Numéro d'identification :

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

— SCIENCES ET TECHNIQUE DU LANGUEDOC —

THÈSE

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

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

Sur quelques interactions entre structures de données et algorithmes efficaces pour les ordres et les graphes

par

Eric THIERRY

Soutenue le 25 Octobre 2001 devant le Jury composé de :

Vincent BOUCHITTÉ, Maître de Conférence HDR, ENS Lyon, ... Rapporteur
Olivier COGIS, Professeur, Université Montpellier II, ... Examinateur
Roland DUCOURNAU, Professeur, Université Montpellier II, ... Examinateur
Michel HABIB, Professeur, Université Montpellier II, ... Directeur de Thèse
Claude JARD, Directeur de Recherches CNRS, IRISA, Rennes, ... Rapporteur
Michel MORVAN, Professeur, Université Paris VII, ... Directeur de Thèse
Lhouari NOURINE, Professeur, Université Clermont-Ferrand II, ... Examinateur

Table des matières

Introduction

Plusieurs problématiques ont été abordées au cours de cette thèse : algorithmique des graphes et des ensembles ordonnées, combinatoire sur les treillis, algorithmique parallèle. Un des principaux fils conducteurs du travail qui a été mené, est la constante interaction entre structures de données et algorithmes dans l'élaboration de solutions à un problème donné. Plusieurs cas de figure peuvent se présenter dans la recherche de solutions à un problème :

1. Ayant en tête un schéma précis d'algorithme pour résoudre le problème (après avoir modélisé le problème, on a une idée des opérations à effectuer et de leur ordre d'exécution), on cherche la structure de données la plus efficace pour représenter les données et traiter les opérations souhaitées.
2. La structure de données représentant les données en entrée du problème est fixée, on cherche alors l'algorithme le plus efficace utilisant directement cette structure de données.
3. On se permet de jouer à la fois sur les structures de données et les algorithmes pour proposer des solutions.

Ces trois schémas de travail ont été rencontrés au cours de cette thèse :

1. La représentation des graphes par la structure de données d'arbre lexicographique, utilisée pour résoudre un problème d'élagage, correspond au cas 1 (voir Partie IV).
2. La résolution de problèmes d'optimisation sur les graphes distance-héréditaires, donnés sous forme d'une séquence d'élagage, correspond au cas 2 (voir Partie IV).
3. L'élaboration de méthodes de codage d'ordres pour traiter efficacement la comparaison entre éléments et d'autres opérations sur les ordres, correspond au cas 3 (voir Partie I).

Le choix des structures de données et des algorithmes est bien sûr très lié au modèle de calcul considéré. Par exemple, les contraintes des modèles séquentiels sont différentes de celles des modèles parallèles. La majeure partie de ce travail de thèse s'est placée dans un cadre séquentiel, mais des questions de parallélisme ont été rencontrées au cours d'une collaboration concernant l'analyse de performance d'algorithmes sur les graphes dans le modèle à gros grains CGM (voir Annexe A).

Pour guider les choix de représentation des données, il peut être bon de mieux connaître les propriétés structurelles des objets manipulés. C'est dans ce cadre que se placent les études des treillis distributifs et des extensions d'ordres présentées dans la partie III.

La première partie de ce mémoire est une synthèse. Elle concerne la représentation des ensembles ordonnées, ou ordres, en informatique. Une problématique usuelle consiste à avoir un ordre fixé $P = (X, \leq_P)$ et à vouloir répondre efficacement à des tests de comparaisons "$x \leq_P y$?" ou bien à d'autres opérations propres aux ordres (comme le calcul de bornes supérieures ou inférieures). Pour cela, de nombreuses méthodes de codage existent dans la littérature : elles se composent de structures de données représentant l'ordre en mémoire et d'algorithmes qui leur sont associés pour traiter les opérations sur l'ordre. La figure 1 illustre l'une d'elles. Elle représente une hiérarchie utilisée dans une application pour modéliser des réseaux téléphoniques. Cette hiérarchie est un ordre sur les types encadrés (les liens relient des types comparables et on convient que les plus petits types sont en bas). Supposons que nous souhaitions coder cet ordre pour pouvoir répondre efficacement à toutes les questions de comparaison "est-ce que x est plus petit que y ?" où x et y sont deux éléments quelconques de l'ordre. Il y a 22 éléments et une première méthode peut consister à stocker en mémoire les 22×22 réponses aux 22×22 comparaisons possibles, ce qui représente un espace de 22×22 bits. C'est le codage correpondant à la matrice binaire de la relation d'ordre. Sur la figure, un autre codage est présenté : chaque élément reçoit un mot de 8 bits de telle manière qu'un élément x est plus petit qu'un élément y si et seulement si chaque bit du mot de x est inférieur au bit correspondant dans le mot de y. Ce codage ne nécessite que 22×8 bits, réalisant ainsi une importante économie en espace par rapport au codage précédent.

