apparaît dans $\mathcal{P}$ avant les sommets de la partie $\mathcal{X}$. Il est donc notamment possible d'éviter la construction de $\mathcal{F}(G, v)$ et du graphe des blocs associé.

Algorithme 10 : Partition modulaire ordonnée : $PMO(G, v)$

Données : La partition $\mathcal{P} = [\{v\}, V \setminus \{v\}]$ de l'ensemble des sommets d'un graphe $G = (V, E)$
Résultat : La partition modulaire ordonnée $\mathcal{M}(G, v)$
début
  Utiliser l'Algorithme 4 en remplaçant la ligne 2 par;
  Soient $\mathcal{Y}_1 = \mathcal{Y} \cap N(x)$ et $\mathcal{Y}_2 = \mathcal{Y} \setminus N(x)$;
  si $\mathcal{X} <_{\mathcal{Q}} \mathcal{Y}$ **alors** Remplacer dans $\mathcal{Q}$, $\mathcal{Y}$ par $[\mathcal{Y}_2, \mathcal{Y}_1]$;
  sinon Remplacer dans $\mathcal{Q}$, $\mathcal{Y}$ par $[\mathcal{Y}_1, \mathcal{Y}_2]$;
fin

Notation 4.21 *Désormais, $\mathcal{M}(G, v)$ désignera la partition modulaire ordonnée obtenue par $PMO(G, v)$ (Algorithme 10). Il est alors possible d'associer à $\mathcal{M}(G, v)$ un unique ordre total $\sigma(G, v)$ sur les sommets du graphe quotient $G_{/\mathcal{M}(G,v)}$ de la manière suivante : $x <_{\sigma(G,v)} y$ ssi $\mathcal{X} <_{\mathcal{M}(G,v)} \mathcal{Y}$ avec $\mathcal{X}$ et $\mathcal{Y}$ étant respectivement les modules de $\mathcal{M}(G, v)$ contenant x et y.*

Lemme 4.22. *Si $G_{/\mathcal{M}(G,v)}$ est un graphe connexe, alors le dernier sommet z de $\sigma(G, v)$ est un sommet issu d'un module fils de la racine de $MD(G_{/\mathcal{M}(G,v)})$.*

Preuve. Nous devons considérer deux cas selon que la racine de $MD(G_{/\mathcal{M}(G,v)})$ est un nœud premier ou série.

- La racine de $MD(G_{/\mathcal{M}(G,v)})$ est un nœud premier : Supposons que z ne soit pas fils de la racine. Et considérons la première étape de l'algorithme où un sommet x fils de la racine et voisin de v a été séparé de z. Soit y le sommet ayant séparé x de z. Puisque $y <_{\sigma(G,v)} x <_{\sigma(G,v)} z$, nous avons $yx \notin E$ et $yz \in E$. Notons que y est nécessairement un fils de la racine : sinon il ne pourrait être un casseur pour $\{z, x\}$ car $G_{/\mathcal{M}(G,v)}$ est un graphe emboîté. Il existe donc à cette étape un sommet, fils de la racine, qui a déjà été séparé de z : contradiction, z est bien fils de la racine.
- La racine de $MD(G_{/\mathcal{M}(G,v)})$ est un nœud série : Puisque $G_{/\mathcal{M}(G,v)}$ est un graphe emboîté, il possède un sommet x universel. Supposons que $x \neq z$. Soit $\mathcal{X}$ la dernière partie ayant contenu x et z lors du processus d'affinage et soit y le sommet pivot ayant permis de séparer x et z. Puisque $\mathcal{X}$ contient le dernier sommet z de $\sigma(G, v)$, nous avons $y <_{\sigma(G,v)} x <_{\sigma(G,v)} z$. La règle d'affinage ordonné implique donc que y et x ne sont pas adjacents : contradiction, x est universel. Nous avons donc montré que $x = z$.

□

Corollaire 4.23. *Si G est un graphe emboîté connexe de sommet intérieur v et si z est le dernier sommet de $\sigma(G, v)$, alors $\mathcal{M}(G, z)$ contient tous les modules forts maximaux de $G_{/\mathcal{M}(G,v)}$.*

Nous pouvons donc déduire du Corollaire 4.23 un algorithme récursif simple pour le calcul de $MD(G_{/\mathcal{M}(G,v)})$ (voir Algorithme 11). Le seul cas restant à prendre en compte est celui où la racine de $MD(G_{/\mathcal{M}(G,v)})$ est un nœud parallèle. Ce cas est néanmoins trivial puisque $G_{/\mathcal{M}(G,v)}$ possède alors un unique sommet isolé z. Le reste des sommets de $G_{/\mathcal{M}(G,v)}$ constitue alors l'unique module fort maximal différent de $\{z\}$.

Algorithme 11 : [MS00] Calcul de $MD(G_{\mathcal{M}(G,v)})$

Données : $G_{/\mathcal{M}(G,v)}$ et l'ordre $\sigma(G, v)$ obtenu par $PMO(G, v)$
Résultat : $MD(G_{/\mathcal{M}(G,v)})$
début
  si $G_{/\mathcal{M}(G,v)}$ *ne possède un sommet isolé* **alors** Soit z ce sommet;
  sinon Soit z le denier sommet de $\sigma(G, v)$;
  si $G_{/\mathcal{M}(G,v)}$ *ne possède qu'un seul sommet* **alors** retourner $\{z\}$;
  sinon
    Soit $[\mathcal{X}_1 \dots \mathcal{X}_k] = PMO(G_{/\mathcal{M}(G,v)}, z)$;
    Soit r la racine de $MD(G_{\mathcal{M}(G,v)})$;
    pour chaque $\mathcal{X}_i$ **faire** soit $MD(G_{/\mathcal{M}(G,v)}[\mathcal{X}_i])$ le i-ème fils de r;
    retourner $MD(G_{/\mathcal{M}(G,v)})$;
fin

Théorème 4.24. *Etant donné un graphe $G = (V, E)$, l'Algorithme 5 permet de calculer $MD(G)$ en temps $O(n + m \log n)$ si les Algorithmes 10 et 11 sont utilisés.*

Preuve. La correction de l'algorithme a déjà été discutée plus tôt. Elle découle du Théorème 3.5, du Lemme 4.2 et du Corollaire 4.23. Nous allons donc discuter ici de l'analyse de complexité.

Rappelons dans un permier temps que la complexité de l'Algorithme 10 est $O(n+m \log n)$ (voir Théorème 3.5). Mais l'Algorithme 5 a besoin d'appeler récursivement l'Algorithme 10 dans chacune des parties de $\mathcal{M}(G, v)$. Pour ne pas acroître la complexité globale, il suffit pour chaque module $\mathcal{X}$ de $\mathcal{M}(G, v)$ de relancer l'algorithme avec la partition $[\{x\}, \mathcal{X} \setminus \{x\}]$ où x est le dernier sommet de $\mathcal{X}$ dont le voisinage a été utilisé pour affiner la partition. Sous cette condition, on peut garantir que le voisinage de chaque sommet sera utiliser au plus $\log n$ fois dans un appel à l'Algorithme 10.

Par ailleurs, nous avons vu [DGM01] qu'avec une structure de donnée adaptée, il est possible d'extraire en temps $O(n + m')$ le graphe quotient d'une partition modulaire $\mathcal{P}$, avec m' le nombre d'arêtes xy telles que x et y n'appartiennent pas aux mêmes modules de $\mathcal{P}$. Notons que les graphes quotients, pour lesquels l'Algorithme 11 est utilisé, ont tous des ensembles d'arêtes disjoints. Ainsi le coût total d'extraction de l'ensemble des graphes quotients est $O(n + m)$ d'une part. Et d'autre part, le coût cumulé des appels à l'Algorithme 11 est $O(n + m \log n)$.

On en déduit que les Algorithmes 10 et 11 permettent une implémentation de l'Algorithme 5 en temps $O(n + m \log n)$. □

4.4 Notes bibliographiques

L'algorithme de Dahlhaus et al. [DGM01] peut être implémenté en temps linéaire grâce à une étape plus attentive de précalcul de l'arbre de récursion. Dans la mesure où certaines propriétés de connexité sont garanties pour l'arbre de récursion, alors analyse très fine de la complexité permet de montrer la borne linéaire. Nous invitons le lecteur à lire l'article d'origine [DGM01] pour plus de détails. Il faut cependant rappeler les différents commentaires (voir [MS00] et [Spi03]) indiquant que le problème de l'existence d'un algorithme simple et linéaire de décomposition modulaire reste à leur connaissance toujours ouvert. Nous proposerons dans les chapitres suivants deux pistes possibles pour résoudre ce problème.

Tarte à l'oignon et aux raisins

Ingrédients : 4 beaux oignons, 150g de raisin de corinthes, gingembre frais, 1 clou de girofle, huile d'olive, vinaigre de Xéres.

Emincer les oignons et les faire revenir à feu doux dans l'huile d'olive avec le clou de girofle. Lorsqu'ils sont bien dorés, déglacer les oignons avec une cuiller à café de vinaigre de Xéres et ajouter 2 cuillers à soupe de gingembre emincé. Couvrir le plat à tarte d'une feuille d'alu. Tapisser le fond du plat avec les oignons et les raisins. Recouvrir la préparation avec la pâte. Mettre au four chaud pendant 25 minutes (250). Démouler la tarte et sopoudrer la coriande coupée finement.

5

Autour des permutations factorisantes

En 1996, Capelle [Cap96] étudie l'algorithmique associée à différentes familles partitives. En particulier, il constate l'existence d'algorithmes calculant l'arbre $T_{\mathcal{S}}$ d'inclusion des parties fortes d'une famille partitive $\mathcal{S}$ à partir d'une permutation factorisante des éléments de $\mathcal{S}$. Les exemples cités sont notamment l'arbre de décomposition modulaire pour les graphes triangulés [HM91] et l'arbre des blocs des graphes d'héritages [HHS95]. Capelle [Cap96] généralise ces travaux et propose le premier algorithme linéaire qui étant donnée une permutation factorisante des modules d'un graphe G, retourne en temps $O(n+m)$ l'arbre de décomposition modulaire $MD(G)$ [CHdM02].

En 1998 [HPV98, HPV99], nous avons proposé un algorithme d'affinage de partition en $O(n + m \log n)$ calculant une permutation factorisante d'un graphe G. Restreint aux cographes, il est possible d'améliorer cet algorithme afin d'obtenir une complexité linéaire [HP05].

L'algorithme de décomposition modulaire peut donc se décomposer en deux étapes : 1) calcul d'une permutation factorisation; 2) calcul de l'arbre de décomposition à partir de la permutation factorisante. Le principal intérêt de cette méthode est d'obtenir un algorithme simple (par exemple basé sur l'affinage de partition) évitant des structures de données telles que *union-find* et *plus petit ancêtre commun*. Même si certaines similitudes peuvent être faites avec l'algorithmes de Ehrenfeucht et al., je suis convaincu que cette méthode est une piste sérieuse pour répondre à la question de Spinrad [Spi03] sur l'existence d'un algorithme linéaire simple de décomposition modulaire basée sur l'affinage de partition.

Par ailleurs, nous le verrons dans le dernier chapitre, dans certaines applications (en génomique comparative par exemple), la donnée n'est pas le graphe, ou la famille partitive, mais bien une permutation factorisante. Il est donc intéressant d'étudier cet objet combinatoire.

5.1 Calcul d'une permutation factorisante

Nous allons dans un premier temps revisiter l'algorithme d'affinage de partition (Algorithme 3) en utilisant cette fois des partitions ordonnées [HPV98]. Nous montrerons ainsi comment il peut être facilement adapté pour obtenir une permutation factorisante en temps $O(n + m \log n)$. Ce premier algorithme est à comparer à l'implémentation de McConnell et Spinrad [MS00] de l'algorithme Ehrenfeucht et al. Ici l'arbre n'est pas construit au fil de l'algorithme, par contre l'ordre entre les différentes parties lors de l'affinage est important.

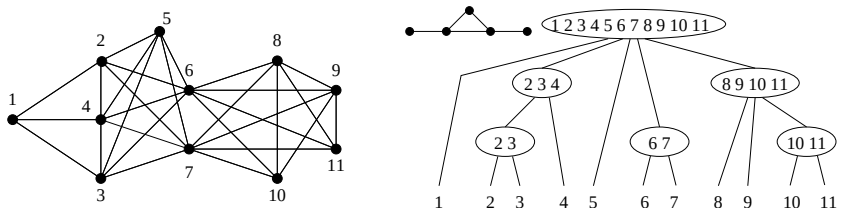

Fig. 5.1. Un graphe $G = (V, E)$ dont une permutation factorisante est $\sigma =$ **1 2 3 4 5 6 7 8 9 10 11** (voir Definition 2.5).

5.1.1 Notations

Comme indiqué ci-dessus, l'algorithme utilise la technique d'affinage de partition, mais pour des partitions ordonnées. Commençons par fixer quelques notations et par généraliser la notion d'intervalle d'une permutation pour l'appliquer aux partitions ordonnées.

Par hypothèse toute partition manipulée dans ce chapitre sera une partition ordonnée. Remarquons qu'une partition ordonnée $\mathcal{P} = [\mathcal{X}_1, \ldots, \mathcal{X}_k]$ d'un ensemble $\mathcal{E}$ peut être interprétrée comme un ordre partiel sur $\mathcal{E}$. Chaque partie $\mathcal{X} \in \mathcal{P}$ est une anti-chaîne et pour toute paire x_i, x_j d'éléments appartenant à des parties distinctes $\mathcal{X}_i$ et $\mathcal{X}_j$, $x_i <_{\mathcal{P}} x_j$ ssi $i < j$. De même, nous noterons abusivement $x <_{\mathcal{P}} M$ avec $x \in \mathcal{E}$ et $M \subset \mathcal{E}$ si $x <_{\mathcal{P}} y$ pour tout $y \in M$.

Définition 5.1. *Soit $\mathcal{P}$ une partition ordonnée d'un ensemble $\mathcal{E}$. Un sous-ensemble $S \subseteq \mathcal{E}$ est un* intervalle *de $\mathcal{P}$ ssi il existe deux parties $\mathcal{L} \in \mathcal{P}$ et $\mathcal{R} \in \mathcal{P}$ (éventuellement identiques) telles que $\mathcal{L} \cap \mathcal{X} \neq \emptyset$ et $\mathcal{R} \cap \mathcal{X} \neq \emptyset$ et telles que pour toute partie $\mathcal{X}$:*

- *si $\mathcal{L} <_{\mathcal{P}} \mathcal{X} <_{\mathcal{P}} \mathcal{R}$, alors $\mathcal{X} \subset S$;*
- *si $\mathcal{X} <_{\mathcal{P}} \mathcal{L}$ ou $\mathcal{R} <_{\mathcal{P}} \mathcal{X}$, alors $\mathcal{X} \cap S = \emptyset$*

Notons que l'opération d'affinage d'une partition ordonnée $\mathcal{P}$ de $\mathcal{E}$ peut être interprétée comme une extension de l'ordre partiel associé : des comparabilités sont ajoutées entre les éléments des parties de $\mathcal{E}$ séparées.

Définition 5.2. *Soient $\mathcal{P}$ et $\mathcal{Q}$ deux partitions ordonnées d'un ensemble $\mathcal{E}$. $\mathcal{Q}$ est* compatible *avec $\mathcal{P}$ ssi pour tout $x \neq y \in \mathcal{E}$ tels que $x <_{\mathcal{P}} y$, alors $x <_{\mathcal{Q}} y$. En particulier, une permutation σ de $\mathcal{E}$ est* compatible *avec $\mathcal{P}$ ssi pour tout $x \neq y \in \mathcal{E}$ tels que $x <_{\mathcal{P}} y$, nous avons $\sigma(x) < \sigma(y)$.*

Autrement dit, si σ est compatible avec $\mathcal{P}$, alors σ est une extension linéaire de $\mathcal{P}$. C'est à dire que σ peut être obtenue par affinages successifs de $\mathcal{P}$.

5.1.2 Principe de base

L'idée principale de ce premier algorithme est : 1) de calculer une partition modulaire $\mathcal{M}(G, v)$ en garantissant que les modules contenant le sommet v soient tous intervalles de $\mathcal{M}(G, v)$; et 2) de calculer récursivement une permutation factorisante pour chacun des sous-graphes induits par un module $M \in \mathcal{M}(G, v)$.

Algorithme 12 : *Permutation-Factorisante(G, v)*

Données : Un graphe $G = (V, E)$ et un sommet $v \in V$
Résultat : Une permutation factorisante de G
début
  Soit $\mathcal{P} = [\overline{N}(v), \{v\}, N(v)]$ une partition ordonnée;
  Appliquer l'Algorithme 4 avec la règle d'affinage suivante;
  Soient x le pivot courant et $\mathcal{Y}$ une partie telle que $N(x) \perp \mathcal{Y}$;
  si $x \leqslant_{\mathcal{P}} v \leqslant_{\mathcal{P}} \mathcal{Y}$ *ou* $\mathcal{Y} \leqslant_{\mathcal{P}} v \leqslant_{\mathcal{P}} x$ **alors**
    Substituer $\mathcal{Y}$ par $[\mathcal{Y} \cap \overline{N}(x), \mathcal{Y} \cap N(x)]$;
  sinon
    Substituer $\mathcal{Y}$ par $[\mathcal{Y} \cap N(x), \mathcal{Y} \cap \overline{N}(x)]$;
  pour chaque *partie* $\mathcal{X} \in \mathcal{M}(G, v)$, *telle que* $|\mathcal{X}| > 1$ **faire**
    Soit x le dernier sommet de $\mathcal{X}$ utilisé comme pivot;
    $\mathcal{P}_{\mathcal{X}} \leftarrow$ Permutation-Factorisante($G[\mathcal{X}]$, x);
    Substituer $\mathcal{X}$ par $\mathcal{P}_{\mathcal{X}}$;
fin

Théorème 5.3. *L'Algorithme 12 calcule en temps $O(n + m \log n)$ une permutation factorisante d'un graphe $G = (V, E)$.*

Preuve. Nous savons déjà que l'Algorithme 4 appliqué à $[\overline{N}(v), \{v\}, N(v)]$ permet de calculer la partition modulaire $\mathcal{M}(G, v)$. D'après le Lemme 4.2, les modules ne contenant pas v sont tous contenus dans un module de $\mathcal{M}(G, v)$. Pour montrer la correction de l'Algorithme 12, il suffit donc de prouver que la propriété

$$\Pi = \textit{tout module fort contenant } v \textit{ est un intervalle de } \mathcal{P}$$

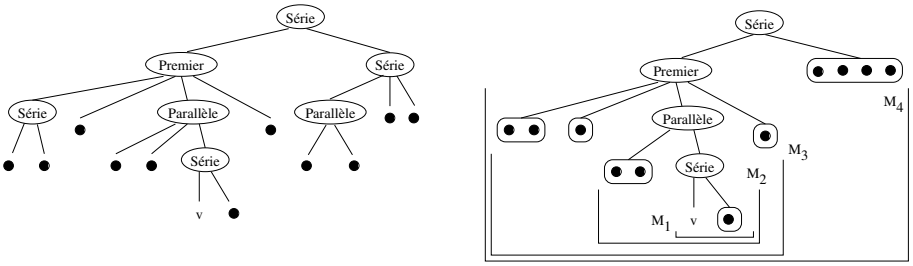

Fig. 5.2. Organisation de l'arbre de décomposition modulaire $MD(G)$ de sorte que les sommets voisins de v soient à droite de v et ceux non-voisins soient à gauche. L'arbre de gauche fait aparaître les modules de $\mathcal{M}(G, v)$ ainsi que les modules forts M_1, M_2, M_3 et M_4 contenant M. La partition $\mathcal{M}(G, v)$ ordonnée de la gauche vers la droite est obtenue au cours de l'Algorithme 12. Il faut ensuite calculer récursivement une permutation factorisante pour chaque module de $\mathcal{M}(G, v)$.

est un invariant. Π est clairement vérifiée par la partition initiale $\mathcal{P} = [\overline{N}(v), \{v\}, N(v)]$. Supposons qu'elle soit aussi vérifiée par la partition $\mathcal{P}$ et soit $\mathcal{P}' = \mathit{Affiner}(\mathcal{P}, N(x))$ la partition obtenue en utilisant les règles d'affinage de l'Algorithme 12. Supposons sans perte de généralité que $x \in \overline{N}(v)$. Soit M un module fort contenant v. Nous avons deux cas à considérer :

- $x \notin M$: Soit $\mathcal{X}$ est une partie de $\mathcal{P}$ telle que $\mathcal{X} \perp N(x)$. Aucun sommet $y \in \mathcal{X} \cap N(x)$ n'appartient à M, sinon x serait un casseur pour v et y.
- $x \in M$: Soit $\mathcal{X}$ est une partie de $\mathcal{P}$ telle que $\mathcal{X} \perp N(x)$. Si $\mathcal{X} \subset N(v)$, alors tout sommet $y \in \mathcal{X} \cap \overline{N}(x)$ appartient à M, sinon un tel sommet y serait un casseur pour x et v. De même, si $\mathcal{X} \subset \overline{N}(v)$, alors tout sommet $y \in \mathcal{X} \cap N(x)$ appartient à M.

On en déduit dans les deux cas que si $\mathcal{P}$ vérifie Π, alors les règles d'affinage garantissent que M reste un intervalle de $\mathcal{P}'$.

Concernant l'analyse de complexité, c'est essentiellement la même que celle de l'Algorithme 10. En effet, elle repose entièrement sur celle de l'Algorithme 4. Il suffit de remarquer que la précaution de relancer récursivement l'algorithme dans les modules $\mathcal{X} \in \mathcal{M}(G, v)$ avec le dernier pivot utilisé pour la partie $\mathcal{X}$ garantit que le voisinage de chaque module est utilisé au plus $\log n$ fois. Puisque le test pour la règle d'affinage a un coût constant, la complexité globale est donc $O(n + m \log n)$. $\square$

5.2 Le cas des cographes

Une étape préliminaire à la compréhension de l'algorithme linéaire de calcul d'une permutation factorisante d'un graphe est de restreindre le problème aux cographes. Pour cette famille, il est possible d'obtenir un algorithme linéaire de décomposition modulaire basé sur l'affinage de partition.

Rappelons que les cographes sont les graphes totalement décomposables pour la décomposition modulaire (voir Chapitre 2 pour la définition et les propriétés de cette famille de graphes). Leur arbre de décomposition modulaire ne contient que des nœuds dégénérés.

Contrairement à l'Algorithme 12, nous allons contraindre le processus d'affinage de partition ordonnée à n'utiliser qu'un seul sommet (ou *pivot*) par partie. Il est donc possible qu'un pivot ait été utilisé pour chaque partie et que la partititon courante $\mathcal{P}$ ne soit pas $\mathcal{M}(G, v)$. Le Lemme 4.2 ne peut donc pas s'appliquer et le processus ne peut pas être relancé récursivement dans chaque partie. Pourtant, une analyse fine des pivots utilisés nous permet de montrer que la permutation factorisante est déjà obtenue pour un certain module fort M contenant le sommet v (voir Lemme 5.6). D'une certaine manière, l'algorithme peut donc être poursuivi sur le sous-graphe quotient $G_{/M}$. L'astuce consiste alors à *"décaller son point de vue"* : l'algorithme est relancé autour d'un nouveau sommet central v'.

Définition 5.4. *Une partition ordonnée $\mathcal{P}$ des sommets d'un graphe G est* centrée *sur le sommet v ssi*

1. *tout module fort de G contenant v est un intervalle de $\mathcal{P}$;*
2. *tout module fort de G ne contenant pas v est inclus dans une partie de $\mathcal{P}$.*

Nous dirons aussi qu'une partition $\mathcal{P}$ est bloquée *si toute partie est* utilisée *(i.e. contient un* pivot*); sinon $\mathcal{P}$ est* libre.

Notation 5.5 *Soit $\mathcal{P}$ une partition ordonnée centrée sur le sommet v. La partie $\mathcal{X}_l \in \mathcal{P}$ est $\mathcal{X}_l = max_{\mathcal{X}\in\mathcal{P}}\{\mathcal{X} \mid \mathcal{X} <_{\mathcal{P}} v$ et $|\mathcal{X}| > 1\}$. Respectivement nous définissons $\mathcal{X}_r = min_{\mathcal{X}\in\mathcal{P}}\{\mathcal{X} \mid \mathcal{X} >_{\mathcal{P}} v$ et $|\mathcal{X}| > 1\}$. Si ces parties sont utilisées, leurs pivots sont notés respectivement p_l et p_r.*

Lemme 5.6. *Soit $\mathcal{P}$ une partition ordonnée bloquée et centrée sur v, des sommets d'un graphe. Alors :*

- *si $p_l p_r \notin E$, alors $\forall x \in M(v, \overline{p_r})^1$, $\{x\} \in \mathcal{P}$;*
- *si $p_l p_r \in E$, alors $\forall x \in M(v, \overline{p_l})$, $\{x\} \in \mathcal{P}$.*

Preuve. Considérons le cas où $p_l p_r \notin E$. L'autre cas se prouve de manière similaire. Puisque $p_l p_r \notin E$, on a $\mathcal{X}_l \cap M(v, \overline{p_r}) = \emptyset$ (sinon p_r serait un casseur de $M(v, \overline{p_r})$). Donc par définition de $\mathcal{X}_l$, pour tout sommet $x \in \overline{N}(v) \cap M(v, \overline{p_r})$, $\{x\}$ est une partie de $\mathcal{P}$.

Supposons qu'il existe $x \in N(v) \cap M(v, \overline{p_r})$. Par hypothèse sur $\mathcal{P}$, il n'est pas possible que $p_r <_{\mathcal{P}} x$. Puisque x existe, $M(v, \overline{p_r}) \neq \{v\}$. Puisque $vp_r \in E$, $m(v, p_r)$ est associé à un nœud série de $MD(G)$ et donc $M(v, \overline{p_r})$ à un nœud parallèle. Il existe donc $y \in \overline{N}(v) \cap M(v, \overline{p_r})$ tel que $\forall x \in N(v) \cap M(v, \overline{p_r})$, $yx \notin E$. Or nous avons prouvé que $\{y\}$ est une partie de $\mathcal{P}$. Puisque $\mathcal{P}$ est stable pour le voisinage de y, $x \notin \mathcal{X}_r$. Par définition de $\mathcal{X}_r$, $\{x\}$ est une partie de $\mathcal{P}$. $\square$

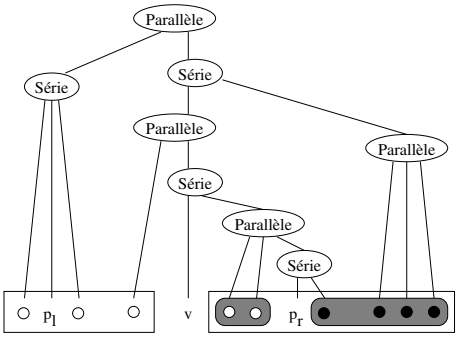

Fig. 5.3. Les sommets p_l et p_r sont les pivots utilisés. La partition $\mathcal{P} = [\overline{N}(v), \{v\}, N(v)]$ est stable pour leurs voisinages. En noir, les voisins du pivot p_r et en blanc, ses non-voisins. La partie $\mathcal{X}_r$ contenant p_r peut être affinée en $[\overline{N}(p_r) \cap \mathcal{X}_r, \{p_r\}, N(p_r) \cap \mathcal{X}_r]$ tout en préservant l'existence d'une permutation factorisante compatible avec $\mathcal{P}$.

Lemme 5.7. *Soit $\mathcal{P}$ une partition ordonnée bloquée et centrée sur v des sommets d'un cographe G. Si tout module non-trivial contenant v contient p_r, alors $\mathcal{P}'$ obtenue en substituant $[\mathcal{X}_r \cap \overline{N}(p_r), \{p_r\}, \mathcal{X}_r \cap N(p_r)]$ à $\mathcal{X}_r$, la partie contenant p_r, est une partition ordonnée centrée sur p_r.*

Preuve. Soit M un module fort ne contenant pas p_r. Alors, par hypothèse, $v \notin M$. Donc M est inclus dans une partie $\mathcal{X}$ de $\mathcal{P}$. Si $\mathcal{X} \neq \mathcal{X}_r$, alors $\mathcal{X} \in \mathcal{P}'$. Sinon soit $M \subseteq N(p_r)$ ou $M \subseteq \overline{N}(p_r)$. M est donc contenu dans une partie de $\mathcal{P}'$.

Soit M un module fort contenant p_r. Si $v \notin M$, alors M est contenu dans $\mathcal{X}_r$ et donc M est un intervalle de $\mathcal{P}'$. Sinon, par hypothèse M est un intervalle de $\mathcal{P}$. Or $N(v) \cap \overline{N}(p_r)$, qui est un sous-ensemble de $\mathcal{X}_r$, est contenu dans M. Donc M reste un intervalle de $\mathcal{P}'$. $\square$

Une illustration du Lemme 5.7 est proposée Figure 5.3. De manière symétrique, nous avons :

Lemme 5.8. *Soit $\mathcal{P}$ une partition ordonnée bloquée et centrée sur v des sommets d'un cographe G. Si tout module non-trivial contenant v contient p_l, alors $\mathcal{P}'$ obtenue en substituant $[\mathcal{X}_l \cap \overline{N}(p_l), \{p_l\}, \mathcal{X}_l \cap N(p_l)]$ à $\mathcal{X}_l$, la partie contenant p_l, est une partition ordonnée centrée sur p_l.*

Tout le matériel est désormais en place pour détailler l'algorithme et en démontrer la correction.

Théorème 5.9. *L'Algorithme 13 calcule en $O(n+m)$ une permutation factorisante d'un cographe G.*

[1] Nous notons $M(x, \overline{y})$ le module fort maximal contenant le sommet x mais pas le sommet y.

Algorithme 13 : *PF-cographe(G, $\mathcal{P}$, v)*

Données : Un cographe $G = (V, E)$, une partition ordonnée $\mathcal{P}$ de V et un sommet $v \in V$
Résultat : Une permutation factorisante de G
début
  si $\mathcal{P}$ *ne contient que des singletons* **alors retourner** $\mathcal{P}$;
  si v *est isolé ou universel* **alors**
    retourner $\mathcal{P}$ =PF-cographe(G, $[V \setminus \{v\}, \{v\}]$, v') avec $v' \neq v \in V$;
  sinon
    Substituer $[\mathcal{X}(v) \cap \overline{N}(v), \{v\}, \mathcal{X}(v) \cap N(v)]$ à $\mathcal{X}(v)$ la partie de $\mathcal{P}$ contenant v;
    tant que *il existe une partie* $\mathcal{X}$ *non-utilisée* **faire**
      $pivot(\mathcal{X}) = x$ avec x un sommet quelconque de $\mathcal{X}$;
      $\mathcal{P}$ =*Affiner*$(\mathcal{P}, N(x))$;
    si $p_l = pivot(\mathcal{X}_l)$ *est voisin de* $p_r = pivot(\mathcal{X}_l)$ **alors**
      $M = (\overline{N}(p_l) \cap N(v)) \cup \{x \mid p_l <_{\mathcal{P}} x \leqslant_{\mathcal{P}} v\}$;
      $\mathcal{P}'$ =PF-cographe($G_{/M}$, $\mathcal{P}_{/M}$, p_l);
    sinon
      $M = (N(p_r) \cap \overline{N}(v)) \cup \{x \mid v \leqslant_{\mathcal{P}} x <_{\mathcal{P}} p_r\}$;
      $\mathcal{P}'$ =PF-cographe($G_{/M}$, $\mathcal{P}_{/M}$, p_r);
1 $\mathcal{P}$ est obtenue en substituant $\mathcal{P}[M]$ à $\{v\}$ dans $\mathcal{P}'$;
    retourner $\mathcal{P}$;
fin

Preuve. Pour démontrer sa correction, il faut de prouver que l'Algorithme 13 manipule toujours une partition ordonnée centrée sur un sommet v. D'après les preuves des Algorithmes 3 et 12, la partition ordonnée bloquée obtenue après affinage successifs de $[\overline{N}(v), \{v\}, N(v)]$ est centrée sur v. Puisque G est un cographe, il est facile de vérifier que M est précisemment le module $M(v, \overline{p_r})$ ou $M(v, \overline{p_l})$ selon le cas. Le Lemme 5.7 ou le Lemme 5.8 s'applique donc. Notons que la partition obtenue en affinant la classe du nouveau centre n'est plus stable et contient plus de parties. Le processus d'affinage peut donc se poursuivre. La propriété d'être centrée reste vérifiée. De plus le processus récursif ne s'arrête que lorsque la partition courante ne contient que des parties singletons. Pour conclure, d'après l'observation suivante, l'opération de substitution de la Ligne 1 est correcte.

Observation 5.10 *Si σ est une permutation factorisante d'un graphe H et τ une permutation factorisante d'un graphe G, alors substituer σ à un sommet x dans τ produit une permutation factorisante du graphe $G_{x \to H}$.*

Concernant l'analyse de complexité, si on admet qu'une opération d'affinage par le voisinage d'un sommet x coûte $O(|N(x)|)$, alors puisque chaque sommet est utilisé un nombre constant de fois, l'algorithme est bien linéaire. □

Remarque 5.11 *Il faut noter que l'Algorithme 13 permet aussi de reconnaître les cographes en temps linéaire. Pour cela, il suffit de lui donner en entrée un graphe quelconque et de tester si à partir de la permutation retournée par l'algorithme, il est possible de construire un ordre d'élimination par sommets jumeaux [HP05]. La complexité de ce test est linéaire.*

5.3 De la permutation factorisante à l'arbre de décomposition

Il existe actuellement plusieurs algorithmes linéaires qui étant donnée une permutation factorisante d'un graphe G calculent l'arbre de décomposition modulaire $MD(G)$. Le premier est dû à Capelle, Habib et de Montgolfier [CHdM02]. En 2005, deux équipes (Bui Xuan, Habib et Paul [BXHP05b] d'un côté, et Bergeron, Chauve, de Montgolfier et Raffinot [BCdMR05] de l'autre) ont proposé de généraliser un algorithme dû à Uno et Yagiura [UY00] qui était utilisé en bioinformatique pour le calcul des intervalles communs à un ensemble de permutations (voir Chapitre 8).

Pour une présentation détaillée de ces algorithmes, nous faisons référence aux articles d'origine [CHdM02, UY00, BXHP05b, BCdMR05]. Décrivons toutefois leurs principes. Ils effectuent plusieurs passes sur la permutation factorisante afin d'en extraire l'arbre de décomposition modulaire. Les approches développées par [CHdM02] d'un côté et par [UY00, BXHP05b, BCdMR05] de l'autre, sont pourtant complémentaires.

5.3.1 L'arbre des fractures

L'idée principale de l'algorithme de [CHdM02] est pour chaque paire (x, y) de sommets consécutifs dans la permutation factorisante σ de repérer deux intervalles (un terminant en x, l'autre débutant en y) qui seront contenus dans tout module contenant x et y. Ces intervalles, lorsqu'ils existent, sont respectivement appelés *fracture gauche* et *fracture droite* de (x, y) et notés $Fg(x, y)$, $Fd(x, y)$:

- $Fg(x, y) = [z, x]$ si z est le casseur de $\{x, y\}$ le plus à gauche de x dans σ;
- $Fd(x, y) = [y, z]$ si z est le casseur de $\{x, y\}$ le plus à droite de y dans σ;

Intuitivement la fracture (gauche ou droite) d'une paire (x, y) est un intervalle contenu dans le plus petit module contenant x et y.

Il est possible de parenthéser la permutation factorisante σ à l'aide de l'ensemble de fractures (gauches et droites) des sommets consécutifs, et d'en déduire un arbre (voir Figure 5.4). Cet arbre est appelé *arbre des fractures*. Il peut être obtenu en deux passes sur la permutation factorisante σ : la première calcule les fractures, la seconde construit l'arbre.

Les auteurs de [CHdM02] montrent que l'arbre des fractures est une bonne estimation de l'arbre de décomposition modulaire. En fait, ils montrent le résultat suivant :

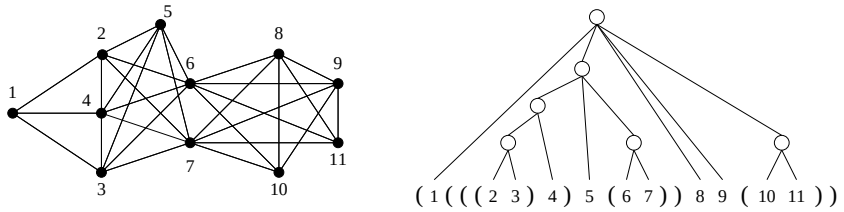

Fig. 5.4. Un graphe $G = (V, E)$ dont une permutation factorisante est $\sigma =$ **1 2 3 4 5 6 7 8 9 10 11** (voir Definition 2.5). La fracture droite de $(3, 4)$ n'existe pas, par contre $Fg(3, 4) = [2, 3]$. Par ailleurs, nous avons $Fd(1, 2) = [2, 7] = Fg(7, 8)$.

Lemme 5.12. *[CHdM02] Soient σ une permutation factorisante d'un graphe G et M un module fort de G. Si M est premier ou si le père de M dans $MD(G)$ est dégénéré, alors il existe un nœud N de l'arbre des fractures réprésentant M (i.e. l'ensemble des feuilles issues de N est exactement le module M).*

Par exemple, Figure 5.4, le module fort $M = \{8, 9, 10, 11\}$ n'est pas représenté alors que tous les autres le sont. Notons qu'il n'y avait aucune garantie pour que $\{2, 3, 4\}$ soit représenté par un nœud de l'arbre des fractures.

Ainsi pour obtenir l'arbre de décomposition modulaire, il suffit de "nettoyer" l'arbre des fractures. Pour cela, Capelle et al. [CHdM02] proposent 4 passes sur la permutation factorisante. La première a pour objectif de retrouver les modules représentés; la deuxième identifie les nœuds inutiles; la troisième détecte d'éventuels modules forts fusionnés en un nœud; et la dernière supprime finalement les nœuds correspondants à des modules non forts. La complexité de chacun de ces parcours est linéaire en la taille du graphe.

5.3.2 L'algorithme de Uno et Yagiura revisité

Un ensemble S d'éléments est un intervalle commun à plusieurs permutations si dans chaque permutation, les éléments de S sont consécutifs (i.e. sont des intervalles). L'algorithme de Uno et Yagiura [UY00] calcule les intervalles communs à deux permutations avec une complexité $O(n+K)$, où K est le nombre d'intervalles communs (K est possiblement quadratique). Nous verrons en détails, Chapitre 8, les liens entre les intervalles communs et les modules. Brièvement les intervalles communs forment une famille faiblement partitive. Il est donc possible de définir des intervalles communs forts et un arbre des intervalles forts. L'algorithme de Uno et Yagiura peut être adapté pour ne calculer que les intervalles communs forts [BXHP05b, BCdMR05]. Le calcul de l'arbre des intervalles forts en découle. La complexité est alors $O(n)$ puisqu'il n'existe au plus que $K = O(n)$ intervalles forts.

La preuve de l'algorithme de Uno et Yagiura est difficile. Dans [BXHP05b] et [BCdMR05] une généralisation aux graphes de l'algorithme de Uno et Yagiura est proposée. Les deux approches sont légèrement différentes. Les auteurs de [BCdMR05] décrivent un nouvel algorithme, plus simple dans le cas des intervalles communs que celui de Uno et Yagiura. Dans [BXHP05b], nous présentons une nouvelle preuve pour l'algorithme de Uno et Yagiura qui permet sa généralisation naturelle aux graphes. Nous décrivons brièvement ci-dessous les idées développées dans [BXHP05b].

L'algorithme de Uno et Yagiura ne parcourt qu'une seule fois la permutation factorisante. Il est décrit ci-dessous. Nous notons abusivement $[i, j]$ l'intervalle de σ constitué des sommets compris entre la position i et la position j (i.e. $[i, j] = \{x \mid i \leqslant \sigma(x) \leqslant j\}$).

Algorithme 14 : Algorithme de Uno et Yagiura [UY00]

Données : Un graphe G et une permutation factorisante σ
Résultat : Les modules facteurs de σ
début
  Soit `Potentiel` une liste vide;
  pour i *de* n *à* 1 **faire**
    (Filtre) supprimer de `Potentiel` les bornes r tq $\forall j \leqslant i$, $[j, r]$ n'est pas un module de σ;
    (Ajout) ajouter i à `Potentiel`;
    (Extraction) parcourir `Potentiel` pour trouver les bornes r tq $[i, r]$ est un module de σ;
fin

A chaque étape, on ne supprime de la liste `Potentiel` que les sommets qui ne sont pas borne gauche d'un module contenant le sommet $\sigma^{-1}(i)$. Sous cette forme, l'algorithme de Uno et Yagiura calcule les modules qui sont facteurs de σ. Notons donc que tous les modules de G ne sont pas calculés et que les modules obtenus ne sont pas nécessairement des modules forts de G.

La gestion de la liste `Potentiel`, i.e. le filtre et l'extraction, peut être faite de manière efficace en comptant le nombre de casseurs de l'intervalle considéré. Soit $[i, j]$ un intervalle de σ, alors $C([i, j])$ est l'ensemble des casseurs de l'ensemble de sommets $[i, j]$ et $c([i, j]) = |C([i, j])|$. Les propriétés suivantes sont fondamentales pour la correction de l'algorithme.

Lemme 5.13. *[UY00]* $[i, j]$ *est un module facteur de* σ *ssi* $c([i, j]) = 0$.

Lemme 5.14. *[UY00, BXHP05b] Si* $c([i, j]) > c([i, j + 1])$*, alors il n'existe pas* $r < i$ *tel que* $[r, j]$ *est un module de* G.

Le dernier lemme dit simplement que si $c([i, j]) > c([i, j + 1])$ alors le sommet $\sigma^{-1}(j+1)$ est un casseur pour l'ensemble de sommet $[i, j]$. Ce sommet

devra donc être compris dans tout module contenant $[i, j]$. Par exemple, sur le graphe de la Figure 5.4 et la permutation $\sigma = \mathbf{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11}$, le sommet 11 est un casseur pour l'intervalle $[9, 10]$. Donc tout module contenant 9 et 10 devra contenir 11. Il n'est donc pas nécessaire de conserver le sommet 10 dans la liste `Potentiel`. Il en est de même avec $[3, 6]$ et $[3, 7]$, 7 est un casseur pour $[3, 6]$.

En fait, si on souhaite se restreindre au calcul des modules forts, il faut légèrement modifier l'algorithme de Uno et Yagiura. Une première passe permet alors de calculer les modules forts à droite (modules facteurs de σ qui ne sont pas chevauchés sur leur borne droite). Puis une seconde passe élimine ceux qui ne sont pas modules forts à gauche. Pour plus de détails, se référer aux articles [UY00, BXHP05b].