Tout comme on parle de compression de textes ou d'images, on pourra parler ici de ***compression d'ordres***, puisque l'un des objectifs des méthodes de codage d'ordre consiste à minimiser l'espace total de stockage, tout en

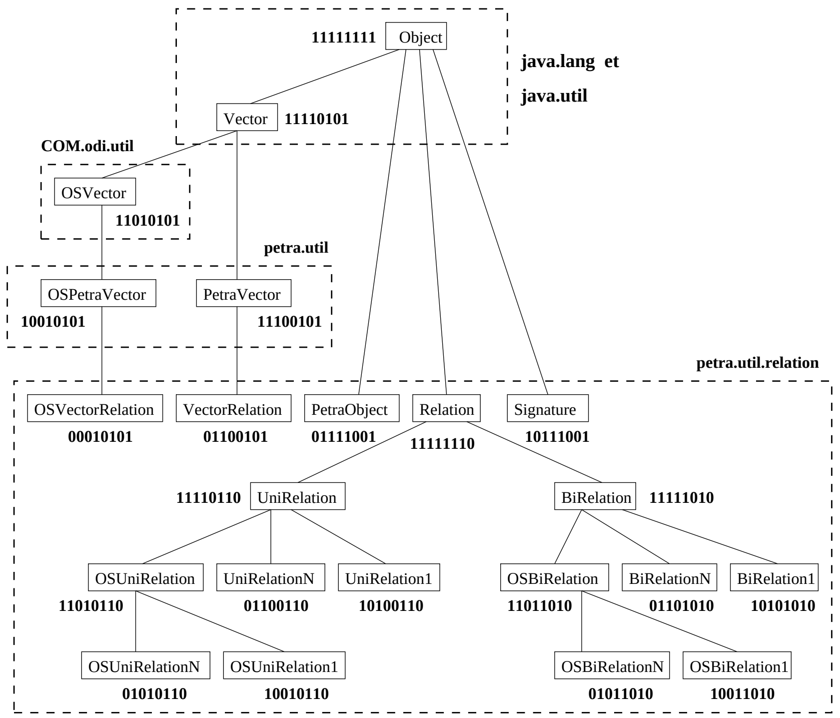


FIG. 1 – Un exemple de codage d'un ordre

conservant assez d'informations pour répondre rapidement aux opérations sur l'ordre. A cette compression en espace et en temps, s'ajoutent d'autres caratéristiques revêtant plus ou moins d'importance suivant les contextes d'utilisation des codages : par exemple le nombre d'instructions des algorithmes, le temps de génération des codages, les mises à jour d'un codage en cas de modification de l'ordre.

Les buts de la synthèse que nous proposons étaient multiples : rassembler des travaux épars dans la littérature, se fixer un cadre commun d'analyse, essayer d'identifier des catégories de méthodes, mettre en perspective les performances de ces méthodes pour essayer de les comparer. Il s'agissait de mieux situer les limites des méthodes proposées dans la littérature et de mieux comprendre les ingrédients qui font qu'une méthode semble plus efficace. Dans la littérature, les auteurs utilisent principalement la simulation ou des jeux de tests pour comparer leur méthode à celles des autres.

Au final, nous proposons une classification originale basée sur plusieurs catégories de méthodes qui sont ressorties de la lecture des différents articles sur le sujet. Plusieurs critères sont mis en avant pour évaluer les qualités et les défauts des méthodes de codage d'ordres, même si la comparaison des méthodes entre elles reste un problème délicat. Cette synthèse n'est pas exhaustive et laisse de nombreux problèmes ouverts, mais nous espérons qu'elle pourra s'avérer utile et peut être conduire à de futures recherches sur le sujet.

Au cours de cette thèse, j'ai eu l'occasion de travailler plus particulièrement sur deux paramètres classiques des ordres : la ***dimension*** et la ***2-dimension***. Chacun de ces paramètres correspond à une méthode de codage d'ordres. La ***dimension*** d'un ordre est le plus petit nombre d'extensions linéaires dont l'intersection redonne l'ordre initial. Il est connu que le problème du calcul de la dimension est un problème d'optimisation $\mathcal{NP}$-complet et de nombreux problèmes d'algorithmique sur ce paramètre sont encore ouverts [Spi94]. Avec Lhouari Nourine au LIRMM, nous avons travaillé sur des versions affaiblies du calcul de la dimension. Nous avons redémontré un résultat concernant la décroissance de la dimension sur une suite d'extensions d'un ordre, avec une technique différente de celle de la preuve originale par Pouzet, Reuter, Rival et Zaguia [NT00]. Avec Michel Habib et Lhouari Nourine, nous nous sommes aussi intéressés au problème de complétion pour les ordres de dimension 3 qui consiste à vérifier s'il est possible de compléter une extension linéaire fixée avec deux autres extensions linéaires telles que l'intersection des trois redonne l'ordre initial. Sur ce problème difficile, nous avons établi une conjecture qui suggère qu'il s'agit d'un problème polynomial. Nous n'avons pu pour l'instant ni prouver, ni infirmer cette conjecture.