5.4 Notes bibliographiques

Le calcul d'une permutation factorisante en temps linéaire (sans passer par l'arbre de décomposition modulaire) reste un problème ouvert. En 2004, nous avons publié un algorithme [HdMP04] à base d'affinage de partition. Cet algorithme n'est malheureusement pas juste dans sa forme publiée. Nous pouvons le corriger mais le prix à payer est une trop grande complication de l'algorithme. Rappelons qu'il existe déjà plusieurs algorithmes linéaires pour la décomposition modulaire d'un graphe. Un nouvel algorithme n'aurait de sens que s'il est plus simple que ceux déjà connus. Ce ne serait pas le cas pour la version corrigée de [HdMP04]. Nous laissons donc pour l'instant ce problème ouvert.

Par ailleurs, je pense qu'il devrait être possible grâce à l'affinage de partition de calculer une orientation transitive d'un graphe de comparabilité sans avoir recours au calcul intermédiaire de l'arbre de décomposition modulaire tel que c'est fait dans [MS99].

Tarte tomates et mozzarella

Ingrédients : tomates pelées, mozzarella, 1 gousse d'ail, basilic, origan, huile d'olive.

Faire revenir les tomates peles dans l'huile d'olive, y ajouter l'ail minc, et le basilic hach. Laisser mijoter à feu doux jusqu'à ce que la sauce devienne épaisse.

Couvrir le fond du plat avec une feuille alu. Disposer la mozzarella découpée en large rondèles sur le plat et étaler la sauce tomate par dessus. Recouvrir la préparation avec la pâte. Mettre au four chaud pendant 25 minutes (250). Démouler la tarte et la sopoudrer avec l'origan.

6

Parcours en largeur lexicographique

Le parcours en largeur lexicographique (LexBFS) a été proposé par Rose, Tarjan et Lueker [RTL76] pour la reconnaissance des graphes triangulés [BLS99]. L'importance de cet algorithme est très vite reconnue puisqu'il est présenté dans le livre référence de Golumbic [Gol80]. Malheureusement, l'implémentation alors proposée était sans doute trop compliquée. LexBFS est resté "dans les cartons" pendant près d'une vingtaine d'années. Au milieu des années 90, il est replacé sur le devant de la scène par Corneil, Olariu et Stewart [COS97] pour le calcul d'une paire dominante dans les graphes sans triplet astèroïde [BLS99]. Une implémentation utilisant l'affinage de partition n'a été proposée qu'en 1998 [HPV98]. Depuis un nombre considérable d'algorithmes basés sur LexBFS ont vu le jour. La liste est trop longue pour être présentée ici. Citons seulement les différents algorithmes de reconnaissance des graphes d'intervalles [HM91, HMPV00, COS98].

L'objet de ce chapitre est donc de présenter LexBFS et ses liens étroits avec la décomposition modulaire. Après les définitions et la description de cet algorithme, nous montrerons en quoi LexBFS est utile dans le problème de l'orientation transitive d'un graphe de comparabilité [HMPV00]. Puis nous décrirons un algorithme récent extrêment simple pour la reconnaissance des cographes [BCHP03]. Cet algorithme n'utilise que deux parcours LexBFS, le premier sur le graphe, l'autre sur son complémentaire.

6.1 L'algorithme LexBFS

De manière générale, un parcours en largeur d'un graphe $G = (V, E)$ visite l'ensemble des sommets en respectant les "couches" de distance depuis le sommet source x. C'est à dire que les sommets à distance $k < k'$ de x sont visités avant les sommets à distance k' de x. Mais aucune contrainte n'est posée sur l'ordre de visite entre deux sommets équidistants de x [1]. Le *parcours en*

[1] De ce point de vue, l'implémentation à l'aide d'une file ne permet pas d'obtenir n'importe quel parcours en largeur.

largeur lexicographique permet de contraindre le choix du prochain sommet visité. A chaque étape, on sélectionne un sommet dont les voisins ont été visités les plus tôt dans le parcours.

Notons que nous considérons ici que le résultat du parcours d'un graphe $G = (V, E)$ est l'ordre selon lequel les sommets ont été visités : c'est donc une permutation $\sigma : V \to [1, n]$ telle que $\sigma(y) = i$ signifie que y est le i-ème sommet visité (et inversement $\sigma^{-1}(i) = y$). Nous noterons $x <_\sigma y$ le fait que $\sigma(x) < \sigma(y)$.

L'implémentation présentée dans [RTL76] et [Gol80] affecte à chaque sommet une étiquette (une liste triée initialement vide). Lorsqu'un sommet v est sélectionné, il ajoute $\sigma^{-1}(x)$ à l'étiquette de chacun de ses voisins non-visités. A chaque étape, on visite un sommet dont l'étiquette est maximum pour l'ordre lexicographique. Obtenir une complexité linéaire en maintenant et comparant ces étiquettes, n'est pas une tâche aisée. Par contre, l'affinage de partition permet une implémentation très simple de LexBFS (voir Algorithme 15). Curieusement cette implémentation n'a été proposée (ou redécouverte ?) que relativement récemment [HMPV00]. Nous verrons dans la suite tous les avantages de cette version de LexBFS. En particulier, elle permet de calculer, en temps linéaire en la taille d'un graphe G, un ordre LexBFS de son complémentaire $\overline{G}$. Puisque le résultat du parcours est une permutation sur les sommets du graphe, l'algorithme d'affinage doit manipuler des partitions ordonnées.

Algorithme 15 : LexBFS(G, x)

Données : Un graphe $G = (V, E)$ et un sommet $x \in V$
Résultat : Un ordre σ sur les sommets de V
début
  $\mathcal{P} \leftarrow [V]$ (une partition triée de V);
  $i \leftarrow 1$;
  tant que $i \neq n$ **faire**
1 Extraire un sommet x de $\mathcal{X}_i$, la i-ème partie de $\mathcal{P}$;
    $\sigma(x) = i$;
    Remplacer dans $\mathcal{P}$, $\mathcal{X}_i$ par $[\{x\}, N(x) \cap \mathcal{X}_i, \overline{N}(x) \cap \mathcal{X}_i]$;
    pour chaque *partie* $\mathcal{X} \in \mathcal{P}$ **faire**
      si $\mathcal{X} \cap N(x) \neq \emptyset$ *et* $\mathcal{X} \cap N(x) \neq \mathcal{X}$ **alors**
        remplacer $\mathcal{X}$ par $[\mathcal{X} \cap N(x), \mathcal{X} \setminus N(x)]$;
    $i \leftarrow i + 1$;
fin

Les ordres LexBFS sont caractérisés par des "configurations interdites" (décrites ci-dessous). Nous serons amenés à utiliser cette caractérisation à différentes reprises. En effet, elle permet souvent de simplifier des preuves existantes.

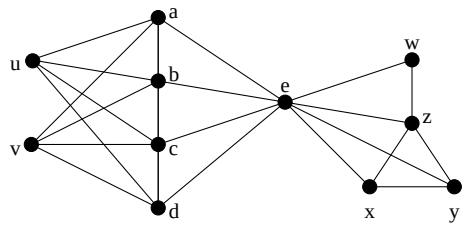

Fig. 6.1. Un graphe dont un ordre LexBFS σ est calculé Tableau 6.1.

$\sigma(v)$	v	$N^+(v)$	Partition
			a u b e v c d w z y x
1	a	{u b v e}	a u b e v c d w z y x
2	u	{b c d}	a u b e v c d w z y x
3	b	{v c e}	a u b e v c d w z y x
4	e	{c d w z y x}	a u b e v c d w z y x
5	v	{c d}	a u b e v c d w z y x
6	c	{d}	a u b e v c d w z y x
7	d	{ }	a u b e v c d w z y x
8	w	{z}	a u b e v c d w z y x
9	z	{y x}	a u b e v c d w z y x
10	y	{x}	a u b e v c d w z y x
11	x	{ }	a u b e v c d w z y x

Tableau 6.1. Pas à pas, le calcul d'un ordre LexBFS du graphe G de la Figure 6.1. $N^+(v)$ désigne l'ensemble des voisins non-numérotés de v. L'ordre obtenu est σ : **a u b e v c d w z y x**

Lemme 6.1. *[DNB96] Soit σ un ordre total des sommets d'un graphe $G = (V, E)$. Alors σ est un ordre LexBFS ssi pour tout triplet de sommets $x <_\sigma y <_\sigma z$ tel que $xy \notin E$ et $xz \in E$, il existe un sommet $v <_\sigma x$ tel que $vy \in E$ et $vz \notin E$.*

6.1.1 Les couches lexicographiques

Etant donné un ordre σ sur V, pour chaque sommet x, nous définissons le sous-ensemble du voisinage

$$N_i(x) = \{u \mid u \in N(x) \text{ et } \sigma(u) < i\}$$

Définition 6.2. *Une* couche lexicographique *(ou ensemble de sommets égaux pour l'ordre lexicographique) d'un ordre LexBFS σ est un ensemble de sommets*

consécutifs $S = \{v \mid i \leqslant \sigma(v) \leqslant j\}$ *tel que pour tout* $v \in S$, $N_i(v) = N_i(\sigma^{-1}(i))$ *et pour tout* v *avec* $j < \sigma(v)$, $N_i(v) \neq N_i(\sigma^{-1}(i))$.

L'ensemble des sommets V est une couche lexicographique, ainsi que le voisinage du premier sommet de σ. Si le graphe est une clique (ou un stable), alors tout ensemble de sommets consécutifs est une couche lexicographique.

Notation 6.3 *Pour chaque sommet* x*, il est possible de définir la couche lexicographique maximale* $S(x)$ *dont le premier sommet est* x.

Remarquons que toute couche lexicographique $S(x)$ est constituée de x suivi de son voisinage dans la couche puis de son voisinage dans la couche. Ainsi pour l'ordre σ (voir Table 6.1), $S(a) = V$, $S(u) = \{u, b, e, v\}, \ldots$, $S(w) = \{w, z, y, x\}$. L'ensemble des couches lexicographiques $S(.)$ possède une structure emboîtée, ou arborescente (voir Equation 6.1).

$$\mathbf{a}\ \boxed{\mathbf{u}\ \boxed{\mathbf{b}}\ \boxed{\mathbf{e}\ \mathbf{v}}}\ \boxed{\mathbf{c}\ \boxed{\mathbf{d}}}\ \boxed{\mathbf{w}\ \boxed{\mathbf{z}}\ \boxed{\mathbf{y}\ \mathbf{x}}} \tag{6.1}$$

Lemme 6.4. *[COS98] Soit* S *une couche lexicographique d'un ordre LexBFS* σ *d'un graphe* $G = (V, E)$*. Alors* σ *restreint aux sommets de* S*, noté* $\sigma[S]$ *est un ordre LexBFS du sous-graphe induit* $G[S]$.

Preuve. Supposons par contradiction que $\sigma[S]$ ne soit pas un ordre LexBFS. D'après le Lemme 6.1, il existe un triplet x, y, z de sommets de S tel que $x <_\sigma y <_\sigma z$, $xz \in E$ et $xy \notin E$ et pour tout sommet $v \in S$ tel que $v <_\sigma x$, $vy \in E$ implique $vz \in E$. Puisque σ est un ordre LexBFS de G, il existe $u \in V$ avec $u <_\sigma x$ tel que $uy \in E$ et $uz \notin E$. Donc $v \notin S$ et $v <_\sigma S$. Les sommets y et z n'appartiennent donc pas à la même couche lexicograhique : contradiction. □

6.1.2 Règles de tie-break

A chaque étape, il est possible que plusieurs sommets soient éligibles à la numérotation. Il peut être intéressant de contraindre davantage le choix afin de bénéficier de propriétés supplémentaires. Différentes règles de tie-break sont possibles. Nous présentons les deux plus courantes : la *règle plus* et la *règle minus*. Pour ces deux règles, on suppose qu'un ordre préliminaire π sur les sommets est donné. A chaque étape i, le sommet $\sigma^{-1}(i)$ sera sélectionné parmi $S = S(\sigma^{-1}(i))$ en fonction de π. Dans la règle plus, on préfèrera le sommet de S maximum dans π. Dans la règle *minus*, on choisira le sommet de S minimum dans π.

Ces deux règles sont très simples à implémenter. Il suffit d'initialiser la liste initiale des sommets en respectant π pour la règle *minus* et en respectant π renversée pour la règle *plus*. Le choix du sommet à numéroter porte alors systématiquement sur le premier de la partie correspondant à $S(\sigma^{-1}(i))$.

L'Equation 6.2 décrit l'ordre LexBFS de $\overline{G}$ obtenu avec la règle *minus* à partir de l'ordre σ du Tableau 6.1.

$$\mathbf{LexBFS}^{-}(\overline{G},\sigma) = \mathbf{a}\ \boxed{\mathbf{c}\ \boxed{\mathbf{w}\ \boxed{\mathbf{y}\ \mathbf{x}}\ \boxed{\mathbf{z}}}\ \boxed{\mathbf{d}}}\ \boxed{\mathbf{b}}\ \boxed{\mathbf{u}\ \boxed{\mathbf{v}}}\ \boxed{\mathbf{e}} \tag{6.2}$$

Ces règles ont été utilisées dans des algorithmes à base de plusieurs parcours LexBFS. La règle *plus* intervient pour de la reconnaissance des graphes d'intervalles [Sim91, COS98]. La règle *minus* permet la reconnaissance des cographes à l'aide de deux parcours LexBFS.

6.2 LexBFS et l'orientation transitive

Les liens entre l'orientation transitive d'un graphe de comparabilité et la décomposition modulaire, mis en évidence par Gallai [Gal67], ont été présentés au Chapitre 2. Nous présentons ici des propriétés portant sur le rôle de LexBFS dans le problème de l'orientation transitive. L'ensemble de ces propriétés permet d'espèrer un jour l'existence d'un algorithme d'orientation transitive uniquement basé sur LexBFS, et pourquoi pas un algorithme de décomposition modulaire à partir de LexBFS.

En fait, le premier résultat dans cette direction est peut-être l'algorithme de Korte et Möhring [KM89] simplifiant les PQ-arbres [BL76] pour la reconnaissance des graphes d'intervalles.

Définition 6.5. *Un graphe $G = (V, E)$ est un* graphe d'intervalles *si c'est le graphe d'intersection d'une famille d'intervalles.*

Les graphes d'intervalles sont des *graphes de co-comparabilité* (les graphes dont le complémentaire admet une orientation transitive) et sont caractérisés par l'existence d'un *arrangement consécutif de leurs cliques maximales*, à savoir un ordre total des cliques maximales tel que pour tout sommet x, les cliques maximales contenant x sont consécutives.[2] Notons la totale correspondance entre un arrangement consécutif des cliques maximales d'un graphe d'intervalles et un réaliseur (famille d'intervalles) de ce graphe.

Lemme 6.6. *[KM89] La clique maximale contenant le dernier sommet x d'un ordre LexBFS est la dernière clique d'un arrangement consécutif.*

[2] En fait, LexBFS est très important pour les graphes d'intervalles. En 1998, Corneil, Olariu et Stewart ont proposé un algorithme pour la reconnaissance des graphes d'intervalles basé sur plusieurs parcours consécutifs afin d'obtenir un ordre sur les sommets caractéristique des graphes d'intervalles. Dans le même temps, nous avons présenté un autre algorithme de reconnaissance utilisant un parcours LexBFS et l'affinage de partition [HPV98].

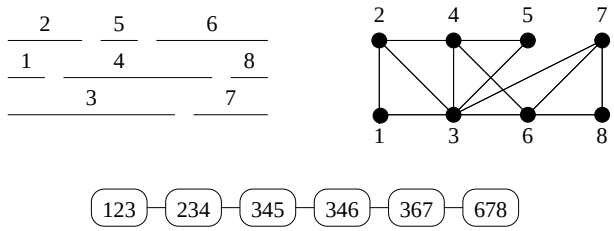

Fig. 6.2. Un graphe d'intervalles n'admettant, au renversement près, qu'un seul arrangement consécutif de ses cliques maximales. Par ailleurs sur ce graphe, tout ordre LexBFS se termine soit sur le sommet 1, soit sur le sommet 8. Par exemple, $\sigma = \mathbf{4\ 5\ 3\ 2\ 6\ 1\ 7\ 8}$ est un ordre LexBFS.

En fait, le lemme précédent signifie que x est une source d'une orientation transitive de $\overline{G}$. Pour obtenir une orientation transitive de $\overline{G}$, il suffit d'orienter les non-arêtes à partir d'un réaliseur (ou d'un arrangement consécutif des cliques) de la manière suivante : $xy \in \overline{E}$ est orientée $\overrightarrow{yx}$ ssi l'intervalle de x précède celui de y. Ainsi un sommet n'appartenant qu'à la dernière clique maximale est effectivement une source. Ce résultat se généralise à tout graphe de co-comparabilité.

Lemme 6.7. *[HMPV00] Soit σ un ordre LexBFS d'un graphe $G = (V, E)$ de co-comparabilité. Si x est le dernier sommet de σ, alors il existe une orientation transitive de $\overline{G}$ pour laquelle x est une source.*

Nous omettons la preuve du Lemme 6.7 mais présentons les propriétés nécessaires à sa démonstration. Elles montrent le "bon comportement" de LexBFS vis-à-vis des modules et nous seront utiles pour la suite. La même propriété avait déjà été observée pour les couches lexicographiques (voir Lemme 6.4).

Lemme 6.8. *[HMPV00] Soit σ un ordre LexBFS d'un graphe $G = (V, E)$. Si M est un module de G alors la restriction $\sigma[M]$ de σ aux sommets de M est un ordre LexBFS du sous-graphe induit $G[M]$.*

Preuve. Puisque M est un module aucun sommet $v \notin M$ ne permet de distinguer des sommets de M. Donc l'ordre relatif entre les sommets de M ne peut jamais être influencer par un sommet $v \notin M$. $\square$

Lemme 6.9. *[HMPV00] Soit σ un ordre LexBFS d'un graphe $G = (V, E)$. Si $\mathcal{P}$ est une partition modulaire de G, alors la restriction $\sigma_{/\mathcal{P}}$ de σ aux premiers sommets de chaque partie de $\mathcal{P}$ est un ordre LexBFS du graphe quotient $G_{/\mathcal{P}}$.*

Preuve. Soient $\mathcal{P} = \{\mathcal{X}_1, \ldots, \mathcal{X}_k\}$ la partition modulaire et x_i le premier sommet de la partie $\mathcal{X}_i$ dans σ. Notons que $G[S = \{x_1, \ldots, x_k\}]$ est isomorphe à $G_{/\mathcal{P}}$. Supposons que la restriction $\sigma_{/\mathcal{P}}$ ne soit pas un ordre LexBFS. D'après le Lemme 6.1, il existe un triplet u, v, w de sommets de S tel que $u <_\sigma v <_\sigma w$,

$uw \in E$ et $uv \notin E$ et pour tout sommet $y \in S$ tel que $y <_\sigma u$, $yv \in E$ implique $yw \in E$. Puisque σ est un ordre LexBFS de G, il existe $x \notin S$ avec $x <_\sigma u$ tel que $xv \in E$ et $xw \notin E$. Soient $\mathcal{X}_i$ la partie de $\mathcal{P}$ contenant x. Notons que par construction w n'appartient pas au module $\mathcal{X}_i$. Puisque $xv \in E$, nous avons aussi $x_i v \in E$. Or nous avons montré que $x_i v \in E$ implique $x_i w \notin E$. Le sommet w est donc adjacent à x mais pas à x_i, deux sommets d'un même module : contradiction. □

Le lemme suivant sera très utile pour les algorithmes de décomposition modulaire que nous présentons dans ce chapitre. Il montre que tout module fort contenant le dernier sommet x d'un ordre LexBFS σ est un intervalle de σ. Il est donc possible pour ce sommet x de reconstruire la branche de $MD(G)$ allant de la racine à la feuille x.

Lemme 6.10. *Soit x le dernier sommet d'un ordre LexBFS σ d'un graphe $G = (V, E)$. Pour tout module fort M contenant x, il existe un sommet y tel que $S(y) = M$.*

Preuve. Soit $y = \sigma(i)$, le sommet ayant séparé M dans deux parties différentes. Supposons que $y \notin M$. Puisque y sépare M dans deux parties différentes, M n'est pas uniforme par rapport à y, contradiction. Donc nous avons $y \in M$. Puisque $x = \sigma(n) \in M$ et puisque y est le premier sommet séparant M, tous les sommets $\{\sigma(i), \ldots, \sigma(n)\}$ appartiennent à la même partie $\mathcal{X}_i = M$. La propriété est satisfaite. □

Pour le graphe de la Figure 6.1 et l'ordre LexBFS σ du Tableau 6.1, les modules contenant le dernier sommet x sont : $S(y) = \{y, x\}$, $S(w) = \{w, z, y, x\}$ et $S(a) = V$.

Pour conclure, mentionnons le résultat peut être le plus prometteur mais aussi le plus difficile à appréhender algorithmiquement entre les graphes de comparabilité et LexBFS. Tout ordre total σ sur les sommets d'un graphe définit naturellement une orientation : chaque arête est orientée de la plus petite de ses deux extrémités vers la plus grande. Le lemme suivant montre que pour les graphes de co-comparabilité, il existe toujours un ordre LexBFS qui définit une orientation transitive du graphe complémentaire.

Lemme 6.11. *[COS99] Pour tout graphe G de co-comparabilité, il existe un ordre LexBFS σ tel que $\forall u <_\sigma v <_\sigma w$, si $uw \in E$, alors $uv \in E$ ou $vw \in E$*

Fig. 6.3. Si G est un graphe de cocomparabilité, alors il existe un ordre LexBFS qui définit une orientation transitive des non-arêtes.

Nous terminerons donc cette section avec le problème ouvert suivant :

Problème 1: LexBFS - orientation transitive

Données : Un graphe de comparabilité G
Question : Calculer en temps linéaire un ordre LexBFS qui définisse une orientation transitive de G.

6.3 Reconnaissance des cographes

Tester si un graphe est un cographe semble un problème bien maîtrisé maintenant puisque le premier algorithme linéaire date de 1985 [CPS85] et aussi que les algorithmes linéaires de décomposition modulaire [CH94, MS94a] peuvent accomplir cette tâche. Toutefois, ce problème peut être résolu de manière extrêmement simple par l'Algorithme 16 ci-dessous. Il est peut-être plus intéressant pour notre propos sur la décomposition modulaire de noter que deux parcours LexBFS suffisent à construire le coarbre d'un cographe en temps linéaire à l'aide d'un algorithme du type de celui de Ehrenfeucht et al. (voir Chapitre 4). Nous ne présenterons pas dans cette section les détails algorithmiques, ni l'analyse précise des complexités. Par contre, nous tenterons de fournir au lecteur des indications pour les preuves de correction.

Algorithme 16 : LexBFS-Cographe

Données : Un graphe $G = (V, E)$
Résultat : Vrai si G est un cographe, un P_4 sinon
début
  $\sigma \leftarrow$ **LexBFS**(G);
  $\overline{\sigma} \leftarrow$ **LexBFS**$^-(\overline{G}, \sigma)$;
  si σ *et* $\overline{\sigma}$ *vérifient la propriété d'inclusion des voisinages* **alors**
    Retourner G est un cographe;
  sinon Construire un P_4;
fin

Avant de prouver la correction de l'Algorithme 16, spécifions quelques notations et définitions. Nous terminerons cette section en montrant comment adapter simplement cet algorithme pour construire un coarbre si le graphe en entrée est un cographe. L'ensemble des concepts et propriétés sera illustré sur le cographe de la Figure 6.4.

Notation 6.12 *Soit σ un ordre LexBFS calculé. Alors pour tout sommet x, nous définissons :*

1. *$S^A(x)$ la couche lexicographique $S(x) \cap N(x)$ de σ;*
2. *$S^N(x) = S(x) \cap \overline{N}(x)$ et $S_1(x), \ldots, S_k(x)$ les couches lexicographiques maximales contenues dans $S^N(x)$.*

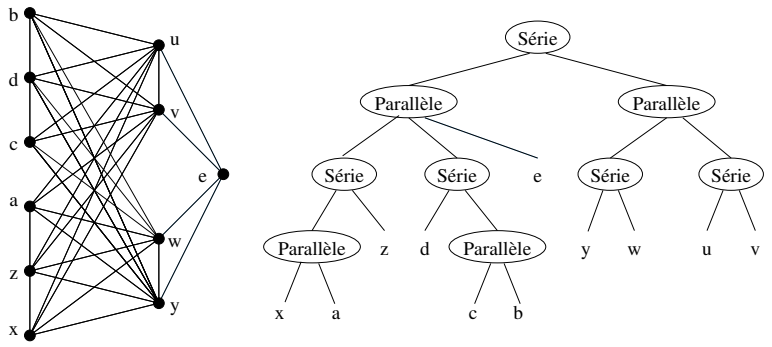

Fig. 6.4. Un cographe et son coarbre dont les ordres LexBFS calculés par l'Algorithme 16 sont : $\sigma = \mathbf{x\ y\ w\ z\ u\ v\ a\ d\ c\ b\ e}$ et $\overline{\sigma} = \mathbf{x\ a\ d\ e\ c\ b\ z\ y\ u\ v\ w}$

De manière similaire, pour l'ordre $\overline{\sigma}$ (calculé dans l'Algorithme 16), nous noterons $\overline{S}(x)$ la couche lexicographique maximale débutant par x, $\overline{S^A}(x) = \overline{S}(x) \cap \overline{N}(x)$, $\overline{S^N}(x) = \overline{S}(x) \cap N(x)$ et $\overline{S_1}(x), \ldots, \overline{S_k}(x)$ les couches lexicographiques maximales contenues dans $\overline{S^N}(x)$.

Par exemple, pour les ordres LexBFS σ et $\overline{\sigma}$ du cographe de la Figure 6.4, nous avons $S^A(x) = \{y, w, z, u, v\}$, $S_1(x) = \{a\}$, $S_2(x) = \{d, c, b, e\}$ et $S^N(x) = \{a, d, e, c, b\}$, $\overline{S}_1(x) = \{z\}$, $\overline{S}_2(x) = \{y, u, v, w\}$.

Si S est une couche lexicographique dont le premier sommet est x, alors le voisinage de S est noté $N(S) = N_i(x)$ avec $i = \sigma(x)$.

6.3.1 La propriété d'inclusion des voisinages

Commençons par présenter quelques propriétés élémentaires de LexBFS sur les cographes. Dans un ordre LexBFS σ, le *voisinage local* d'une couche lexicographique $S_i(x)$ est défini par

$$N^l(S_i(x)) = N(S_i(x)) \cap S(x)$$

Lemme 6.13. *Soit σ un ordre LexBFS d'un cographe G. Alors pour tout sommet v et tout $i < j$, nous avons:*

1. $\forall x \in S_i(v),\ \forall y \in S_j(v),\ xy \notin E$ *et*
2. $N(S_j(v)) \subset N(S_i(v))$

En fait le Lemme 6.13 a déjà été décrit en d'autres termes dans [Dah95a]. Nous pouvons d'ores et déjà énoncer une conséquence directe de cette propriété : chaque couche lexicographique $S_i(v)$ est un module de G et en particulier :

Corollaire 6.14. *Si v est le premier sommet de σ, alors toute couche lexicographique $S_i(v)$ est un module de la partition modulaire $\mathcal{M}(G, v)$.*

La seconde propriété du Lemme 6.13 est en fait très importante pour les cographes et permet de les caractériser.

Définition 6.15. *Un ordre LexBFS σ d'un graphe $G = (V, E)$ satisfait la* propriété d'inclusion des voisinages *ssi*

$$\forall x \in V,\ \forall i < j,\ N^l(S_j(x)) \subset N^l(S_i(x))$$

Théorème 6.16. *[BCHP03] Soit σ un ordre LexBFS d'un graphe $G = (V, E)$. Alors G est un cographe ssi σ et $\overline{\sigma} = LexBFS^{-}(\overline{G}, \sigma)$ vérifient la propriété d'inclusion des voisinages (dans les graphes G et $\overline{G}$ respectivement).*

Preuve. ($\Rightarrow$) Supposons que G soit un cographe. Notons que $N^l(S_i(v)) = N(S_i(v)) \cap (\{v\} \cup S^A(v))$. Ainsi le Lemme 6.13 implique la propriété d'inclusion des voisinages sur σ. Puisque $\overline{G}$ est un cographe, il en est de même pour $\overline{\sigma}$.

($\Leftarrow$) Supposons que G ne soit pas un cographe, i.e., il existe un P_4 induit. Le théorème repose sur les deux propriétés suivantes dont nous omettons les preuves.

Fait. Soit $p = \{a, b, c, d\}$ un P_4 de G dont les extrémités sont a et d. si $S(a)$ contient p, alors il existe deux entiers i, j tels que $S_i(a)$ et $S_j(a)$ ne respectent pas la propriété d'inclusion des voisinages.

Fait. Soit $p = \{a, b, c, d\}$ un P_4 de G. Si $S(v)$ avec $v \notin \{a, b, c, d\}$ est la couche lexicographique minimale de σ contenant p, alors il existe deux entiers i, j tels que $S_i(a)$ et $S_j(a)$ ne respectent pas la propriété d'inclusion des voisinages.

Soit $\overline{S}(v)$ la couche lexicographique minimal contenant les sommets du P_4 $\overline{p} = \{a, b, c, d\}$ de $\overline{G}$. Nous devons distinguer 3 cas différents :

1. v est une extrémité, par exemple a, de $\overline{p}$: alors le premier fait appliqué à $\overline{S}(v)$ montre que $\overline{\sigma}$ ne vérifie pas la propriété d'inclusion des voisinages.
2. v n'est pas un sommet de $\overline{p}$: alors le second fait ci-dessus appliqué à $\overline{S}(v)$ montre que $\overline{\sigma}$ ne vérifie pas la propriété d'inclusion des voisinages.
3. v est un sommet de $\overline{p}$, mais pas une extrémité, par exemple b. Puisque $v = b$ est le premier sommet de $S(v)$ dans $\overline{\sigma}$, la règle *minus* implique que v est aussi le premier sommet parmi $\{a, b, c, d\}$ dans σ. Soit $S(u)$ la couche lexicographique minimale de σ contenant p. Si $u = v$ alors le premier fait ci-dessus appliqué à $S(u)$ montre que σ ne vérifie pas la propriété d'inclusion des voisinages. Sinon u n'est pas un sommet de p et puisque $S(u)$ est minimale, le second fait ci-dessus s'applique, montrant que σ ne vérifie pas la propriété d'inclusion des voisinages.

□

Pour conclure, notons qu'il est possible de tester en temps $O(n + m)$ si les ordres σ et $\overline{\sigma}$ vérifient la propriété d'inclusion des voisinages (donc sans

calculer le graphe complémentaire). Par ailleurs, il est possible, si l'un des deux ordres LexBFS ne vérifie la propriété, de trouver en temps $O(n)$ un P_4. Supposons sans perte de généralité que σ n'a pas la propriété requise (pour $\overline{\sigma}$ et $\overline{G}$ un résultat symétrique existe).

Lemme 6.17. *[BCHP03] Soit σ un ordre LexBFS d'un graphe $G = (V, E)$ ne vérifiant pas la propriété d'inclusion des voisinages pour les couches lexicographique $S_i(x)$. Soit j le plus petit entier tel que $N^l(S_{j+1}(x)) \not\subset N^l(S_j(x))$, alors il existe un P_4 induit par les ensembles de sommets*

$$\{x, w, v, u_{j+1}\} \text{ ou } \{u_{j+1}, v, w, u_j\}$$

avec $u_i \in S_i(x)$, $w \in N^l(S_j(x)) \setminus N^l(S_{j+1}(x))$ et $v \in N^l(S_{j+1}(x)) \setminus N^l(S_j(x))$.

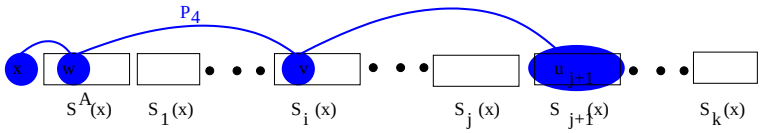

Fig. 6.5. Exemple de configuration permettant de trouver un P_4 lorsque σ ne satisfait pas la propriété d'inclusion des voisinages. Ici $N^l(S_{j+1}(x)) \not\subset N^l(S_j(x))$ et il existe $v \in S_i(x) \cap N(S_{j+1}(x))$.

6.3.2 Construction du coarbre

Rappelons encore une fois que si le problème qui nous intéresse est la reconnaissance des graphes, alors ce qui suit est strictement inutile. Nous allons montrer que dans le cas des cographes, LexBFS permet d'implémenter l'algorithme de Ehrenfeucht et al. (voir Chapitre 4). Plus précisemment le coarbre peut être construit en temps $O(n+m)$ à partir de deux ordres LexBFS σ et $\overline{\sigma} = \text{LexBFS}^-(\overline{G}, \sigma)$. Nous supposerons donc dans cette section que G est un cographe.

Lemme 6.18. *Soient σ un ordre LexBFS d'un cographe G débutant en v et $\overline{\sigma} = \text{LexBFS}^-(\overline{G}, \sigma)$. Alors la partition modulaire $\mathcal{M}(G, v)$ est composée du singleton $\{v\}$, des couches lexicographiqes $S_i(v)$ de σ et des couches lexicographiques $\overline{S_j}(v)$ de $\overline{\sigma}$.*

Preuve. Conséquence directe du Lemme 6.13 et du fait que tout module appartenant à la partition modulaire $\mathcal{M}(G, v)$ est inclus soit dans $N(v)$, soit dans $\overline{N}(v)$. □

Afin d'établir la correspondance entre les couches lexicographiques et les sous-arbres du coarbre de G, introduisons quelques notations.

Notation 6.19 *Soit x un sommet de G, alors 0_i^x (resp. 1_j^x) désigne le i-ème nœud parallèle (resp. j-ème nœud série) sur le chemin allant de la feuille correspondant au sommet x à la racine du coarbre. Le sous-arbre T_{0i}^x (resp. T_{1j}^x) dénote le sous-arbre enraciné en 0_i^x (resp. 1_j^x) et contenant l'ensemble de feuilles pour lesquelles 0_i^x (resp. 1_j^x) est le plus petit ancêtre commun avec x.*

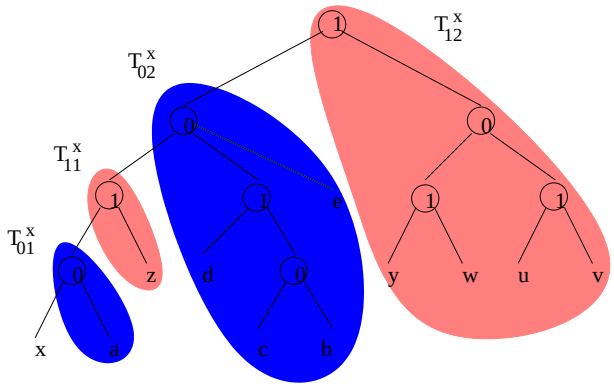

Fig. 6.6. Correspondance entre les sous-arbres et les couches lexicographiques. La couche $S_1(x) = \{a\}$ contient les feuilles de T_{01}^x, $S_2(x) = \{d, c, b, e\}$ celles de T_{02}^x, $\overline{S}_1(x) = \{z\}$ celles de T_{11}^x et $\overline{S}_2(x) = \{y, u, v, w\}$ celles de T_{11}^x.

Corollaire 6.20. *Soient σ un ordre LexBFS d'un cographe G débutant en v* [3] *et $\overline{\sigma} = LexBFS^-(\overline{G}, \sigma)$. Alors les feuilles des sous-arbres T_{0i}^v et T_{1j}^v sont respectivement les sommets de $S_i(v)$ et $\overline{S}_j(v)$.*

Le Corollaire 6.20 s'applique en particulier pour les ordres LexBFS σ et $\overline{\sigma} = LexBFS^-(\overline{G}, \sigma)$. De plus, un simple test d'adjacence entre les premiers sommets de $S_1(v)$ et $\overline{S}_1(v)$ permet de déterminer si le père de v dans le coarbre est un nœud parallèle ou série. L'ensemble du chemin allant de v à la racine du cographe peut donc être construit. Ainsi le Corollaire 6.20 permet une implémentation récursive de l'algorithme de Ehrenfeucht et al. [EGMS94] : il suffit de calculer récursivement un ordre LexBFS sur le graphe induit par chaque module de $\mathcal{M}(G, v)$ et un autre sur son complémentaire. Bien entendu, la complexité obtenue n'est pas linéaire. Pour cela, il faut "pousser" le Lemme 6.8 et mieux tirer partie de la règle *minus*. En fait, on peut montrer la propriété suivante.

Lemme 6.21. *Soit σ un ordre LexBFS d'un cographe G. Alors pour tout sommet v et tout i, la couche lexicographique $S_i(v)$ est un module de la partition $\mathcal{M}(G, v)$.*

[3] Notons que ni dans le Lemme 6.18, ni dans ce Corollaire, $\overline{\sigma}$ n'a besoin d'être obtenu par la règle *minus*. Il doit seulement débuter par le même sommet v.

Preuve. D'après le Lemme 6.18, la propriété est vraie pour le premier sommet de σ. Supposons qu'elle le soit pour tous les sommets précédents v. Soit $S(z)$ la couche lexicographique minimale contenant $S(v)$.

- $S(v) = S_j(z)$: Par hypothèse $S_j(z)$ est un module de $\mathcal{M}(G, z)$. Donc restreint à $S(v)$, σ reste un ordre LexBFS et le Lemme 6.18 s'applique. Donc dans le sous-graphe $G[S_j(z)]$, $S_i(v)$ est un module maximal ne contenant pas v. Puisque $S_j(z)$ est un module de $\mathcal{M}(G, z)$, $S_i(v) \in \mathcal{M}(G, v)$.
- $S(v) = S^A(z)$: Montrons que si $S_i(v)$ n'est pas un module, alors il existe un P_4. Supposons qu'il existe $y \notin S_i(v)$ tel que $ay \in E$ et $by \notin E$ pour $a, b \in S_i(v)$. Alors $y \notin S(v)$, sinon le Lemme 6.13 serait contredit. Puisque $S(v)$ est une couche lexicographique, $S(v)$ précède y dans σ. Soit $S(x)$ la couche lexicographique minimale contenant $S(v)$ et y. Alors nous devons avoir $S(v) \subseteq S^A(x)$. En examinant les adjacences entre les sommets x, v, a, b, y, on constate dans tous les cas l'existence d'un P_4 : contradiction. Il reste à montrer que $S_i(v) \in \mathcal{M}(G, v)$. Si ce n'est pas le cas, alors il existe $M \in \mathcal{M}(G, v)$ contenant $S_i(v)$ et un sommet $x \notin S_i(v)$. Notons que nécessairement x précède z dans σ. Donc x est soit adjacent soit non-adjacent à $S(z)$. Le premier cas est impossible car v sépare x de $S_i(v)$. Le second cas non plus car z sépare x de $S_i(v)$. Nous pouvons donc conclure que $S_i(v) \in \mathcal{M}(G, v)$.

□

Théorème 6.22. *Soient G un cographe et σ, $\overline{\sigma} = LexBFS^-(\overline{G}, \sigma)$ des ordres LexBFS. Pour tout sommet v, si $\overline{S}(z)$ est la plus petite couche lexicographique de $\overline{\sigma}$ contenant $\overline{S}(v)$, alors $\mathcal{M}(G[M(v, \overline{z})], v)$ est composée des ensembles $\{v\}$, $S_i(v)$ $(\forall i)$ et $\overline{S_j}(v)$ $(\forall j)$.*

Preuve. En utilisant le Lemme 6.21, il suffit de montrer que $S_i(v) \subset M(v, \overline{z})$ et que $\overline{S}_i(v) \subset M(v, \overline{z})$. Si $\overline{S}(v) = \overline{S}_j(z)$, alors $\overline{S}(v)$ est un module maximal ne contenant pas z et donc $\overline{S}_i(v) \subset M(v, \overline{z})$. Si $\overline{S}(v) = \overline{S}^A(z)$, puisque v sépare z de $\overline{S}_i(v)$, nous avons aussi $\overline{S}_i(v) \subset M(v, \overline{z})$. Considérons maintenant la couche lexicographique $S_i(v)$. Notons que la règle *minus* implique que z précède v. Si $v \in S(z)$, alors les mêmes arguments que ci-dessus s'appliquent et la propriété est vérifiée. Supposons donc que $v \notin S(z)$. Soit $S(y)$ la couche lexicographique minimale contenant z et v, alors $S(v) = S_j(y)$. Or d'après le Lemme 6.21, $S_j(y)$ est un module de $\mathcal{M}(G, y)$. Ce module contient v mais pas z. On en conclut que le module $S_i(v) \subset M(v, \overline{z})$. □

L'énoncé du Théorème 6.22 se traduit en termes de sous-arbres du coarbre.

Corollaire 6.23. *Soient G un cographe G et σ, $\overline{\sigma} = LexBFS^-(\overline{G}, \sigma)$ des ordres LexBFS. Pour tout sommet v, si $\overline{S}(z)$ est la plus petite couche lexicographique de $\overline{\sigma}$ contenant $\overline{S}(v)$, alors*

- *pour tout i, $S_i(v)$ est l'ensemble des feuilles de $T^v_{0i} \subset T_{lca(z,v)}$;*
- *pour tout i, $\overline{S}_i(v)$ est l'ensemble des feuilles de $T^v_{1i} \subset T_{lca(z,v)}$*

avec $T_{lca(v,z)}$ le sous-arbre enraciné en $lca(z,v)$ du cographe de G.

Les Lemmes 6.18 et 6.21 avec le Théorème 6.22 permettent de trouver récursivement les partitions modulaires recherchées dans l'algorithme de Ehrenfeucht et al. On déduit que toute l'information nécessaire à la construction du coarbre est présente dans les ordres σ et $\overline{\sigma} = \text{LexBFS}^-(\overline{G}, \sigma)$. De plus, il est possible en temps linéaire, d'extraire cette information. Pour cela, il suffit de construire pour chaque sommet x deux listes ordonnées $L(x)$ et $\overline{L(x)}$ contenant respectivement les premiers sommets des couches lexicographiques $S_i(x)$ et $\overline{S_j}(x)$. La construction de ces listes est linéaire. Pour plus de détails sur cette implémentation, se référer à [BCHP03] ou [Bre04].

6.4 Notes bibliographiques

Une vaste littérature existe sur les cographes. Le premier algorithme linéaire de reconnaissance de cette famille [CPS85] a marqué une étape importante pour l'algorithmique de la décomposition modulaire. De nombreux articles (même très récents) s'en inspirent encore, citons par exemple [MS89, SS04, CP04]. Vu les troublantes propriétés de LexBFS vis à vis de la décomposition modulaire, je ne serais vraiment pas surpris de l'existence d'un algorithme de décomposition modulaire simple basée sur l'analyse de plusieurs parcours LexBFS.

Concernant l'algorithme 16, une anecdocte mérite d'être racontée. L'algorithme original, présenté dans [BCHP03] et [Bre04] utilise en fait trois parcours LexBFS. Ce n'est qu'en relisant la preuve pour la rédaction de la version complète de l'article que nous nous sommes rendu compte que deux parcours suffisaient. Ce qui finalement est troublant dans cet algorihme, c'est sa non-symétrie en G et $\overline{G}$ (un parcours LexBFS quelconque sur G est suffisant). Nous n'avions pas pensé un seul instant qu'une telle assymétrie était possible.