D'autre part avec Michel Habib, Lhouari Nourine et Olivier Raynaud, nous nous sommes aussi intéressés à la ***2-dimension*** des ordres, paramètre issu de la mèthode de codage appelée ***représentation par ensembles*** ou ***codage par vecteur de bits***. C'est la méthode utilisée pour la figure 1. Cette méthode consiste à plonger l'ordre initial dans un treillis booléen. Pour traiter les opérations sur l'ordre, on est ainsi ramené à travailler sur un treillis booléen. Or les opérations sur les treillis booléens se prêtent bien au traitement informatique. La 2-dimension d'un ordre est la taille du plus petit treillis booléen dans lequel on peut plonger cet ordre. Générer de tels codages est un problème difficile. En particulier le calcul de 2-dimension est connu comme un problème d'optimisation $\mathcal{NP}$-complet. Pourtant de nombreux auteurs ont proposé des heuristiques de génération, stimulés par les très bons taux de compression en espace obtenus.

Nous présentons dans le chapitre 5 une synthèse sur les codages par vecteurs de bits. Nous rappelons les principales propriétés de ces codages et les heuristiques les plus connues. Notre apport a consisté à préciser certaines bornes sur la 2-dimension et certains résultats de complexité : le calcul de la 2-dimension est non-approximable à une constante près, à moins que $\mathcal{P}$=$\mathcal{NP}$. Nous avons aussi proposé une heuristique presque optimale pour générer des codages par vecteurs de bits des arbres, ce qui a donné lieu à un article présenté à la conférence ECOOP'2001 [RT01]. Nous avons étendu cette heuristique à tous les ordres, dans l'espoir d'améliorer les performances des meilleures heuristiques connues à ce jour. Les expérimentations sont en cours.

Ce mémoire de thèse présente aussi de nouvelles propriètés concernant les extensions d'un ordre et les treillis distributifs.

Avec Philippe Baldy et Michel Morvan au LIAFA, nous avons généralisé une bijection établie par Bonnet et Pouzet entre les extensions linéaires d'un ordre et les chaines maximales de son treillis des idéaux. Nous l'avons étendue à l'ensemble de toutes les extensions d'un ordre. Cette nouvelle bijection a été présentée dans le journal Order [BMT99] et nous la décrivons au chapitre 6. Ce résultat renforce une fois de plus les liens entre un ordre et son treillis des idéaux, qui suggèrent que beaucoup de propriétés d'un ordre peuvent se lire sur son treillis des idéaux et inversement pour travailler sur un treillis distributif on peut se placer sur l'ordre dont il est le treillis des idéaux.

Nous présentons au chapitre 7 une nouvelle transformation des treillis appelée ***retournement***. A l'origine, cette transformation modélise le comportement de certains systèmes dynamiques lorsqu'on étudie les suites de transitions pouvant mener d'un état à un autre état ou lorsqu'on inverse certaines règles d'évolution. Le comportement de certains systèmes dynamiques pouvant se modéliser sous forme de treillis, nous nous concentrons ici sur les propriétés de cette transformation dans le cadre de la théorie de l'ordre. En essayant de caractériser les classes de treillis stables pour cette transformation, nous avons montré que les treillis distributifs se comportent bien. Ce travail est effectué en collaboration avec Nathalie Caspard, Michel Morvan et Eric Remila.

Le chapitre 8 du mémoire traite d'algorithmique des graphes. Deux des cas d'intéraction entre structures de données et algorithmes mentionnés en début d'introduction, y sont présents. Tout d'abord, avec Jean-Marc Lanlignel, Lhouari Nourine et Olivier Raynaud, nous avons étudié un processus d'élagage des sommets d'un graphe non orienté : il consiste à supprimer au fur et à mesure les sommets ayant un jumeau (un autre sommet avec les mêmes voisins) et les sommets pendants (avec un seul voisin) tant qu'il en reste. Le graphe réduit obtenu après élagage est unique à un isomorphisme près (c.à.d. indépendant des différents ordres possibles de suppression des sommets). L'objectif était de proposer un algorithme efficace pour calculer ce graphe réduit et si possible une séquence d'élagage. Nous avons proposé un algorithme de calcul d'une séquence d'élagage et du graphe réduit presque linéaire en temps et en espace. L'idée consiste à réinterpréter le problème en terme de famille d'ensembles pour avoir un algorithme basé sur le repérage d'ensembles égaux et modifiant la famille en supprimant des éléments ou des ensembles. Pour pouvoir effectuer ces opérations efficacement, une structure de données présentant de très bonnes bornes de complexité amortie a été proposée. Appelée ***arbre lexicographique***, c'est une variante des arbres digitaux de recherche (***tries*** en anglais) qui trouve là une utilisation originale pour représenter les graphes.