J'espère avoir illustré dans ce chapitre la "puissance" de l'algorithme LexBFS. Les innombrables développements autour de cet algorithme ces dix dernières années, mériteraient qu'une synthèse bibliographique soit faite. D. Corneil a entamé ce travail long et important dans [Cor04].

Jus d'orange à la menthe et au gingembre

Ingrédients : 3 litre de jus d'orange, un bouquet de menthe, une racine de gingembre frais

Ciseler la memthe, émincer le gingembre et mettre le tout dans le jus do'range. Mettre au frais plusieurs heures avant de servir.

7

Représentation de graphes dynamiques

Ce chapitre considère le problème de la représentation de graphes dynamiques permettant de répondre efficacement à des requêtes d'adjacences. Un graphe est *dynamique* [1] s'il est soumis à des modifications d'arêtes ou de sommets. Les quatres opérations classiques sont :

- ajout et suppression d'un sommet (avec son voisinage);
- ajout et suppression d'une arête.

L'objectif est que la représentation soit la plus compacte possible. En particulier, plus petite que les représentations classiques en listes d'adjacence par exemple. Il n'est donc pas possible de résoudre ce problème pour tous les graphes. Cependant la décomposition modulaire s'avère efficace pour certaines familles de graphes dynamiques. C'est le cas par exemple des cographes [SS04, CP06] et des graphes de permutation [CP05].

A plus long terme, le problème de la maintenance dynamique de la décomposition modulaire nous intéresse. Idéalement, nous aimerions que le coût d'une modification soit linéaire en la taille de la modification (ou du moins en la taille de la mise à jour). Nous montrerons qu'une modification peut impliquer $O(n)$ modifications. Mais le problème reste toutefois ouvert quant à la décomposition modulaire dynamique d'un graphe quelconque en temps $O(n)$ par opérations.

Comme application, les algorithmes présentés permettent de résoudre le problème de *reconnaissance et représentation dynamique* d'une famille de graphe $\mathcal{F}$. Ce problème consiste à maintenir une représentation, caractéristique de la famille $\mathcal{F}$, d'un graphe tant que les modifications apportées ne remettent pas en cause l'appartenance à cette famille. Lorsque le graphe modifié n'appartient plus à la famille, on peut éventuellement rechercher un certificat. Relativement peu de résultats existaient pour ce problème à part

[1] L'objectif de ce chapitre n'est pas de faire une synthèse des algorithmes pour les graphes dynamiques. Pour cela, d'excellents articles, auxquels on peut se référer, existent, par exemple [EGI97, FK00].

pour les graphes d'intervalles propres [HSS02], les graphes triangulés [Iba99] et les cographes [SS04].

Notation 7.1 *Nous noterons $G' = G + x$ le graphe obtenu par l'ajout au graphe $G = (V, E)$ d'un sommet x et son voisiange $N(x) \subseteq V$:*

$$G' = (V \cup \{x\}, E \cup \{xy : y \in N(x)\})$$

Le graphe $G' = G - x$ résulte de la suppression d'un sommet $x \in V$ et des ses arêtes incidentes :

$$G' = (V \setminus \{x\}, E \setminus \{xy : y \in N(x)\})$$

Finalement $G' = G + xy$ et $G' = G - xy$ désignent respectivement les graphes $G' = (V, E \cup \{xy\})$ et $G' = (V, E \setminus \{xy\})$.

7.1 Le cas général

7.1.1 Modification d'arête

La propriété suivante nous indique qu'il n'est pas possible d'espérer mieux qu'une complexité en linéaire en le nombre de sommets du graphe.

Lemme 7.2. *[Cre05] L'ajout ou la suppression d'une arête peut générer $O(n)$ modifications dans l'arbre de décomposition modulaire.*

Preuve. Notons que l'ajout d'une arête dans un graphe G a le même effet que la suppression de cette arête dans $\overline{G}$. Considérons donc le cographe G de la figure 7.1 au quel nous ajoutons l'arête xy. Remarquons que : i) l'arête xy casse tout module M de G tel que $M \perp \{xy\}$; ii) tout module M de G contient exactement un sommet parmi x et y (i.e. $M \perp \{x, y\}$). Donc aucun module de G n'est conservé dans $G' = G + xy$. Soit M' un module de G'. Puisque M' n'est pas un module de G, nécessairement, M' contient soit x soit y mais pas les deux. Supposons sans perte de généralité que $x \in M'$. Puisque $xy \in E'$, tout sommet $v \in M$ doit vérifier $vy \in E'$ et donc $v \in \{v_n, \dots v_{2n}\}$ (car $lca(v, y)$ doit être un nœud série). Or il existe $v' \in \{v_1, \dots v_n\}$ tel que $v'v \notin E'$ et $v'x \in E$. Nous venons de montrer que v' est un casseur pour M' et ne peut pas appartenir à M'. M' est donc restreint au sommet x. Finalement nous pouvons conclure que G' ne contient que des modules triviaux : G' est donc premier. □

Nous pouvons donc conclure que tout algorithme dynamique maintenant l'arbre de décomposition modulaire d'un graphe a une complexité en $\Omega(n)$ par opération. Cette remarque est valable y compris pour les modifications de sommets puisque toute modification d'arête peut être faite par la suppression et l'ajout d'un sommet.

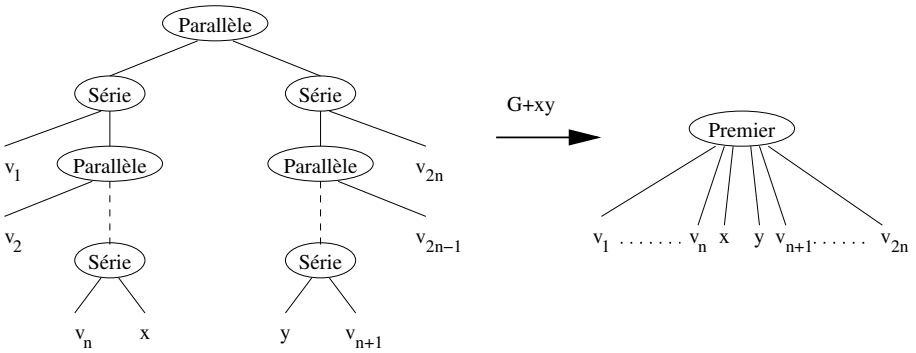

Fig. 7.1. Un exemple d'ajout d'arête entrainant $O(n)$ modification dans l'arbre de décomposition modulaire $MD(G)$.

7.1.2 Modification de sommet

Concernant l'ajout de sommet, l'algorithme quadratique de Muller et Spinrad [MS89] construit l'arbre de décomposition modulaire de manière incrémentale avec un coût $O(n)$ par sommet. Reste donc à considérer la suppression de sommet. Supprimer un sommet x dont le module fort minimal $m(x) \supset \{x\}$ est dégénéré, ne pose pas de problème. Il suffit de :

i. supprimer la feuille correspondant à x dans l'arbre de décomposition;
ii. si le nœud $n(x)$ correspondant à $m(x)$ ne possède plus qu'un seul fils f, supprimer $n(x)$;
iii. si $m(x)$ n'est pas la racine, raccrocher f au père $p(x)$ de $n(x)$, sinon f devient la racine de l'arbre de décomposition

Lemme 7.3. *Le coût de mise à jour de l'arbre de décomposition modulaire, lors de la suppression d'un sommet x dont le module minimal $m(x) \supset \{x\}$ est dégénéré, est $O(d)$.*

Le Lemme 7.3 a été présenté dans le cadre des cographes dynamiques [SS04]. Le cas difficile consiste donc à supprimer un sommet x dont la feuille correspondante est fils d'un nœud premier.

Problème 7.4. Existe-t-il un algorithme complètement dynamique de décomposition modulaire ayant un complexité $O(n)$ par opération ?

Remarquons toutefois qu'il est peu probable que l'algorithme de Muller et Spinrad [MS89] puisse s'étendre de manière à supporter les suppressions de sommets. En effet, pour obtenir la complexité $O(n)$ par ajout de sommet, ils ne maintiennent pas une reprénsetation complète du graphe. En particulier les graphes quotients des nœuds premiers ne sont pas maintenus.

7.2 Les cographes dynamiques

Est-il possible, si l'on se restreint à une famille de graphe particulière de faire mieux que $O(n)$ par opération ? La première classe de graphe à examiner est la famille des graphes complètement décomposable par la décomposition modulaire, à savoir les cographes (voir Chapitre 2). En effet, nous avons vu dans la section précédente que le point difficile était la gestion des nœuds premiers. Nous présentons dans cette section les résultats dûs à Shamir et Sharan [SS04] basés sur l'algorithme incrémental de reconnaissance des cographes de Corneil et al. [CPS85].

Nous allons dans cette section présenté les conditions dans lesquelles il est possible de modifier un cographe en restant dans la famille des cographes. Nous décrirons alors les mises à jour du coarbre impliqué par ces modifications et leur coût.

7.2.1 Modification de sommet

Puisque l'arbre de décomposition modulaire d'un cographe, le coarbre, ne contient que des nœuds dégénérés, le Lemme 7.3 s'applique. Le premier algorithme linéaire de reconnaissance des cographes [CPS85] est un algorithme incrémental. A chaque étape, l'ajout d'un sommet x coûte $O(d)$ où d est le degré du sommet x.

L'algorithme incrémental se base sur une étape préliminaire de marquage du coarbre en fonction du voisinage $N(x)$ du sommet x. Initialement tous les nœuds du coarbre sont non-marqués, sauf les feuilles correspondantes aux voisins de x qui sont marquées *adjacentes*. Un nœud adjacent marque son père (s'il existe) *mixte*. Si tous les fils d'un nœud sont *adjacents*, alors ce nœud devient aussi *adjacent*. La complexité de ce processus de marquage des nœuds du coarbre est $O(d)$, avec $d = |N(x)|$.

Le Lemme 7.5 caractérise les insertions possible d'un sommet x dans un cographe.

Lemme 7.5. *[CPS85] Soient $G = (V, E)$ un cographe et $x \notin V$ un sommet voisin de $N(x) \subseteq V$. $G + x$ est un cographe ssi*

1. *le coarbre $MD(G)$ ne contient pas de nœud mixte;*
2. *les nœuds mixtes, sauf le plus profond, sont les nœuds série d'un chemin de $MD(G)$ contenant la racine et tous leurs fils sauf un (celui du chemin) sont adjacents.*

Le nœud mixte q le plus profond pourrait être appelé le nœud d'insertion de x. Lorsque l'insertion de x est possible, alors toutes les modifications du coarbre $MD(G)$ ont lieu dans le sous-arbre enraciné en q. En effet, le module

fort Q correspondant au nœud d'insertion q est le plus petit module fort de G tel que $F \cup \{x\}$ est un module fort de $G' = G + x$.[2]

En fait les modifications consistent à ajouter un fils au nœud d'insertion q qui permet de distinguer les fils marqués adjacent des autres. Un exemple de modification est présenté Figure 7.2.

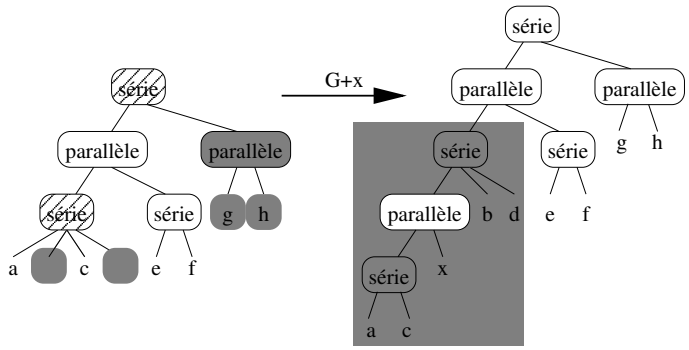

Fig. 7.2. Ajout du sommet x voisin de $\{b, e, f\}$. Dans $MD(G)$, les nœuds grisés sont marqués adjacents, les nœuds rayés sont mixtes. Si un des sommets c ou d avait été voisin de x, un P_4 serait apparu (e.g. a, b, x, d). De même, si l'un des sommets e ou f n'était pas voisins de x, G' ne serait pas un cographe. Notons que le nouveau nœud de $MD(G')$ (en blanc) permet de distinguer les fils de q adjacents à x des autres.

7.2.2 Modification d'arête

Notons que les problèmes de l'ajout et de la suppression d'une arête dans un cographe sont les mêmes. En effet, ajouter une arête dans un graphe G équivaut à la supprimer dans son complémentaire $\overline{G}$. Or le complémentaire $\overline{G}$ d'un cographe G est un cographe et son coarbre est obtenu à partir de celui de G en inversant les nœuds séries et parallèles. Les modifications sur la topologie du coarbre seront donc les mêmes dans les deux cas. Une illustration du lemme de caractérisation ci-dessous est donnée Figure 7.3.

Lemme 7.6. *[SS04] Soient x et y deux sommets non-adjacent d'un cographe G $G' = G+xy$ est un cographe ssi x est un fils de $m(x, y)$ et $M(y, \overline{x}) \subseteq N(x)$.*[3]

[2] Cette approche consistant à identifier le nœud d'insertion comme le plus petit module fort F de G tel que $F \cup \{x\}$ est un module fort de G' est utilisée pour les graphes de permutations dynamiques (voir Section 7.3). Son intérêt est de dissocier l'aspect combinatoire de l'aspect algorithmique de la mise à jour dynamique.

[3] Rappelons que $m(x, y)$ est le module fort minimal contenant x et y alors que $M(y, \overline{x})$ est le module fort maximal contenant y mais pas x. Voir Chapitre 2.

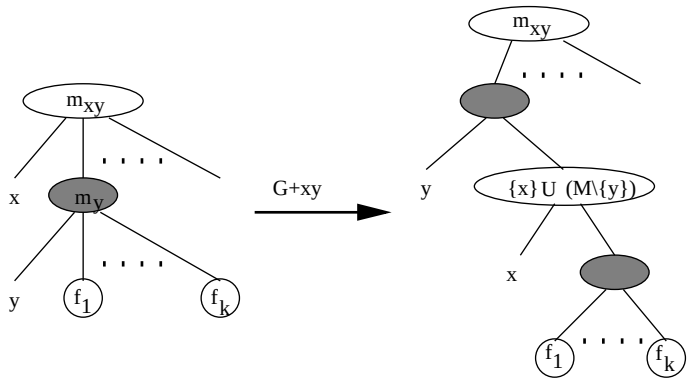

Fig. 7.3. Mise à jour du coarbre lors de l'insertion d'une arête dans un cographe.

On déduit du Lemme 7.6 que les modifications du coarbre sont locales. Il est donc possible de les mener en $O(1)$.

Théorème 7.7. *[SS04, CPS85] Il existe un algorithme complètement dynamique de maintient de l'arbre de décomposition modulaire pour la famille des cographes dont la complexité est $O(d)$ par modification impliquant d arêtes.*

Notons que dans [CP04, CP06] nous étendons l'algorithme de reconnaissance dynamique des cographes de [SS04] aux cographes orientés. De plus, notre algorithme permet d'extraire, dans la même complexité, un sous graphe exclu lorsque le graphe modifié n'est plus un cographe. Le cas non-orienté est relativement simple puisque la seule configuration interdite est le P_4 (voir Chapitre 2) et ce problème était déjà résolu pour l'insertion de sommet dans [CPS85]. Par contre, le cas orienté nécessite plus de travail. Les sous-graphes exclus sont au nombre de 7 et peuvent contenir 3 ou 4 sommets. Notons que le processus de marquage utilisé dans [CP04, CP06] diffère de celui décrit dans [CPS85] (voir la note 2 page précédente).

7.3 Les graphes de permutation dynamiques

Il n'est pas très étonnant d'obtenir des mises à jour efficace pour les cographes puisque leur arbre de décomposition ne contient pas de nœud premier. Il serait plus intéressant de trouver une famille $\mathcal{F}$ de graphes qui répond au problème 7.4 et n'impose aucune restriction sur la topologie de l'arbre (i.e. n'ayant aucune restriction sur le nombre de nœuds premiers, ni leurs positions dans l'arbre, ni leur degré). Nous montrons dans cette section que les *graphes de permutation* vérifient ces propriétés [CP05]. Pour un rappel de la définition de ces graphes et des exemples, voir Chapitre 2.

La famille des graphes de permutations généralise celle des cographes. Pour s'en rendre compte, observons qu'un réaliseur peut facilement être construit

à partir d'une permutation factorisante π_1 d'un cograph G. Pour obtenir π_2, il suffit de renverser chaque facteur correspondant à un module fort sauf celui de la racine si G n'est pas connexe.

Puisque l'arbre de décomposition modulaire d'un graphe de permutation G peut contenir des nœuds premiers, nous devrons maintenir en plus de $MD(G)$, un codage des graphes représentatifs de chacun des nœuds premiers, nous utiliserons leurs réaliseurs. Avant de présenter les algorithmes de maintenance, intéressons nous à la structure de $MD(G)$ et son codage.

7.3.1 Décomposition modulaire des graphes de permutation.

La famille des graphes de permutation est une famille héréditaire. Donc tout graphe représentatif associé à un nœud premier de l'arbre de décomposition modulaire est aussi un graphe de permutation. Il est facile de voir que la réciproque est vraie. En effet, soient $G = (V, E)$ et $H = (V', E')$ deux graphes de permutation, l'opération de substitution d'un sommet $x \in V$ par H produit un graphe de permutation.

Lemme 7.8. *[Gol80] Un graphe est un graphe de permutation ssi les graphes représentatifs des nœuds premiers de l'arbre de décomposition modulaire sont des graphes de permutation.*

On peut aussi montrer, pour les graphes de permutation, une propriété similaire au Lemme 2.29 dont une des conséquences est que les graphes de comparabilité premiers n'admettent (au renversement près) qu'une orientation transitive. En effet, nous avons :

Lemme 7.9. *[Gol80] Un graphe de permutation premier admet un unique réaliseur (à retournement près [4]).*

Par ailleurs, on déduit du Lemme 7.8 si le réaliseur du graphe représentatif de chaque nœud premier est stocké avec l'arbre de décomposition modulaire, alors d'après le Lemme 2.19, on obtient une représentation de taille $O(n)$ d'un graphe de permutation.

Dans la suite, nous maintiendrons pour un graphe de permutation G, son arbre de décomposition modulaire $MD(G)$ couplé avec le réaliseur de chacun de ses nœuds premiers. De plus, pour des raisons d'efficacité, il sera utile d'avoir aussi un réaliseur de l'ensemble du graphe.

Un ensemble de sommets $I \subseteq V$ est un *intervalle commun* à deux permutations π_1 et π_2 de V si I est un intervalle (ou facteur) de π_1 et π_2. Autrement dit les éléments de I sont consécutifs dans les deux permutations. La propriété suivante sera très utile pour reconstituer les modules d'un graphe de permutation modifié.

[4] Soit π une permutation, $\overline{\pi}$ désigne la permutation. Si (π_1, π_2) est un réaliseur d'un graphe de permutation, alors (π_2, π_1), $(\overline{\pi}_1, \overline{\pi}_2)$ et $(\overline{\pi}_2, \overline{\pi}_1)$ le sont aussi. Nous dirons que ces quatres réaliseurs sont les mêmes.

Lemme 7.10. *[dM03] Soit (π_1, π_2) le réaliseur d'un graphe de permutation G. Alors tout module fort M de G est un intervalle commun à π_1 et π_2.*

7.3.2 Modification d'arête

Notons que le graphe G de la Figure 7.1 est un cographe, donc un graphe de permutation, et que $G' = G + xy$ est un aussi un graphe de permutation. Le Lemme 7.2 s'applique aux graphes de permutation. Puisque, comme nous allons le voir l'insertion et la suppression de sommet coûtent $O(n)$, pour supprimer ou ajouter une arète xy, il suffira de supprimer le sommet x, puis le rajouter avec son voisinage mis à jour.

7.3.3 Suppression de sommet

Puisque les graphes de permutation sont héréditaires, la suppression d'un sommet quelconque maintient le graphe dans la famille des graphes de permutation. De plus, pour tout module M d'un graphe G, $M' = M \setminus \{x\}$ est un module de $G' = G - x$. Les modules de G sont donc conservés, éventuellement tronqués du sommet x. Par contre, G' peut contenir des nouveaux modules : les ensembles de sommets pour lesquels x étaient le seul casseur.

Lemme 7.11. *[CP05] Soit (π_1, π_2) le réaliseur d'un graphe de permutation premier $G = (V, E)$. Soit M' un module fort de $G' = G - x$ pour $x \in V$ tel que $M' \cup \{x\}$ n'est pas un module de G, alors $M' \cup \{x\}$ est un intervalle de π_1 ou un intervalle de π_2.*

On en déduit qu'il y a au plus $2n$ nouveaux modules dans $G' = G - x$ et qu'ils forment deux familles totalement ordonnées pour l'inclusion. En utilisant un algorithme de recherche d'intervalles communs [UY00, BXHP05b] à deux permutations (voir Chapitre 5), il est possible de les identifier en temps $O(n)$. Il est toutefois possible d'utiliser un algorithme plus simple compte tenu de la structure imbriquée des modules de G'.

7.3.4 Ajout de sommet

La mise à jour de l'arbre de décomposition modulaire $MD(G)$ et de l'ensemble des réaliseurs associés aux nœuds premiers après l'ajout d'un sommet x s'effectue en trois étapes :

1. Un processus de marquage (alternatif à celui développé dans [CPS85]) permet d'identifier le plus petit module fort Q du graphe G tel que $Q \cup \{x\}$ est un module fort de $G' = G + x$.
2. L'arbre de décomposition $MD(G')$ est alors calculé étant donné $MD(G)$ et l'ensemble des réaliseurs. Notons que cette étape est valide quelle que soit la nature du graphe G'. Par contre, elle ne permet pas de mettre à jour les réaliseurs de G' si ce graphe est un graphe de permutation.

3. Enfin, un test basé sur une caractérisation des graphes de permutation permet de vérifier si G' est un graphe de permutation. Si c'est le cas, il est alors possible de mettre à jour les réaliseurs des nœuds premiers de G'.

La complexité de chacune de ces trois étapes n'excède pas $O(n)$. Nous en décrivons ci-dessous les grandes lignes mais ne présenterons pas les preuves, ni ne décrirons les algorithmes dans le détail. Pour plus de précision, se référer à [CP05]. Pour faciliter l'intuition sur cette partie relativement technique, commençons la discussion par l'insertion d'un sommet dans un graphe premier. La suite de la section, présente les trois étapes évoquées ci-dessus.

Insertion d'un sommet dans un graphe de permutation premier

Supposons que nous souhaitions ajouter un sommet non-universel et non-isolé à un graphe de permutation premier. Rapellons que si G est un graphe premier, alors il ne possède (à renversement près) qu'un seul réaliseur (π_1, π_2). De plus l'ajout de x ne permet de créer qu'un seul module M de taille deux contenant x (sinon G ne serait pas premier). Donc soit $G' = G+x$ est premier, soit x possède un jumeaux y dans G'. On en déduit que si G' est un graphe de permutation, alors il existe au plus deux positions d'insertion dans (π_1, π_2) pour le sommet x. Si x possède un jumeaux y, alors x et y seront consécutifs dans les deux permutations du réaliseur de G'.

Il est possible de tester si x possède un jumeau ou si x peut être inséré dans le réaliseur (π_1, π_2) avec un parcours simultané de π_1 et π_2. La complexité de ce test est $O(n)$.

Processus de marquage et nœud d'insertion

Chaque nœud de $MD(G)$ est marqué en fonction de ses relations de voisinage avec le nouveau sommet x à insérer.

Notation 7.12 *Pour simplifier, les nœuds de $MD(G)$ seront notés par des minuscules $p, q, r \ldots$ et les modules forts associés par les majuscules correspondantes $P, Q, R \ldots$*

Soit p un nœud de $MD(G)$. Alors p est *adjacent* à x si $P \subseteq N(x)$, *non-adjacent* à x si $P \subseteq \overline{N}(x)$, et *mixte* sinon[5]. Dans les deux premiers cas, p est *uniforme* par rapport à x. Marquer l'ensemble des nœuds de $MD(G)$ par leurs types (adjacents, non-adjacents ou mixte), se fait en $O(n)$ par un simple parcours de $MD(G)$ depuis les feuilles jusque la racine.

Afin de caractériser le nœud d'insertion, nous avons besoin de la définition technique suivante.

[5] Remarquons le parallèle avec les marques utilisées pour les cographes [CPS85]. Celui-ci est plus complet. La marque "non-adjacent" n'était pas utilisée et la marque "mixte" était plus limitée. En contre-partie, nous n'aurons pas la même complexité.

Définition 7.13. *Un nœud p de $MD(G)$ est* propre *ssi soit p est uniforme par rapport à x, soit p est mixte mais ne possède qu'un seul fils mixte f et $F \cup \{x\}$ est un module de $G'[P \cup \{x\}]$. Dans le cas contraire, p est dit* impropre.

Lemme 7.14. *[CP05] Si q est le plus petit ancêtre commun de tous les nœuds impropres de $MD(G)$, alors $Q' = Q \cup \{x\}$ est un module fort de $G' = G + x$.*

Notons que q existe toujours, c'est éventuellement la racine de $MD(G)$. Enfin, il est possible d'obtenir $MD(G')$ en remplaçant dans $MD(G)$ le sous-arbre enraciné en q par $MD(G'[Q'])$. De plus, nous pouvons prouver que q est le plus petit nœud vérifiant ces propriétés. Ce nœud q, plus petit ancêtre commun à tous les nœuds impropres, sera donc le nœud d'insertion.

Calcul de $MD(G')$

Remarquons que le nœud d'insertion q est lui-même impropre. De ce fait, différentes situations peuvent se distinguer. Nous dirons que q est *préservé* si

1. q est un nœud dégénéré (i.e. série ou parallèle) dont les fils sont uniformes mais de différents types;
2. q est premier, tous ses fils sont uniformes dont un est jumeaux de x dans le graphe représentatif G_q;

Si q n'est pas préservé, il est dit *cassé*. Dans ce cas, q peut être dégénéré avec un fils mixte ou premier sans fils jumeaux de x dans G_q.

Dans le cas où le nœud d'insertion q est préservé, alors $MD(G')$ est facile à obtenir. Si q est dégénéré, la méthode de mise à jour utilisée pour les cographes [CPS85] s'applique (voir Figure 7.2 ou Figure 7.4). Si q est premier, alors il existe dans G_q un jumeau de x. Il suffit donc de remplacer le fils f de q correspondant à ce jumeau par un nœud série ou parallèle (selon sa relation avec x) et ayant f et x comme fils.

Il est plus difficile de mettre à jour l'arbre de décomposition modulaire lorsque q est cassé. Si q est série (resp. parallèle), alors les fils de q marqués voisins (resp. non-voisins) seront des fils de q' et les autres seront regroupés sous un fils q'_s de q'. Pour décrire l'arbre $MD(G')$, nous devons donc décrire l'ensemble des sommets Q_s correspondant à ce fils q'_s de $MD(G')$

- si q est premier, alors $Q_s = Q$;
- si q est série, alors Q_s est l'union des modules fils de q marqués mixtes ou non-voisins;
- si q est parallèle, alors Q_s est l'union des modules fils de q marqués mixtes ou voisins;

Le processus de marquage a permis d'identifier des modules forts uniformes (voisins ou non-voisins). Mais il est possible que certains de ces modules soient coupés ou fusionnés pour former des modules forts de G'. En fait, il faut s'intéresser aux modules uniformes maximaux contenus dans Q_s. Ils forment une partition de Q_s.

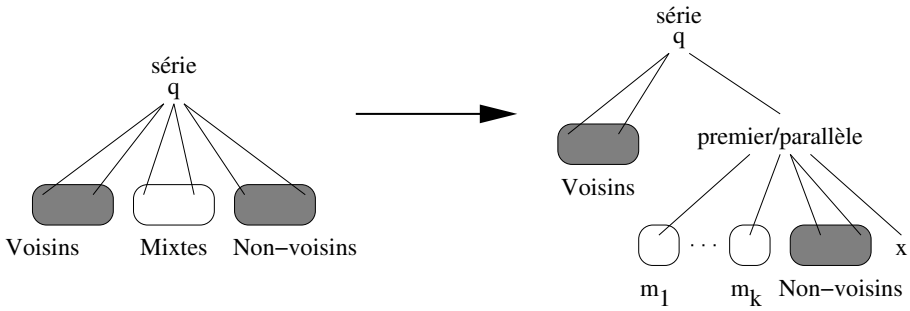

Fig. 7.4. Modification de $MD(G)$ lors de l'ajout d'un sommet x. Seul le sous-arbre enraciné en q, le nœud d'insertion, est représenté. Ce nœud est série. Si q est préservé, alors 1) q ne possède pas de fils mixte; 2) le nœud interne de $MD(G')$ est parallèle et ses fils sont x et les fils de q marqués non-voisins. Par contre si q est cassé, le nœud interne devient premier et les fils $m_1, \ldots, m_k$ sont définis par les modules uniformes maximaux de $G[Q]$ (voir Théorème 7.15).

Théorème 7.15. *[CP05] Si q est préservé, alors $G'[Q_s \cup \{x\}]$ est connexe et co-connexe et ses modules forts maximaux sont exactement $\{x\}$, et les modules uniformes maximaux de $G[Q_s]$.*

$MD(G')$ est donc décrit par le Théorème 7.15. Puisque la structure de $MD(G)$ en dehors du sous-arbre enraciné en q est conservée. Aussi pour chaque modules uniforme maximal M_i, $MD(G[M_i])$ peut être extrait de $MD(G)$ et le Théorème 7.15 spéficie les modifications autour du nœud q.

Notons que ces modifications peuvent être effectuées en temps $O(n)$ (voir [CP05]).

Caractérisation dynamique des graphes de permutation

Il reste à vérifier que le graphe $G' = G + x$ est un graphe de permutation. Remarquons que d'après les discussions précédentes sur la mise à jour de $MD(G)$, G' est un graphe permutation ssi $G'[Q \cup \{x\}]$ est un graphe de permutation. De plus, nous pouvons mettre en évidence une propriété similaire au Lemme 7.11. A savoir, au plus 2 branches de $MD(G)$ enracinées en q contiennent des nœuds mixtes.

Lemme 7.16. *[CP05] Si G' est un graphe de permutation, alors le nœud d'insertion q possède au plus deux fils mixtes, et tout nœud $p \neq q$ possède au plus un fils mixte.*

Preuve. L'idée de la preuve est simple. Suppposons que G' soit un graphe de permutation, alors la suppression de x dans G' provoque éventuellement la création dans $MD(G)$ de deux ensembles de modules totalement ordonnés par l'inclusion tel que le décrit le Lemme 7.11. Ces nœuds de $MD(G)$ sont précisemment les nœuds mixtes. □

Le Lemme 7.16 n'est encore pas suffisant pour caractériser les insertions possibles d'un sommet dans un graphe de permutation. La caractérisation ci-dessous est relativement technique. Pourtant elle peut être testé en temps $O(n)$. Nous suggérons à nouveau de se référer à [CP05] pour plus de détails.

Notons $F_{mixte}(p)$ l'union des modules forts mixtes fils d'un nœud p de $MD(G)$. Si nous supprimons les arêtes entre x, le sommet à ajouter, et les modules mixtes $F_{mixte}(p)$, alors les modules forts maximaux de $G[P]$, notés $\mathcal{M}_{fort}(p)$ sont aussi des modules de $G'[P \cup \{x\}] - xF_{mixte}(p)$ [6].

Pour chaque nœud p descendant de q dans $MD(G)$, nous définissons le sous-graphe suivant :

$$H(p) = (G'[P \cup \{x\}] - xF_{mixte}(p))_{/(\mathcal{M}_{fort}(p) \cup \{\{x\}\})}$$

L'idée est en fait de partitionner les arêtes incidentes à x en fonction des modules forts de G uniformes par rapport à x. Ainsi, il sera possible de tester l'insertion de manière récursive sur chaque nœud mixte descandant du nœud d'insertion. En fait, grâce au graphe $H(p)$, chaque nœud mixte p descendant du nœud d'insertion q est transformé en un nœud préservé.

Théorème 7.17. *[CP05] Soit G un graphe de permutation et x un sommet à ajouter. Alors $G' = G + x$ est un graphe de permutation ssi soit le nœud d'insertion q est préservé; soit q est cassé et les conditions suivantes sont satisfaites.*

1. *le nœud d'insertion q vérifie l'une de ces deux conditions :*
   a) *q possède exactement deux fils mixtes f_1 et f_2 and $H(q)$ est un graphe de permutation admettant un réaliseur (π_1, π_2) tel que x et f_1 sont consécutifs dans π_1 et x et f_2 le sont dans π_2;*
   b) *q possède un unique fils mixtes f_1 et $H(q)$ est un graphe de permutation admettant un réaliseur (π_1, π_2) tel que x et f_1 sont consécutifs dans π_1;*
   c) *q n'a pas de fils mixte et $H(q)$ est un graphe de permutation.*
2. *tout nœud p descendant de q vérifie l'une de ces deux conditions :*
   a) *p possède un unique fils mixte f_1 et $H(q)$ est un graphe de permutation admettant un réaliseur (π_1, π_2) tel que x et f_1 sont consécutifs dans π_1 et x est le premier élément de π_2;*
   b) *p n'a pas de fils mixte et $H(q)$ est un graphe de permutation admettant un réaliseur (π_1, π_2) tel que x est le premier élément de π_2;*

Pour prouver le théorème précédent, nous avons besoin des lemmes suivants. Ils permettent notamment de comprendre comment un réaliseur du graphe $G' = G + x$ peut être reconstruit en remontant dans l'arbre de décomposition et en substituant les réaliseurs associés aux différents modules dans ceux de leur père.

[6] Si S est un ensemble de sommets d'un graphe $G = (V, E)$ et $y \in V \setminus S$, alors $G - yS$ désigne le sous-graphe dont les arêtes sont $E \setminus \{yz \mid z \in S\}$.

Lemme 7.18. *Si $G' = G + x$ est un graphe de permutation et si le nœud d'insertion q est cassé et possède deux fils mixtes f_1 et f_2, alors $H(q)$ est un graphe de permutation admettant un réaliseur (π_1, π_2) tel que 1) x et f_1 sont consécutifs dans π_1; et 2) x et f_2 sont consécutifs dans π_2.*

Lemme 7.19. *Si $G' = G + x$ est un graphe de permutation, si le nœud d'insertion q est cassé et si le nœud $p \neq q$, descendant de q, possède un unique fils mixte f_1, alors $H(p)$ est un graphe de permutation admettant un réaliseur (π_1, π_2) tel que 1) x et f_1 sont consécutifs dans π_1.; et 2) x est le premier élément de π_2.*

Rappelons les grandes lignes de l'algorithme d'insertion d'un sommet x dans un graphe de permutation G : 1) un processus de marquage permet en une passe sur $MD(G)$ de trouver le nœud d'insertion q; 2) $MD(G' = G + x)$ est ensuite calculé en fonction de la nature de q (préservé, cassé); 3) un test permet de vérifier si G' est un graphe de permutation en n'examinant que les réaliseurs associés aux nœuds descendants de q; 4) les nouveaux réaliseurs, y compris celui de G sont calculés en insérant x dans les réaliseurs des nœuds de $MD(G)$ selon un parcours des feuilles vers la racine.

La complexité de chacune de ces étapes est $O(n)$. On en déduit le résultat suivant.

Théorème 7.20. *[CP05] Il existe un algorithme complètement dynamique de maintient de l'arbre de décomposition modulaire pour la famille des graphes de permutation dont la complexité est $O(n)$ par opération.*

7.4 Notes bibliographiques

Quelques résultats similaires sont connus pour d'autres familles de graphes : graphes d'intervalles propres [HSS02], graphe triangulés [Iba99]. Toutefois pour cette dernière famille, les complexités annoncées ne sont pas fantastiques. Il est possible que des améliorations soient possibles, du moins pour certaines sous-familles telles que la famille des graphes d'intervalles. Concernant les graphes d'intervalles propres, on pouvait s'attendre à des résultats intéressants. En effet, comme les cographes, un graphe d'intervalle propre possède une représentation compacte canonique (voir [HSS02]).

Je suis persuadé que toute famille de graphes ayant une représentation canonique compacte doit supporter des algorithmes efficaces pour leur reconnaissance dynamique. Par exemple, une famille intéressante à considérer est celle des graphes distance héréditaire qui jouent pour la décomposition en coupe, le même rôle que les cographes pour la décomposition modulaire.

Tarte à la courgette et à la menthe

Ingrédients : 3 courgettes, 1 bouquet de menthe, 3 œufs, huile d'olive et coulis de tomates (épais avec morceaux).

Faire revenir les courgettes (rincées et non épluchées) découpées en gros cubes dans l'huile d'olive. Les réserver lorsqu'elles sont dorées mais toujours fermes. Dans un saladier, mélanger le coulis (pour le faire soit même, voir par exemple Chapitre 5), la menthe ciselée et les œufs.

Couvrir le fond du plat avec une feuille alu. Disposer les courgettes et verser la préparation et recouvrir avec la pâte. Mettre au four chaud pendant 25 minutes (250). Démouler la tarte et la sopoudrer avec l'origan.

8

Tri par renversements et décomposition modulaire

Ce chapitre aborde un problème issu de la bioinformatique et en particulier de la génomique comparative pour lequel la décomposition modulaire joue un rôle primordial. Les objets combinatoires manipulés dans ce problème ne sont plus les graphes mais des permutations.

En 1992, Sankoff [San92] définit la *distance de renversement* entre deux permutations. *Renverser* un intervalle I dans une permutation π consiste à inverser la relation d'ordre entre les éléments de I. La distance de renversement entre deux permutations π et τ est le minimum de renversements nécessaires pour transformer π en τ. En supposant que les permutations modèlisent des génomes, la distance de renversement est alors interprétée comme une distance d'édition (ou d'évolution) entre ces génomes. Ce modèle (trop[1] ?) simple est justifié puisque des opérations de renversements de parties de génomes sont observées dans les processus d'évolution. Malheureusement, le problème du calcul d'une séquence de renversements de longueur minimum entre deux permutations a été prouvé NP-difficile [Cap97]. Mais, du point vue biologique, utiliser des permutations signées comme modèle est plus réaliste. Et le problème du tri par renversements d'une permutation signée est, lui, bien maîtrisé [BMS05]. Une théorie importante [HP99] et des algortihmes de tri efficaces [TS04] existent.

Une critique possible à l'encontre de la distance par renversements est que les opérations impliquées dans un scénario transformant une permutation en une autre ne respectent pas la structure commune à ces deux permutations. En effet, si ces deux permutations possèdent des fragments de génomes communs, alors il est raisonnable de penser qu'ils aient toujours été présents au cours du processus d'évolution. Ces fragments de génomes communs peuvent être modélisés par la notion d'*intervalle commun*. Dans ce contexte, Figeac et Varré [FV04], ont introduit le problème du *tri parfait* par renversements de permutations signées. Une séquence de renversements est parfaite si aucun renversement ne casse un intervalle commun. Malheureusement Figeac et

[1] En particulier, ce modèle n'autorise pas la duplication ou la répétition de gènes.

Varré [FV04] annoncent que le problème du tri parfait est NP-difficile [FV04]. Néanmois, en étudiant la structure des intervalles communs à deux permutations, il est possible de montrer que dans la plupart des cas, ce n'est pas un problème difficile.

Ce chapitre considère dans un premier temps les intervalles communs à deux permutations et leurs structures. Nous étudierons ensuite les scénarios parfaits et terminerons ce chapitre par quelques résultats et questions ouvertes sur les permutations qui ont la propriété d'admettre un scénario parcimonieux (de longueur minimal) et parfait.

8.1 Intervalles communs

Cette section ne considère que des permutations non-signées sur un ensemble $\{1 \dots n\}$. La permutation identité sur $\{1 \dots n\}$ est notée $\mathbb{I}_n$. Si π est une permutation de $\{1, \dots n\}$, alors $\overline{\pi}$ désigne la permutation *renversée* : i précède j dans $\overline{\pi}$ ssi j précède i dans π. Rappelons (voir Chapitre 2, Section 2.2) qu'un *intervalle* d'une permutation π d'un ensemble V est un ensemble $I \subseteq \{1 \dots n\}$ d'éléments consécutifs dans π. L'objectif de cette section est d'étudier les structures communes à plusieurs permutations.

Définition 8.1. *Soient π et σ deux permutations sur $\{1 \dots n\}$. Un* intervalle commun *à π et σ est un ensemble $I \subseteq \{1 \dots n\}$ tel que I est à la fois un intervalle de π et un intervalle de σ.*

Sans perte de généralité et pour simplifier les notations, nous ne considérons pas une paire de permutations quelconques, mais une permutation quelconque π et la permutation identité $\mathbb{I}_n$ (au besoin, il suffit d'effectuer une composition de permutations pour retrouver le cas général). Dans la suite, nous dirons abusivement que I est un *intervalle commun de* π signifiant que c'est un intervalle commun à π et $\mathbb{I}_n$. Il est clair que par définition, l'ensemble $\{1 \dots n\}$ est un intervalle commun ainsi que tout ensemble singleton $\{i\}$ (pour $i \in \{1 \dots n\}$).

Fig. 8.1. Une permutation $\pi = 3\ 2\ 5\ 1\ 4\ 9\ 6\ 7\ 8$ et ses intervalles communs non-triviaux (avec $\mathbb{I}_n$) : $\{2,3\}$, $\{1,2,3,4,5\}$, $\{6,7\}$, $\{7,8\}$, $\{6,7,8\}$ et $\{6,7,8,9\}$.

Lemme 8.2. *Si I et J sont deux intervalles communs d'une permutation π, alors $I \cup J$, $I \cap J$, $I \setminus J$ et $J \setminus I$ sont des intervalles communs de π.*

Autrement dit, d'après la Définition 2.1, la famille des intervalles communs d'une permutation, notée $\mathcal{I}$, est une famille faiblement partitive. En définissant une opération de fermeture, il est possible de se restreindre à un sous-ensemble des intervalles communs.

Définition 8.3. *Soit $\mathcal{F}$ un ensemble d'intervalles commun d'une permutation π. La fermeture $\mathcal{F}^*$ de $\mathcal{F}$ est le plus petit ensemble d'intervalles communs de π contenant $\mathcal{F}$, les intervalles triviaux et tel que pour tout $I, J \in \mathcal{F}^*$ tels que $I \perp J$, $\{I \cup J, I \cap J, I \setminus J, J \setminus I\} \subset \mathcal{F}^*$.*

Par définition donc $\mathcal{F}^*$ est aussi une famille faiblement partitive. Nous pouvons donc appliquer tous les résultats connus et en particulier définir, pour une famille $\mathcal{F}$ d'intervalles communs d'une permutation, l'*arbre des intervalles forts*, noté $T_{\mathcal{F}^*}$. Un intervalle commun est un *fort* s'il ne chevauche aucun autre intervalle commun de $\mathcal{F}^*$. Dans la suite, un intervalle fort I est *fils* d'intervalle fort J si I est fils de J dans l'arbre $T_{\mathcal{F}^*}$.