Ensuite, avec Olivier Cogis au LIRMM nous avons avons étudié des problèmes classiques d'optimisation sur les graphes distance-héréditaires (les graphes que l'élagage précédent réduit à un sommet isolé) [CT01]. Cette classe de graphes a été très étudiée dans la littérature et de nombreux problèmes ont déjà été résolus sur cette classe. Nous proposons de nouveaux algorithmes simples pour résoudre certains de ces problèmes. Cette fois, nous avons choisi de fixer la structure de données pour représenter ces graphes en ne prenant en entrée de nos algorithmes qu'une séquence d'élagage, autrement dit un mot. Nos algorithmes résolvent plusieurs problèmes d'optimisation classiques “à la volée” par une simple lecture de la séquence d'élagage.

En parallèle, j'ai pu poursuivre une collaboration avec Isabelle Gúerin-Lassous de l'équipe ReMap de l'INRIA, rencontrée au LIAFA, autour de la conception et l'analyse d'algorithmes pour le modèle CGM. Ce modèle de calcul parallèle à gros grain a fait l'objet d'une étude approfondie dans la thèse d'Isabelle Gúerin-Lassous [GL99]. Au départ, il s'agissait de mener des expériences de simulation sur un algorithme de traitement des composantes connexes d'un graphe en CGM, pour vérifier la prédictivité des performances. Une présentation courte des résultats a été faite à la conférence francophone RenPar'97 [BGLT97]. Confrontés au problème de la génération aléatoire de données pour la simulation, nous avons proposé plusieurs algorithmes simples de génération de permutations aléatoires en CGM. Ils ont été présentés à la conférence OPODIS'2000 [GLT00].

On suppose le lecteur familier avec les notions élémentaires de théorie des graphes ainsi qu'avec des notions de complexité. Néanmoins des rappels seront présentés dans le chapitre 1. Les titres de plusieurs livres de références dans ces domaines seront aussi donnés.

La partie I présente une synthèse sur les méthodes de codage d'ordres. Le chapitre 2 introduit la problématique et le cadre d'étude dans lequel nous nous plaçons. Le chapitre 3 liste les principales méthodes de codage d'ordres de la littérature, elles sont décrites sous forme de fiches techniques avec de nombreuses références. Le chapitre 4 clôt cette synthèse par un premier bilan et des perspectives.

La partie II s'intéresse plus précisément à l'une de ces méthodes de codage d'ordres : la méthode de *représentation par ensembles*, ou *représentation par vecteurs de bits*, qui est associée au paramètre appelé *2-dimension*. Une synthèse sur cette méthode est présentée dans le chapitre 5.

La partie III est consacrée aux extensions des ordres et aux treillis distributifs. Le chapitre 6 expose la généralisation d'une bijection bien connue entre extensions linéaires d'un ordre et chaines maximales de son treillis des idéaux. Puis le chapitre 7 présente une nouvelle opération sur les treillis dont nous commençons à étudier les propriétés.

La partie IV de ce mémoire s'intéresse à des problèmes d'algorithmique des graphes. Elle regroupe dans le chapitre 8 les travaux sur deux problèmes qui sont liés puisque dans les deux cas il s'agit de l'étude des séquences d'élagage de sommets d'un genre particulier. Cependant ces problèmes correspondent à deux approches distinctes : dans le premier cas, il s'agit de trouver une structure de données efficace pour construire des séquences d'élagage, dans l'autre cas on se fixe les séquences d'élagage comme structure de données pour représenter certains graphes (les graphes distance-héréditaires) et il s'agit de trouver des algorithmes exploitant ces séquences pour résoudre efficacement des problèmes d'optimisation. Nous développerons principalement cette dernière étude sur l'optimisation, le problème de construction de séquences d'élagage ayant été traité très en détail dans la thèse de Jean-Marc Lanlignel [Lan01].

L'annexe A présente un travail d'algorithmique dans le modèle parallèle CGM, nous y faisons figurer un article en anglais présentant des algorithmes de génération de permutations aléatoires en CGM.

Parmi les références citées dans le texte et en bibliographie, on trouvera des adresses de sites Web. Sources de travaux, de logiciels, d'illustrations sur les sujets traités ici, elles ont bien sûr le caractère éphémère propre à toute adresse Web.

Chapitre 1

Définitions et notations

Ce chapitre rappelle les définitions et notations classiques utilisées en théorie des graphes et des ordres, et que nous emploieront dans les chapitres suivants. Pour toutes ces définitions et d'autres encore, le lecteur pourra aussi se référer au livre de Davey et Priestley "Introduction to Lattices and Order" [DP91] pour les ensembles ordonnés et aux livres de Golumbic [Gol80] ou West [Wes01] pour les graphes.

En l'absence d'ambiguité, certains des indices utilisés dans les notations seront omis.