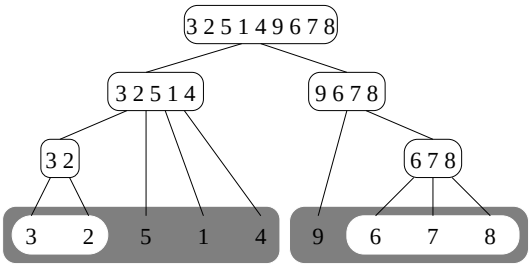

Fig. 8.2. L'arbre $T_{\mathcal{I}}$ des intervalles forts de π. Les intervalles communs $\{6, 7\}$ et $\{7, 8\}$ ne sont pas forts.

Une partition $\mathcal{P} = \{I_1, \dots I_k\}$ de $\{1, \dots n\}$ est une *partition intervallaire* de la permutation π si pour tout $j \in \{1 \dots k\}$, I_j est une intervalle commun de π. Les partitions intervallaires ont les mêmes propriétés que les partitions modulaires des sommets d'un graphe. De même, il est possible de définir la notion de permutation quotiente. En effet si I et J sont deux intervalles communs d'une permutation π, alors tout élément de I précède tout élément de J dans π (ou l'inverse).

Définition 8.4. *Soient π une permutation de $\{1, \dots n\}$ et $\mathcal{P} = \{I_1, \dots I_k\}$ une partition modulaire. Alors $\pi_{/\mathcal{P}}$ est une permutation sur $\{1 \dots k\}$ telle que i précède j dans $\pi_{/\mathcal{P}}$ ssi I_i précède I_j dans π.*

Ainsi pour la permutation π de la Figure 8.1, si la partition intervallaire est $\mathcal{P} = \{\{1, 2, 3, 4, 5\}, \{9\}, \{6, 7\}, \{8\}\}$, la permutation quotiente est $\pi_{/\mathcal{P}} = 1\ 4\ 2\ 3$. Parmi les partitions intervallaires, il faut distinguer la *partition intervallaire maximale* qui ne contient que les intervalles forts non-triviaux

maximaux. (Cette notion est définie aussi bien pour l'ensemble $\mathcal{I}$ des intervalles communs que pour n'importe quelle famille $\mathcal{F}^*$ d'intervalles communs.) Nous pouvons alors formuler un théorème de décomposition des permutations similaire au Théorème 2.16. Nous l'énonçons pour l'ensemble $\mathcal{I}$ des intervalles communs d'une permutation, il reste vrai pour toute famille $\mathcal{F}^*$.

Théorème 8.5. *Soit π une permutation sur $\{1, \dots n\}$ et P la partition intervallaire maximale de π. Alors:*

1. *soit tout ensemble $I = \{i, \dots j\}$ déléments consécutifs dans $\pi_{/\mathcal{P}}$ est un intervalle commun de $\pi_{/\mathcal{P}}$ (et $K = \bigcup_{i \leqslant h \leqslant j} I_j$ est un intervalle commun de π);*
2. *soit les seuls intervalles communs de $\pi_{/\mathcal{P}}$ sont triviaux.*

Dans le premier cas du Théorème 8.5, nous avons soit $\pi_{/\mathcal{P}} = \mathbb{I}_k$, soit $\pi_{/\mathcal{P}} = \overline{\mathbb{I}_k} = k\ k-1 \dots 1$. Dans ce cas, $\pi_{/\mathcal{P}}$ est dit *linéaire* (croissante ou décroissante); dans le second cas $\pi_{/\mathcal{P}}$ est dit *première*.

L'application récursive du Théorème 8.5 permet de construire l'arbre des intervalles forts $T_{\mathcal{I}}$ (voir Figure 8.3). Par la suite à chaque intervalle fort I (ou nœud de $T_{\mathcal{I}}$), nous associons une permutation représentative, π_I. Si $\pi[I]$ désigne la sous-permutation induite par les éléments de I, et $\mathcal{P} = \{I_1, \dots I_k\}$ la partition de I en intervalles forts maximaux, alors π_I est la permutation quotiente $\pi[I]_{/\mathcal{P}}$. Par exemple la permutation représentative de l'intervalle fort $\{3, 2, 5, 1, 4\}$ de la permutation π de la Figure 8.2 est $\pi_I = 2\ 4\ 1\ 3$.

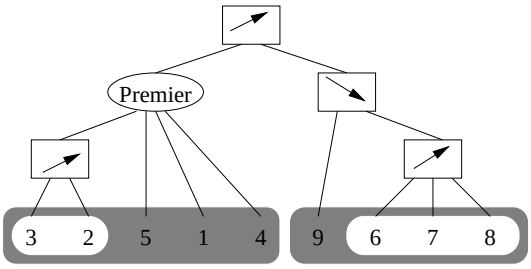

Fig. 8.3. La permutation quotiente associée au nœud "rond" est 2 4 1 3, c'est une permutation première. Dans la suite les intervalles linéaires seront représentés par des rectangles.

Le Théorème 8.5 peut se reformuler de la manière suivante.

Lemme 8.6. *Un intervalle d'une permutation π est un intervalle commun ssi soit c'est un intervalle fort, soit c'est l'union d'intervalles forts consécutifs fils d'un intervalle fort linéaire.*

8.1.1 Calcul des intervalles communs

L'étude des intervalles communs à deux permutations et plus généralement à un ensemble de permutations a connu des récents développement dans le cadre de la génomique comparative et plus particulièrement pour le problème du tri par renversement (voir Section suivante).

Le premier problème considéré est le calcul des intervalles communs d'une permutation. En 2000, Uno et Yagiura [UY00] ont proposé un algorithme en $O(n+N)$ où N est le nombre d'intervalles communs de la permutation π. La complexité est donc possiblement quadratique (borne atteinte pour l'identité par exemple). En 2001, Heber et Stoye [HS01] obtiennent un algorithme de complexité $O(n)$. En fait, ils ne calculent pas l'ensemble des intervalles forts, mais une famille génératrice des intervalles forts, les intervalles irréductibles. Cette famille, contrairement aux intervalles forts, n'a pas l'élégante propriété de pouvoir s'organiser simplement en un arbre.

Récemment, nous avons adapté [BXHP05b] l'algorithme de Uno et Yagiura [UY00] pour qu'il ne calcule que les intervalles forts (voir Chapitre 5). Nous proposons aussi une preuve plus combinatoire et plus simple de cet algorithme. De plus l'arbre des intervalles forts est calculé. De manière indépendante, Bergeron et al. [BCdMR05] ont proposé un autre algorithme pour le calcul de l'arbre des intervalles forts. Ce dernier algorithme est sans doute le plus simple à l'heure actuelle.

8.1.2 Liens avec les graphes de permutation

Nous évoquons ici une correspondance, observée dans [dM03] puis [BXHP05b, BCdMR05], entre les intervalles communs à deux permutations et les modules d'un graphe de permutation (voir Définition **??**).

Soit (π, τ) un réaliseur d'un graphe de permutation. Sans perte de généralité et pour simplifier les notations, nous pouvons supposer que $V = \{1, \dots n\}$ et $\tau = \mathbb{I}_n$. Il est bien connu que les graphes de permutations sont exactement les graphes de comparabilité (voir Chapitre 2) dont le complémentaire est aussi de comparabilité (voir par exemple [Gol80, BLS99]). Un exemple de graphe de permutation est présenté Figure 8.4.

Lemme 8.7. *[dM03] Soit $G = (V, E)$ un graphe de permutation dont le réaliseur est $(\pi, \mathbb{I}_n)$. Un ensemble de sommets M est un module fort de G ssi c'est un intervalle fort de π.*

Il faut noter que la correspondance ne porte que sur les intervalles forts et les modules forts. Par exemple, dans le graphe de la Figure 8.4, l'ensemble $\{8, 10, 11\}$ est un module de G mais pas un intervalle commun de π. En fait, les deux permutations du réaliseur d'un graphe de permutation sont des permutations factorisantes. Une conséquence du Lemme 8.7 et des algorithmes linéaires de calcul de l'arbre des intervalles forts d'une permutation est qu'étant donné le réaliseur d'un graphe de permutation G, $MD(G)$ peut être calculé en $O(n)$.

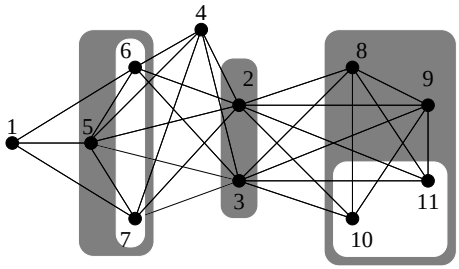

Fig. 8.4. Un graphe de permutation dont le réaliseur est $(\pi, \mathbb{I}_{11})$ avec $\pi = 6\ 7\ 5\ 1\ 4\ 10\ 11\ 9\ 8\ 2\ 3$. Les modules forts non-triviaux sont représentés.

Lorsque l'arbre des intervalles forts d'une permutation π possède la propriété de ne pas avoir de nœud premier, alors $MD(G)$, avec G le graphe de permutation associé à π, ne possède pas non plus de nœud premier. G est donc un cographe.

Lemme 8.8. *Soient π une permutation et $T_{\mathcal{I}}$ son arbre des intervalles forts. Si $T_{\mathcal{I}}$ ne contient que des nœuds linéaires, alors le graphe de permutation G ayant $(\pi, \mathbb{I}_n)$ pour réaliseur est un cographe.*

Pour terminer, remarquons que les graphes de permutation sont des graphes de comparabilité de dimension 2. Les graphes de comparabilité de dimension k sont représentés par un réaliseur comprenant k permutations (voir [Gol80, BLS99]). Malheureusement, la correspondance entre les intervalles forts de k permutations et les modules forts des graphes de comparabilités n'est plus valide.

8.2 Tri par renversements et scénarios parfaits

Une *permutation signée* d'un ensemble $\{1, \ldots n\}$ est une permutation de $\{1, \ldots n\}$ où un signe (positif ou négatif) est affecté à chaque élément. Toute permutation considérée dans la suite sera signée.

Renverser un intervalle I d'une permutation signée consiste à renverser l'ordre entre les éléments de I et changer leurs signes (voir Figure 8.5). Nous noterons $\overline{\pi}$ la permutation π entièrement renversée. Pour la permutation π de la Figure 8.5, $\overline{\pi} = \overline{4}\ 2\ 6\ \overline{1}\ 3\ \overline{5}$.

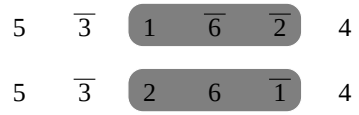

Fig. 8.5. Renversement d'intervalle d'une permutation signée π.

Définition 8.9. *Un* scénario *entre deux permutations π et τ est une séquence $\mathcal{S}$ de renversements qui transforme π en τ (ou π en $\overline{\tau}$). La* longueur *de $\mathcal{S}$ est le nombre de renversements qu'il contient. La distance de renversement est la longueur minimum d'un scénario.*

Sans perte de généralité et pour simplifier les notations, nous supposerons que τ est la permutation identité $\mathbb{I}_n$. Nous parlerons alors de scénario pour π. Un scénario de longueur minimum est appelé *scénario parcimonieux*. Le calcul de la longueur d'un scénario parcimonieux est linéaire [BMY01], alors que la complexité du meilleur algorithme calculant un scénario est sous-quadratique, $O(n\sqrt{n \log n})$ [TS04].

Remarque 8.10. Etant entendu qu'à une étape donnée d'un scénario, un intervalle renversé n'est pas nécessairement un intervalle de la permutation d'origine, nous identifierons un renversement (plus généralement un intervalle) par l'ensemble des éléments qu'il contient.

Définition 8.11. *[FV04] Un scénario $\mathcal{S}$ est* parfait *pour la permutation π et un ensemble $\mathcal{F}$ d'intervalles communs de π, si aucun renversement de $\mathcal{S}$ ne chevauche un quelconque intervalle commun de $\mathcal{F}$. Un scénario parfait est* optimal *s'il est de longueur minimum parmi l'ensemble des scénarios parfaits.*

Lemme 8.12. *Soient π une permutation signée et $\mathcal{F}$ un ensemble d'intervalles communs de π. Un scénario $\mathcal{S}$ est parfait pour π et $\mathcal{F}$ ssi il est parfait pour π et la fermeture $\mathcal{F}^*$ de $\mathcal{F}$.*

Preuve. Soient I et J deux intervalles de $\mathcal{F}$ se chevauchant. Puisque $\mathcal{S}$ est parfait, aucun renversement ne chevauche I et J et donc aussi $I \cup J$, $I \cap J$, $I \setminus J$ et $J \setminus I$. Reciproquement, puisque $\mathcal{F} \subseteq \mathcal{F}^*$, si $\mathcal{S}$ est parfait pour $\mathcal{F}^*$, alors $\mathcal{S}$ est parfait pour $\mathcal{F}$. $\square$

Pour simplifier la discussion, nous ne considérerons que l'ensemble $\mathcal{I}$ de tous les intervalles communs de π. Notons les résultats applicables à $\mathcal{I}$ le sont aussi à tout ensemble $\mathcal{F}^*$ d'intervalles communs de π.

Il existe toujours un scénario parfait [FV04]. La difficulté n'est donc pas de trouver un scénario parfait, mais un parfait optimal.

Théorème 8.13. *[BBCP04] Un scénario $\mathcal{S}$ pour une permutation signée π est parfait ssi chaque renversement de $\mathcal{S}$ est soit un intervalle fort de π, soit l'union d'intervalles forts fils d'un intervalle premier de $T_{\mathcal{I}}$*

Preuve. $\Rightarrow$ Soit I un renversement de $\mathcal{S}$. Si I est un intervalle fort, alors par définition, il ne chevauche aucun intervalle commun. Supposons que I soit l'union de fils d'un intervalle premier J. I ne chevauche aucun intervalle commun contenant J ni aucun intervalle commun contenu dans un fils de J. Puisque J est premier, il n'y a pas d'autres intervalles communs.

$\Leftarrow$ Soit I un renversement de $\mathcal{S}$ et soit $\mathcal{P} = \{I_1, \dots I_k\}$ la partition de I en intervalles forts maximaux contenus dans I. Si $k \geqslant 2$, les intervalles $I_1 \dots I_k$ doivent être fils d'un même intervalle fort J (sinon I chevaucherait le père de n'importe quel intervalle I_j). J ne peut pas être linéaire. En effet puisque $I \subset J$, il existerait un fils K de J non contenu dans I et donc il existerait un intervalle $I_j \in \mathcal{P}$ tel que $I_j \cup K$ soit un intervalle commun de π. Or cet intervalle $I_j \cup K$ chevaucherait I. $\square$

Le Théorème 8.13 montre que l'arbre $T_{\mathcal{I}}$ des intervalles forts est un guide précieux pour le calcul d'un scénario parfait optimal. Pour cela, nous devons signer $T_{\mathcal{I}}$ afin de mieux prendre en compte les informations présentes sur la permutation. Notons que la définition d'intervalle commun reste indépendante du signe des éléments. Affecter un signe positif, respectivement négatif, à un intervalle fort I signifie qu'il faut trier la permutation induite par I vers $\mathbb{I}_k$, respectivement $\overline{Idk}$.

Définition 8.14. *L'*arbre signé *d'une permutation signée π est l'arbre $T_{\mathcal{I}}$ des intervalles forts de π au quel un signe a été affecté à chaque nœud de la manière suivante :*

1. *le signe d'une feuille de $T_{\mathcal{I}}$ est le signe de l'élément correspondant dans π;*
2. *le signe d'un nœud linéaire est $+$ si l'intervalle est croissant, $-$ s'il est décroissant;*
3. *le signe d'un nœud premier est celui du signe de son père si celui-ci est linéaire.*

Remarquons donc qu'en général, la Définition 8.14 ne signe pas tous les nœuds de $T_{\mathcal{I}}$: c'est le cas des nœuds premiers dont le père est premier. La Figure 8.6 montre un exemple d'arbre totalement signé.

Lemme 8.15. *[BBCP04] Si I est un intervalle fort de π ayant un père linéaire J de signe opposé à celui de I, alors I appartient à tout scénario parfait.*

Preuve. Soient $\mathcal{S}$ un scénario parfait et I un nœud négatif dont le père J est linéaire croissant (i.e. positif). Supposons que $I \notin \mathcal{S}$. Notons que I n'est pas premier. D'après le Théorème 8.13, tout renversement contenant I contient aussi J. Soit m le nombre de renversements contenant J. Si m est pair, puisque J est croissant, $\mathcal{S}$ tri π vers $\mathbb{I}_n$. Si I est une feuille, elle le restera après les m renversements : contradiction. De même, si I est un nœud linéaire, il est décroissant et le restera après les m renversements le contenant : contradiction. L'argument est similaire si m est impair ou I positif et J négatif. $\square$

Première conséquence intéressante du Lemme 8.15, un scénario parfait pour toute permutation correspondant à un cographe se lit directement sur $T_{\mathcal{I}}$. Ces permutations ont notamment été étudiée par [BBC04].

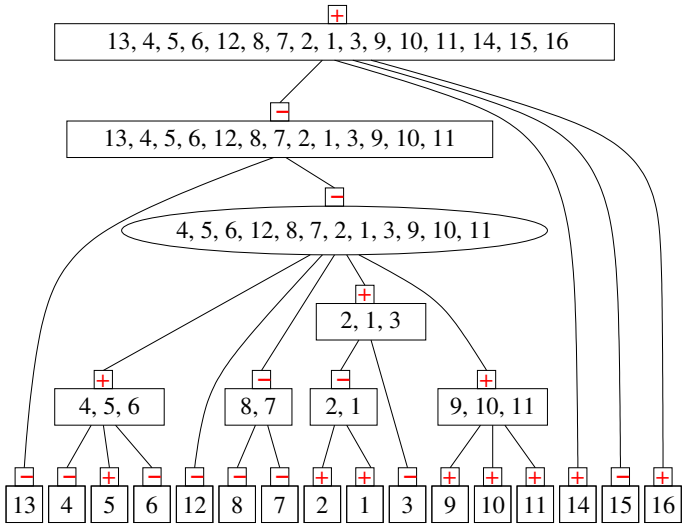

Fig. 8.6. L'arbre signé des intervalles forts de la permutation $\pi = \overline{13}\ \overline{4}\ 5\ \overline{6}\ \overline{12}\ \overline{8}\ \overline{7}\ 2\ 1\ \overline{3}\ 9\ 10\ 11\ 14\ \overline{15}\ 16$. Notons que cet arbre est complètement signé. π ne possède qu'un seul intervalle fort premier dont la permutation quotiente associée est $2\ \overline{5}\ \overline{3}\ 1\ 4$. Si on modèlisait une partie du chromosome X de l'homme, alors π représenterait portion la correspondante du chromosome X du rat. Un scénario de tri transformant π en $\mathbb{I}_n$ est un scénario d'évolution possible du chromosome X entre ces deux espèces.

Lemme 8.16. *[BBCP04] Si π est une permutation signée n'ayant aucun intervalle fort premier, alors il exsite un unique scénario parfait. De plus, ce scénario peut se calculer en temps $O(n)$.*

Le problème essentiel reste donc la détermination des signes des intervalles premiers fils de premiers. Actuellement, on se sait pas mieux faire que d'essayer toutes les affectations des signes possible.

Théorème 8.17. *[BBCP04] Etant donnée un permutation signée π et son arbre signé, l'algorithme 17 calcule un scénario parfait optimal en temps $O(2^k \times n\sqrt{n \log n})$.*

Preuve. La correction de l'Algorithme 17 est impliquée par le Lemme 8.15 et le Théorème 8.13. Quant à sa complexité, observons que : 1) 2^k boucles internes sont effectuées; et 2) le tri de π_I pour un intervalle premier I se fait en temps $O(n\sqrt{n \log n})$ [TS04]. Pour être complet, rappelons que le calcul de $T_{\mathcal{I}}$ [BXHP05b, BCdMR05] et l'affectation des signes nécessitent un coût $O(n)$. $\square$

Remarque 8.18. L'Algorithme 17 reste valide si on ne considère qu'une sous famille $\mathcal{F}$ d'intervalles communs. Il faut alors signer l'arbre $T_{\mathcal{F}^*}$.

Algorithme 17 : Tri parfait

Données : une permutation signée π et son arbre signé $T_{\mathcal{I}}$
Résultat : Un scénario parfait optimal
début
  Soit $I_1, \ldots I_k$ les intervalles forts non-signés;
  pour chaque *affectation de signes aux k intervalles non-signés parmi les 2^k possibles* **faire**
    $\mathcal{S} \leftarrow \emptyset$;
    pour chaque *intervalle fort I premier* **faire**
      Soit π_I la permutation quotiente associée;
      si *I est positif* **alors**
        Calculer un scénario parsimonieux de π_I vers $\mathbb{I}_k$;
      sinon
        Calculer un scénario parsimonieux de π_I vers $\overline{\mathbb{I}_k}$;
      Soit $\mathcal{S}_I$ le meilleur des deux scénario ci-dessus;
      pour chaque *renversement R de $\mathcal{S}_I$* **faire**
        Ajouter à $\mathcal{S}$ le renversement $\bigcup_{J \in R} J$ (J fils de I);
    Ajouter à $\mathcal{S}$ tous les intervalles I linéaires de père linéaire J n'ayant pas le même signe que I;
  Retourner le scénario $\mathcal{S}$ optimal;
fin

Nous pouvons déduire du Théorème 8.17 que, pour les permutations signées dont l'arbre T_I est totalement signé, le calcul d'un scénario parfait optimal est polynomial.

8.3 Discussion et notes bibliographiques

Récemment, Sagot et Tannier [ST05] ont proposé un algorithmique polynomial pour le calcul d'un scénario parfait et parcimonieux. Si un tel scénario existe, il sera toujours trouvé par leur algorithme. Sinon l'algortihme stoppe lorsqu'il détecte une obstruction. Ces obstructions sont exprimées à l'aide de calcul de distance. Autrement dit, elles reposent essentiellement sur la théorie de Hannenhali et Pevzner [HP99]. Pourtant, il serait intéressant de pouvoir caractériser ces permutations à l'aide de leur arbre des intervalles forts. Pour cela, il faudrait peut-être reconsidérer des objets combinatoires tels que le graphe des chevauchements des intervalles [BMS05] qui ont été utilisés dans la théorie de Hannenhali et Pezner [HP99]. Par exemple, quels sont les liens entre ce graphe des chevauchements et l'arbre des intervalles forts ?

Il existe d'autres notions que les intervalles communs pour modéliser la structure conservée entre deux génomes. Par exemple, Béal et al. [BBCR04] s'intéressent à la notion d'*équipe de gènes*. Un ensemble de gènes G forme une δ-équipe dans une permutation π si la distance séparant deux membres successifs de G est au plus δ dans π. Béal et al. proposent un algorithme basé sur

des techniques d'affinage de partition pour le calcul des équipes de gènes communes à plusieurs génomes. Ce problème peut s'exprimer et se généraliser en terme de graphes [HPR04] de la manière suivante : étant donnés deux graphes G_1 et G_2 sur le même ensemble de sommets, calculer les ensembles maximaux de sommets qui induisent un sous-graphe de G_1 et de G_2. Ces ensembles sont appelés *composantes connexes communes* à G_1 et G_2. L'aspect algorithmique est assez bien maîtrisé, le calcul des composantes connexes communes utilise aussi les techniques d'affinage de partition. Par contre, l'aspect combinatoire est peut-être moins bien connu. Des liens ont pu être mis en évidence avec la décomposition modulaire.

Tarte tomates, basilic et chèvre

Ingrédients : tomates pelées, 3 oignons blancs, chèvre frais, 2 œufs, 20cl de crème fraiche légère, huile d'olive.

Faire revenir les tomates peles dans l'huile d'olive avec les oignons blancs et le basilic. Dans un bol, mélanger les œufs, la crème et la sauce tomates. Découper les fromages de chèvre en rondelles.

Couvrir le fond du plat avec une feuille alu. Disposer les rondelles de chèvre sur le plat et étaler la sauce tomate par dessus. Recouvrir la préparation avec la pâte. Mettre au four chaud pendant 25 minutes (250). Démouler la tarte et la sopoudrer avec l'origan.

Bibliographie

[BBC04] S. Bérard, A. Bergeron, and C. Chauve, *Conserved structures in evolution scenarios*, Comparative Genomics, RECOMB 2004 International Workshop, RCG, Lecture Notes in Computer Science, vol. 3388, 2004, pp. 1–15.

[BBCP04] A. Bergeron, S. Bérard, C. Chauve, and C. Paul, *Sorting by reversal is not always difficult*, Workshop on Algorithm for Bio-Informatics (WABI), Lecture Notes in Computer Science, 2004.

[BBCR04] M.P. Béal, A. Bergeron, S. Corteel, and M. Raffinot, *An algorithmic view of gene teams.*, Theoretical Computer Science **320** (2004), no. 2-3, 395–418.

[BCdMR05] A. Bergeron, C. Chauve, F. de Montgolfier, and M. Raffinot, *Computing common intervals of k permutations, with applications to modular decomposition of graphs*, European Symposium on Algorithms (ESA), Lecture Notes in Computer Science, no. 3669, 2005, pp. 779–790.

[BCHP03] A. Bretscher, D.G. Corneil, M. Habib, and C. Paul, *A simple linear time LexBFS cograph recognition algorithm*, 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Computer Science, vol. 2880, 2003, pp. 119–130.

[Ber61] C. Berge, *Frbung von graphen, deren smtliche bzw. deren ungerade kreise starr sind*, Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe **10** (1961), 114.

[BL76] K.S. Booth and G.S. Lueker, *Testing for the consecutive ones property, interval graphs and graph planarity using pq-tree algorithm*, J. Comput. Syst. Sci. **13** (1976), 335–379.

[Bla78] A. Blass, *Graphs with unique maximal clumpings*, Journal of Graph Theory **2** (1978), 19–24.

[BLS99] A. Brandstädt, VB. Le, and J. Spinrad, *Graph classes: a survey*, SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics, 1999.

[BMS05] A. Bergeron, J. Mixtacki, and J. Stoye, *The inversion distance problem*, Mathematics of evolution and phylogeny, Oxford University Press, 2005, pp. 262–290.

[BMY01] D.A. Bader, B.M.E. Moret, and M. Yan, *A linear time algorithm for computing inversion distance between signed permutations with experi-*

mental study, Journal of Computational Biology **8** (2001), no. 1, 483–491.

[Bou87] A. Bouchet, *Reducing prime graphs and recognizing circle graphs*, Combinatorica **7** (1987), 243–254.

[Bre04] A. Bretscher, *Lexbfs based recognition algorithms for cographs and related families*, Ph.D. thesis, University of Toronto, dept. of Computer Science, 2004.

[BXHP05a] B.-M. Bui-Xuan, M. Habib, and C. Paul, *From permutation to graph algorithm*, Tech. Report RR05-017, LIRMM, March 2005, Submitted.

[BXHP05b] ———, *Revisiting uno and yagiura's algorithm*, 16th International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Computer Science, vol. 3827, 2005, pp. 146–155.

[Cap96] C. Capelle, *Décomposition de graphes et permutations factorisantes*, Ph.D. thesis, Univ. de Montpellier II, 1996.

[Cap97] A. Caprara, *Sorting by reversals is difficult*, 1st Conference on Computational Molecular Biology (RECOMB), ACM Press, 1997, pp. 75–83.

[CC] V. Chvàtal and W. Cook, *Traveling salesman problem*, http://www.tsp.gatech.edu/.

[CE80] W.H. Cunnigham and J. Edmonds, *A combinatorial decomposition theory*, Canadian Journal of Mathematics **32** (1980), no. 3, 734–765.

[CER93] B. Courcelle, J. Engelfriet, and G. Rozenberg, *Handle rewriting graph grammars*, Journal of Computer and System Science **46** (1993), 218–270.

[CH94] A. Cournier and M. Habib, *A new linear algorithm of modular decomposition*, Trees in algebra and programming (CAAP), Lecture Notes in Computer Science, vol. 787, 1994, pp. 68–84.

[CHdM02] C. Capelle, M. Habib, and F. de Montgolfier, *Graph decompositions and factorizing permutations*, Discrete Mathematics and Theoretical Computer Science **5** (2002), 55–70.

[CHL$^+$00] D.G. Corneil, M. Habib, J.-M. Lanlignel, B. Reed, and U. Rotics, *Polynomial time algorithm for the* 3*-clique-width problem*, 4th Latin American Symposium on Theoretical Informatics (LATIN), Lecture Notes in Computer Science, vol. 1776, 2000, pp. 126–134.

[CHM81] M. Chein, M. Habib, and M.-C. Maurer, *Partitive hypergraphs*, Discrete Mathematics **37** (1981), 35–50.

[CI98] A. Cournier and P. Ille, *Minimal indecomposable graphs*, Discrete Mathematics **183** (1998), 61–80.

[CJS72] D.D. Cowan, L.O. James, and R.G. Stanton, *Graph decomposition for undirected graphs*, 3rd S-E Conference on Combinatorics, Graph Theory and Computing, Utilitas Math, 1972, pp. 281–290.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Algorithms*, MIT Press, 1990.

[CLSB81] D.G. Corneil, H. Lerchs, and L.K. Stewart-Burlingham, *Complement reducible graphs*, Discrete Applied Mathematics **3** (1981), no. 1, 163–174.

[Cor04] D.G. Corneil, *Lexicographique breadth first search - a survey*, 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Computer Science, vol. 3353, 2004, pp. 1–19.

[COS97] D.G. Corneil, S. Olariu, and L.K. Stewart, *Asteroidal triple-free graphs*, SIAM Journal on Discrete Mathematics **10** (1997), no. 3, 399–430.

[COS98] ———, *The ultimate interval graph recognition algorithm?*, 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1998, pp. 175–180.

[COS99] ———, *LBFS orderings and cocomparability graphs*, 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1999, pp. 883–884.

[Cou86] B. Courcelle, *The expression of graph properties and graph transformation in monadic second ordre logic*, Journal of Algorithms **7** (1986).

[CP04] C. Crespelle and C. Paul, *Fully-dynamic recognition algorithm and certificate for directed cographs*, 30th International Workshop on Graph Theoretical Concepts in Computer Science (WG), Lecture Notes in Computer Science, vol. 3353, 2004, pp. 93–104.

[CP05] ———, *Fully dynamic algorithm for modular decomposition and recognition of permutation graphs*, 31st International Workshop on Graph Theoretical Concepts in Computer Science (WG), Lecture Notes in Computer Science, 2005.

[CP06] ———, *Fully-dynamic recognition algorithm and certificate for directed cographs*, Discrete Applied Mathematics (2006), A paraître.

[CPS85] D.G. Corneil, Y. Perl, and L.K. Stewart, *A linear time recognition algorithm for cographs*, SIAM Journal on Computing **14** (1985), no. 4, 926–934.

[Cre05] C. Crespelle, *A note on fully dynamic modular decomposition*, personal communication, 2005.

[CRST02] M. Chudnovsky, N. Roberston, P. Seymour, and R. Thomas, *The strong perfect graph theorem*, manuscript, 2002.

[Cun82] W.H. Cunningham, *Decomposition of directed graphs*, SIAM Journal on Algebraic and Discrete Methods **3** (1982), 214–228.

[Dah95a] E. Dahlhaus, *Efficient parallel algorithms for cographs and distance hereditary graphs*, Discrete Applied Mathematics **57** (1995), 29–54.

[Dah95b] ———, *Efficient parallel modular decomposition*, 21th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), Lecture Notes in Computer Science, vol. 1017, 1995, pp. 21–32.

[Dah00] ———, *Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition*, Journal of Algorithms **36** (2000), no. 2, 205–240.

[DF97] R.G. Downey and M.R. Fellows, *Parameterized complexity*, Springer, 1997.

[DGM97] E. Dahlhaus, J. Gustedt, and R.M. McConnell, *Efficient and practical modular decomposition*, 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1997, pp. 26–35.

[DGM01] ———, *Efficient and practical algorithm for sequential modular decomposition algorithm*, Journal of Algorithms **41** (2001), no. 2, 360–387.

[DGM02] ———, *Partially complemented representation of digraphs*, Discrete Mathematics and Theoretical Computer Science **5** (2002), no. 1, 147–168.

[dM03] F. de Mongolfier, *Décomposition modulaire des graphes - théorie, extensions et algorithmes*, Ph.D. thesis, Univ. de Montpellier II, 2003.

[DNB96] F.F. Dragan, F. Nicolai, and A. Brandstädt, *LexBFS-orderings and power of graphs*, 22th International Workshop on Graph-Theoretic Con-

cepts in Computer Science (WG), Lecture Notes in Computer Science, vol. 1197, 1996, pp. 166–180.

[EGI97] D. Eppstein, Z. Galil, and G. Italiano, *Dynamic graph algorithms*, Handbook of Algorithms and Theory of Computation, Chapter 22, 1997.

[EGMS94] A. Ehrenfeucht, H.N. Gabow, R.M. McConnell, and S.L. Sullivan, *An $o(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs*, Journal of Algorithms **16** (1994), 283–294.

[EHR99] A. Ehrenfeucht, T. Harju, and G. Rozenberg, *The theory of 2-structures*, World Scientific, 1999.

[FK00] Feigenbaum and Kannan, *Dynamic graph algorithms*, Handbook of Discrete and Combinatorial Mathematics (Rosen, Michaels, Gross, Grossman, and Shier, eds.), CRC Press, 2000.

[FV04] M. Figeac and J.-S. Varré, *Sorting by reversals with common intervals*, 4th International Workshop on Algorithms in Bioinformatics - WABI, Lecture Notes in Computer Science, vol. 3240, 2004, pp. 26–37.

[Gal67] T. Gallai, *Transitiv orientierbare graphen*, Acta Mathematica Acad. Sci. Hungar. **18** (1967), 25–66.

[GHS89] C.P. Gabor, W.L. Hsu, and K.J. Suppowit, *Recognizing circle graphs in polynomial time*, Journal of the ACM **36** (1989), 435–473.

[Gol77] M.C. Golumbic, *The complexity of comparability graphs recongition and coloring*, Computing **18** (1977), 199–208.

[Gol80] ———, *Algorithmic graph theory and perfect graphs*, Academic Press, 1980.

[HdMP04] M. Habib, F. de Montgolfier, and C. Paul, *A simple linear-time modular decomposition algorithm*, 9th Scandinavian Workshop on Algorithm Theory (SWAT), Lecture Notes in Computer Science, vol. 3111, 2004, pp. 187–198.

[HHS95] M. Habib, M. Huchard, and J.S. Spinrad, *A linear algorithm to decompose inheritance graphs into modules*, Algorithmica **13** (1995), 573–591.

[HM79] M. Habib and M.-C. Maurer, *On the x-join decomposition of undirected graphs*, Discrete Applied Mathematics **1** (1979), 201–207.

[HM91] W.-L. Hsu and T.-H. Ma, *Substitution decomposition on chordal graphs and applications*, 2nd International Symposium on Algorithms (ISA), Lecture Notes in Computer Science, vol. 557, 1991, pp. 52–60.

[HMPV00] M. Habib, R.M. McConnell, C. Paul, and L. Viennot, *Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing.*, Theoretical Computer Science **234** (2000), 59–84.

[Hop71] J. Hopcroft, *An $n \log n$ algorithm for minimizing states in a finite automaton*, Theory of machines and computations (1971), 189–196.

[HP99] S. Hannenhali and P.A. Pevzner, *Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals*, SIAM Journal on Computing **46** (1999), no. 1, 1–27.

[HP05] M. Habib and C. Paul, *A simple linear time algorithm for cograph recognition*, Discrete Applied Mathematics **145** (2005), no. 2, 183–187.

[HPR04] M. Habib, C. Paul, and M. Raffinot, *Common connected components of interval graphs*, 15th Annual Combinatorial Pattern Matching Symposium (CPM), Lecture Notes in Computer Science, vol. 3109, 2004, pp. 347–358.

[HPV98] M. Habib, C. Paul, and L. Viennot, *A synthesis on partition refinement: a useful routine for strings, graphs, boolean matrices and automata*, 15th Symposium on Theoretical Aspect of Computer Science (STACS), Lecture Notes in Computer Science, vol. 1373, 1998, pp. 25–38.

[HPV99] ———, *Partition refinement : an interesting algorithmic tool kit*, International Journal of Foundation of Computer Science **10** (1999), no. 2, 147–170.

[HS01] S. Heber and J. Stoye, *Finding all common intervals of k permutations*, 12th Annual Symposium on Combinatorial Pattern Matching (CPM), Lecture Notes in Computer Science, no. 2089, 2001, pp. 207–218.

[HSS02] P. Hell, R. Shamir, and R. Sharan, *A fully dynamic algorithm for recognizing and representing proper interval graphs*, SIAM Journal on Computing **31** (2002), no. 1, 289–305.

[Hsu92] W.-L. Hsu, *A simple test for interval graphs*, 3rd International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Computer Science, vol. 557, 1992, pp. 459–468.

[HT84] D. Harel and R.E. Tarjan, *Fast algorithm for finding nearest common ancestor*, SIAM Journal on Computing **13** (1984), 338–355.

[Iba99] L. Ibarra, *Fully dynamic algorithms for chordal graphs*, 10th ACM-SIAM Annual Symposium on Discrete Algorithm (SODA), 1999, pp. 923–924.

[JO92a] B. Jamison and S. Olariu, *Recognizing p_4-sparse graphs in linear time*, SIAM Journal on Discrete Mathematics **21** (1992), 381–406.

[JO92b] ———, *A tree representation of p_4-sparse graphs*, Discrete Applied Mathematics **35** (1992), 115–129.

[JO95a] ———, *A linear time recognition algorithm for p_4-reducible graphs*, Theoretical Computer Science **1-2** (1995), 329–344.

[JO95b] ———, *P-components and the homogeneous decomposition of graphs*, SIAM Journal on Discrete Mathematics **8** (1995), no. 3, 448–463.

[KM89] N. Korte and R. Möhring, *An incremental linear-time algorithm for recongizing interval graphs*, SIAM journal of Computing **18** (1989), 68–81.

[Möh85a] R.H. Möhring, *Algorithmic aspect of comparability graphs and interval graphs*, Graphs and orders (1985), 42–101.

[Möh85b] ———, *Algorithmic aspect of the substitution decomposition in optimization over relations, set systems and boolean functions*, Annals of Operations Research **4** (1985), 195–225.

[MR84] R.H. Möhring and F.J. Radermacher, *Substitution decomposition for discrete structures and connections with combinatorial optimization*, Annals of Discrete Mathematics **19** (1984), 257–356.

[MS89] J.H. Muller and J.P. Spinrad, *Incremental modular decomposition*, Journal of the ACM **36** (1989), no. 1, 1–19.

[MS94a] T.-H. Ma and J.P. Spinrad, *An $o(n^2)$ algorithm for the undirected split decomposition*, Journal of Algorithms **16** (1994), no. 1, 145–160.

[MS94b] R.M. McConnell and J.P. Spinrad, *Linear-time modular decomposition and efficient transitive orientation of comparability graphs*, 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1994, pp. 536–545.

[MS97] ———, *Linear time transitive orientation*, 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1997, pp. 19–25.

[MS99] ———, *Modular decomposition and transitive orientation*, Discrete Mathematics **201** (1999), 189–241.

[MS00] ———, *Ordered vertex partitioning*, Discrete Mathematics and Theoretical Computer Science **4** (2000), 45–60.

[Nie02] R. Niedermeier, *Invitation to fixed-parameter algorithms*, Habilitation, 2002.

[Pau98] C. Paul, *Parcours en largeur lexicographique: un algorithme de partitionnement. application aux graphes et généralisation*, Ph.D. thesis, Univ. de Montpellier II, 1998.

[PT87] R. Paigue and R.E. Tarjan, *Three partition refinement algorithms*, SIAM Journal on Computing **16** (1987), no. 6, 973–989.

[RS86] N. Roberston and P.D. Seymour, *Graph minors ii: Algorithmic aspects of tree-width*, Journal of Algorithms **7** (1986), 309–322.

[RS91] ———, *Graph minors x: Obstructions to tree-decomposition*, Journal on Combinatorial Theory Series B **18** (1991), 255–288.

[RS04] ———, *Graph minors xx: Wagner's conjecture*, Journal of Combinatorial Theory B **92** (2004), no. 2, 325–357.

[RTL76] D.J. Rose, R.E. Tarjan, and G.S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM Journal on Computing **5** (1976), no. 2, 266–283.

[San92] D. Sankoff, *Edit distance for genome comparaison based on non-local operations*, 3rd Annual Symposium on Combinatorial Pattern Matching - CPM, Lecture Notes in Computer Science, no. 644, 1992, pp. 121–135.

[Sei74] S. Seinsche, *On a property of the class of n-colorable graphs*, Journal of Combinatorial Theory (B) (1974), 191–193.

[SF70] L. Shevrin and N. Filippov, *Partially ordered sets and their comparability graphs*, Siberian Journal of Mathematics **11** (1970), 648–667.

[Sha67] L. Shapley, *On committees*, New methods of though and procedure, Springer-Verlag, 1967, pp. 648–667.

[Sim91] K. Simon, *A new simple linear algorithm to recongize interval graphs*, Workshop on Computational Geometry - CG91, Lecture Notes in Computer Science, no. 553, 1991, pp. 289–308.

[Spi85] J.P. Spinrad, *On comparability and permutation graphs*, SIAM Journal on Computing **14** (1985), no. 3, 191–193.

[Spi94] ———, *Recognition of circle graphs*, Journal of Algorithms **16** (1994), 264–282.

[Spi03] ———, *Efficient graph representation*, Fields Institute Monographs, vol. 19, American Mathematical Society, 2003.

[SS04] R. Shamir and R. Sharan, *A fully dynamic algorithm for modular decomposition and recognition of cographs*, Discrete Applied Mathematics **136** (2004), no. 2-3, 329–340.

[ST05] M.-F. Sagot and E. Tannier, *Perfect sorting by reversals*, 11th Annual International Conference on Computing and Combinatorics (COCOON), Lecture Notes in Computer Science, no. 3595, 2005, pp. 45–52.

[Sum73] D.P. Sumner, *Graphs indecomposable with respect to the x-join*, Discrete Mathematics **6** (1973), 281–298.

[Tar83] R.E. Tarjan, *Data-structures and network algorithms*, Society for Industrial and Applied Mathematics, 1983.

[TS04] E. Tannier and M.-F. Sagot, *Sorting by reversals in subquadratic time*, 15th Annual Combinatorial Pattern Matching Symposium (CPM), Lecture Notes in Computer Science, vol. 3109, 2004, pp. 1–13.

[UY00] T. Uno and M. Yagiura, *Fast algorithms to enumerate all common intervals of two permutations*, Algorithmica **26** (2000), no. 2, 290–309.

[Wag37] K. Wagner, *Uber eine eigenschaft der ebenen komplexe*, Math. Annals **114** (1937), 570–590.