Relations binaires

Une ***relation binaire*** $\mathcal{R}$ sur un ensemble X est une partie du produit $X \times X$. Les notations adoptées pour dire que deux éléments de X sont en relation pour $\mathcal{R}$ seront $(x, y) \in \mathcal{R}$ ou bien $xy \in \mathcal{R}$ ou bien $x\mathcal{R}y$. Cette relation binaire est dite :

1. ***réflexive*** si $\forall\ x \in X,\ x\mathcal{R}x$.
2. ***irréflexive*** si $\forall\ x \in X,\ non(x\mathcal{R}x)$.
3. ***symétrique*** si $\forall\ x, y \in X,\ x\mathcal{R}y \Rightarrow y\mathcal{R}x$.
4. ***antisymétrique*** si $\forall\ x, y \in X,\ x\mathcal{R}y\ et\ y\mathcal{R}x \Rightarrow x = y$.
5. ***transitive*** si $\forall\ x, y, z \in X,\ x\mathcal{R}y\ et\ y\mathcal{R}z \Rightarrow x\mathcal{R}z$.

Graphes

Un ***graphe orienté*** $G = (V, E)$ est un ensemble V muni d'une relation binaire $E \subseteq V \times V$. Les éléments de V sont appelés les ***sommets*** du graphe et les couples $xy \in E$ sont appelés les ***arcs*** du graphe.

Un ***graphe non orienté*** $G = (V, E)$ (sous-entendu sans boucles) est un ensemble V muni d'une relation binaire $E \subseteq V \times V$ qui est irréflexive et symétrique. Autrement dit E correspond à un ensemble de paires d'éléments de V. Les éléments de V sont appelés ***sommets*** et les paires $xy \in E$ sont appelées ***arêtes***. Deux sommets x, y tels que $xy \in E$ sont dits ***adjacents*** ou ***voisins***.

Soit $G = (V, E)$ un graphe orienté (respectivement non orienté). Soient $x, y \in V$, un ***chemin de*** x ***vers*** y dans G (resp. une ***chaine de*** x ***vers*** y) est une séquence d'arcs de G (resp. arêtes) v_0v_1,v_1v_2,...,$v_{k-1}v_k$ tels que $v_0 = x$ et $v_k = y$. La ***longueur*** d'un chemin (resp. chaine) est son nombre d'arcs (resp. arêtes). Un chemin (resp. une chaine) est dit ***élémentaire*** si tous les sommets de ses arcs (resp. arêtes) sont distincts, sauf éventuellement le premier et le dernier sommet. Un ***circuit*** (resp. ***cycle***) de G est un chemin (resp. une chaine) élémentaire de longueur non nulle qui commence et finit par le même sommet.

La ***fermeture transitive*** d'un graphe orienté $G = (V, E)$ est le graphe $G_{tc} = (V, E_{tc})$ tel que $xy \in E_{tc}$ si et seulement si il existe un chemin de longueur non nulle de x vers y dans G. Une ***réduction transitive*** d'un graphe orienté $G = (V, E)$ est un graphe $G_{tr} = (V, E_{tr})$ tel que G et G_{tr} ont la même fermeture transitive et tel que E_{tr} est minimal pour l'inclusion (il n'y pas unicité de la réduction transitive dans le cas général).

Ordres

Un ***ensemble ordonné***, ou ***ordre partiel***, $P = (X, \leq_P)$ est un ensemble X muni d'une relation binaire $\leq_P$ sur X qui est réflexive, antisymétrique et transitive. L'***ordre strict*** $(X, <_P)$ lui correspondant est défini par : $x <_P y$ si et seulement si $x \leq_P y$ et $x \neq y$. Deux éléments $x, y \in X$, $x \neq y$, sont dits ***comparables*** (noté $x \sim_P y$) si $x \leq_P y$ ou $y \leq_P x$, sinon ils sont dits ***incomparables*** (noté $x \parallel_P y$).

On dit qu'un ordre $Q = (Y, \leq_Q)$ est un ***sous-ordre*** de P si et seulement si $Y \subseteq X$ et pour tout $x, y \in Y$, $x \leq_Q y$ si et seulement si $x \leq_P y$. On dit aussi que Q est l'***ordre induit*** par P sur Y. Soit Y une partie de X, on notera parfois $P|Y$ pour désigner l'ordre induit par P sur Y et $P \backslash Y$ pour désigner l'ordre induit par P sur $X \backslash Y$.

Le ***dual*** d'un ordre $P = (X, \leq_P)$ est l'ordre $P^d = (X, \leq_{P^d})$ vérifiant $x \leq_{P^d} y$ si et seulement si $y \leq_P x$. Pour passer de P à son dual, on a simplement renversé le sens de toutes les comparabilités.

Deux ordres $P = (X, \leq_P)$ et $Q = (Y, \leq_Q)$ sont dits ***isomorphes*** s'il existe une bijection ϕ de X sur Y telle que $\forall x, x' \in X$, $x \leq_P x'$ si et seulement si $\phi(x) \leq_Q \phi(x')$. Dans ce cas, on note $P \cong Q$.

On dit que P est fini si X est fini, dans ce cas on utilisera parfois la notation $|P|$ pour désigner $|X|$.

Liens avec les graphes orientés, transitivité

Soit $G = (V, E)$ un graphe orienté sans circuit, alors la fermeture transitive de G est un ordre strict sur V. Inversement tout ordre strict $P = (X, <_P)$ peut être vu comme le graphe orienté sans circuit $G_{tc} = (X, E)$ avec $E = \{(x, y)$ tel que $x <_P y\}$. Il admet une unique réduction transitive G_{tr} aussi appelée ***graphe orienté de couverture***. On dit ainsi pour deux éléments $x, y \in X$ que y ***couvre*** x (noté $x \prec_P y$) si $x <_P y$ et il n'existe pas d'élément $z \in X$ tel que $x <_P z <_P y$ (ces couples (x, y) sont les arcs de la réduction transitive).

Liens avec les graphes non orientés

A tout ordre $P = (X, \leq_P)$, on peut associer son ***graphe de comparabilité*** $G_{comp}(P) = (X, E)$ tel que pour tout $x, y \in X$, x et y sont adjacents si et seulement si ils sont comparables dans P (c'est un graphe non orienté).

Voisinage

Soit $P = (X, \leq_P)$ un ordre et $x \in X$. L'ensemble des ***prédécesseurs*** de x est défini par $Pred_P(x) = \{y \in X \mid y <_P x\}$. L'ensemble des ***successeurs*** de x est défini par $Succ_P(x) = \{y \in X \mid x <_P y\}$. L'ensemble des ***prédécesseurs immédiats*** de x est défini par $ImPred_P(x) = \{y \in X \mid y \prec_P x\}$. L'ensemble des ***successeurs immédiats*** de x est défini par $ImSucc_P(x) = \{y \in X \mid x \prec_P y\}$. Un élément $x \in X$ est dit ***minimal*** (resp. ***maximal***) si $Pred_P(x) = \emptyset$ (resp. $Succ_P(x) = \emptyset$). L'ensemble des éléments minimaux (resp. maximaux) de P est noté $Min(P)$ (resp. $Max(P)$). L'ordre P admet un élément ***minimum*** (resp. ***maximum***) x si $\forall y \in X$, $x \leq_P y$ (resp. $y \leq_P x$). Par extension, on définit les élément minimaux, maximaux, minimum, maximum d'une partie S de X à partir de l'ordre induit par P sur S.

Un ***idéal*** d'un ordre $P = (X, \leq_P)$ est une partie I de X telle que $\forall x, y \in X$, si $x \in I$ et $y \leq_P x$ alors $y \in I$. Un idéal I est dit ***principal*** s'il existe un élément $a \in X$ tel que $I = \{x \in X \mid x \leq_P a\}$, on note alors $I = \downarrow_P a$. De manière duale, un ***filtre*** d'un ordre $P = (X, \leq_P)$ est une partie F de X telle que $\forall x, y \in X$, si $x \in F$ et $y \geq_P x$ alors $y \in F$. Un filtre F est dit ***principal***

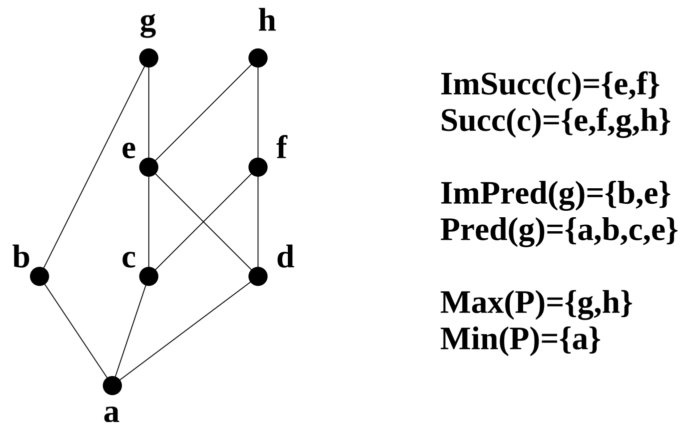


FIG. 1.1 – Diagramme de Hasse d'un ordre P (qui admet un minimum)

s'il existe un élément $a \in X$ tel que $F = \{x \in X \mid x \geq_P a\}$, on note alors $F = \uparrow_P a$.

Dessiner un ordre, diagramme de Hasse

La représentation usuelle d'un ordre P consiste à dessiner le graphe orienté de couverture G_{tr} de P avec une orientation du bas vers le haut : on place les sommets de telle manière que si $x \prec_P y$ alors x est nécessairement en dessous de y, autrement tous les arcs sont orientés du bas vers le haut (c'est possible car G_{tr} est sans circuit). Avec cette convention, on peut donc omettre de dessiner l'orientation des arcs et les remplacer par des arêtes. Par la suite, par défaut c'est cette convention ***du bas vers le haut*** qui sera conservée (mais dans la littérature on trouve quelques fois des ensembles ordonnés dessinés du haut vers le bas, c'est à dire avec les éléments les plus petits en haut).

Parmi ce type de représentations, l'une d'elle est couramment utilisée : il s'agit du diagramme de Hasse. Le ***diagramme de Hasse*** d'un ordre P est construit de la manière suivante : on place tous les sommets correspondants aux éléments de $Min(P)$ sur un même niveau, puis on place ceux de $Min(P \backslash Min(P))$ sur un niveau juste au dessus et ainsi de suite, sur chaque nouveau niveau on récupère les éléments minimaux parmi ceux qui restent. Pour finir, on place les arêtes qui joignent les éléments comparables.

La figure 1.1 est le diagramme de Hasse d'un ordre P qui permet d'illustrer les différents définitions données jusqu'à présent.

Ordre total, chaines, antichaines

Un ordre ***total*** ou ***chaine*** $P = (X, \leq_P)$ est un ordre tel que $\forall x, y \in X$, $x \leq_P y$ ou $y \leq_P x$. Soit $P = (X, \leq_P)$ un ordre quelconque. Une partie C de X telle que l'ordre induit par P sur C est total est appelée une ***chaine*** de P. Dans le cas fini, la ***hauteur*** d'une chaine C est l'entier $h(C) = |C| - 1$. Par extension, on définit la ***hauteur*** de l'ordre P par $h(P) = \max\{h(C) \mid C \text{ est une chaine de } P\}$. La ***hauteur*** d'un élément $x \in X$ dans P, notée $h(P, x)$, est la hauteur de l'ordre induit sur $\downarrow_P x$.

Par exemple, l'ordre P de la figure 1.1 est de hauteur 3, une chaine de hauteur 3 est par exemple $a <_P c <_P e <_P h$.

Un ordre $P = (X, \leq_P)$ est une ***antichaine*** si tous ses éléments sont incomparables. De même, une ***antichaine*** A d'un ordre $P = (X, \leq_P)$ est une partie de X tels que tous ses éléments sont incomparables, autrement dit $\forall x, y \in A$, $x \parallel_P y$. La ***largeur*** d'une antichaine A est l'entier $w(A) = |A|$. Par extension, la ***largeur*** d'un ordre P est définie comme $w(P) = \max\{w(A) \mid A \text{ est une antichaine de } P\}$.

Par exemple, la largeur de l'ordre P de la figure 1.1 vaut 3, l'ensemble $A = \{b, e, f\}$ est une antichaine de P.

On utilisera les notations $\mathcal{C}_n$ et $\mathcal{A}_n$ pour désigner respectivement un ordre total à n éléments (donc de hauteur $n - 1$) et une antichaine à n éléments.

On notera $\underline{1}$ pour désigner un ordre réduit à un seul élément.

Extensions et réductions

Soient $P = (X, \leq_P)$ et $Q = (Y, \leq_Q)$ deux ordres. On dit que Q est une ***extension*** de P si $X = Y$ et $\forall x, y \in X$, $x \leq_P y$ implique $x \leq_Q y$. Si de plus Q est un ordre total, on dit que Q est une ***extension linéaire*** de P. On dit que Q est une ***réduction*** de P si $X = Y$ et $\forall x, y \in X$, $x \leq_Q y$ implique $x \leq_P y$.

Par exemple, l'ordre total L défini sur $\{a, b, c, d, e, f, g, h\}$ par $a <_L b <_L c <_L d <_L e <_L f <_L g <_L h$ est une extension linéaire de l'ordre P de la figure 1.1.

Classes d'ordres

Certaines classes d'ordres reviennent souvent et ont été plus particulièrement étudiées. C'est par exemple le cas des ***arbres*** et des ***ordres d'intervalles***. Pour d'autres classes d'ordres, le lecteur pourra consulter une synthèse par Möhring [Moh89].

Dans le contexte des ensembles ordonnés, nous appelons ***arbre*** un ensemble ordonné $T = (X, \leq_T)$ non vide, possédant un minimum $\perp$ et tel que $\forall x \in X, x \neq \perp, |ImPred(x)| = 1$ (la terminologie exacte parle d'***arborescence enracinée***). Si on retire la condition qu'il existe un minimum, on parle de ***forêt***. Un ***sous-arbre*** d'un arbre T est un sous-ordre de T qui est aussi un arbre. De même, on peut définir la notion de ***sous-forêt*** de T. Remarquer qu'on passe naturellement d'une forêt à un arbre en ajoutant un minimum et inversement. Le dual d'un arbre est appelé un ***anti-arbre***.

Dans un arbre $T = (X, \leq_T)$, le minimum $\perp$ est aussi appelé la ***racine*** de l'arbre. Soit $x, y \in X$, on dit que x est le ***père*** de y si $\{x\} = ImPred(y)$ et dans ce cas y est un ***fils*** de x. Les ***feuilles*** de T sont les éléments $x \in X$ tels que $ImSucc(x) = \emptyset$, les autres éléments sont appelés ***noeuds***. Le ***degré*** d'un élément $x \in X$, noté $deg(x)$, est le nombre de fils de x, c'est à dire $|ImSucc(x)|$. Le ***degré*** de l'arbre T, noté $\Delta(T)$, est le maximum des degrés de ses éléments.

Un ***ordre d'intervalles*** $P = (X, \leq_P)$ est un ordre tel que chaque élément $x \in X$ est associé à un intervalle $I_x = [g_x, d_x]$ d'entiers et tel que l'ordre strict associé à P vérifie pour tout $x, y \in X$, $x <_P y$ si et seulement si $d_x \leq g_y$ (c'est-à-dire I_x est entièrement à gauche de I_y). Il existe de nombreuses caratérisations de ces ordres [Moh89].

Treillis

Soit $P = (X, \leq_P)$ un ensemble ordonné et $x, y \in X$. Soit $Majorant(x, y) = \{z \in X \mid x \leq_P z \text{ et } y \leq_P z\} = \uparrow_P x \cap \uparrow_P y$ l'ensemble des majorants de x et y. Si $Majorant(x, y)$ a un minimum, alors on dit que ce minimum est la ***borne supérieure*** de x et y. Elle est notée $x \vee_P y$ ou $Sup_P(x, y)$. Autrement dit si elle existe, la borne supérieure de x et y vérifie :

1. $x \leq_P x \vee_P y$ et $y \leq_P x \vee_P y$,
2. $\forall z \in X$, si $x \leq_P z$ et $y \leq_P z$ alors $x \vee_P y \leq_P z$.

La figure 1.2 schématise cette configuration.

De manière duale, on définit la ***borne inférieure*** de x et y comme le maximum de l'ensemble $Minorant(x, y) = \{z \in X \mid z \leq_P x \text{ et } z \leq_P y\}$ s'il existe. Elle est notée $x \wedge_P y$ ou $Inf_P(x, y)$. On peut aussi définir la borne supérieure $\bigvee_P S$ et la borne inférieure $\bigwedge_P S$ d'une partie S de X comme le minimum de $\{z \in X \mid \forall x \in S,\ x \leq_P z\}$ et le maximum de $\{z \in X \mid \forall x \in S,\ z \leq_P x\}$ s'ils existent.

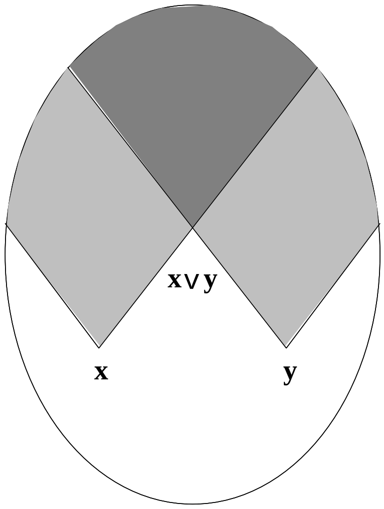


FIG. 1.2 – Borne supérieure $x \vee y$ de x et y dans un ordre P

Un ordre T est appelé ***treillis*** si pour tout couple d'éléments les bornes supérieure et inférieure existent dans T. Un ordre T est appelé ***treillis complet*** si pour tout ensemble d'éléments les bornes supérieure et inférieure existent dans T. Il est clair que tout treillis fini est complet. Dans un treillis complet $T = (X, \leq_T)$, il existe un minimum $\bigwedge_T X$ aussi noté $\perp$ et un maximum $\bigvee_T X$ aussi noté $\top$. Notons que le dual d'un treillis est aussi un treillis

Un ***sous-treillis*** T' d'un treillis T est un sous-ordre de T tel que T' est un treillis et pour tout x et y éléments de T', alors $x \vee_T y \in T'$ et $x \wedge_T y \in T'$.

Exemples de treillis

Nous donnons ici quatre exemples classiques de treillis complets. D'abord tout ordre total fini est un treillis complet. Ensuite soit $n \geq 0$ et X un ensemble à n éléments, on note $\mathcal{P}(X)$ l'ensemble des 2^n parties de X. Alors $\mathcal{B}(X) = (\mathcal{P}(X), \subseteq)$ est un treillis appelé ***treillis booléen*** sur X, on le note aussi $\underline{2}^X$. Puisque tous les treillis booléens sur n éléments sont isomorphes, on peut se ramener au cas où $X = \{1, 2, \ldots, n\}$ et on note aussi ce treillis $\mathcal{B}_n$ ou $\underline{2}^n$. Les treillis booléens sur n éléments sont aussi appelés ***hypercube de dimension*** n.

Soit $P = (X, \leq_P)$ un ordre. L'ensemble des idéaux de P noté $I(P)$ muni de la relation d'inclusion est un treillis complet : l'opération borne supérieure

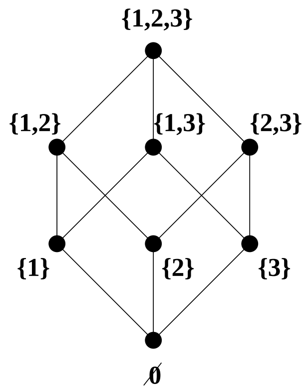


FIG. 1.3 – Hypercube (treillis booléen) de dimension 3

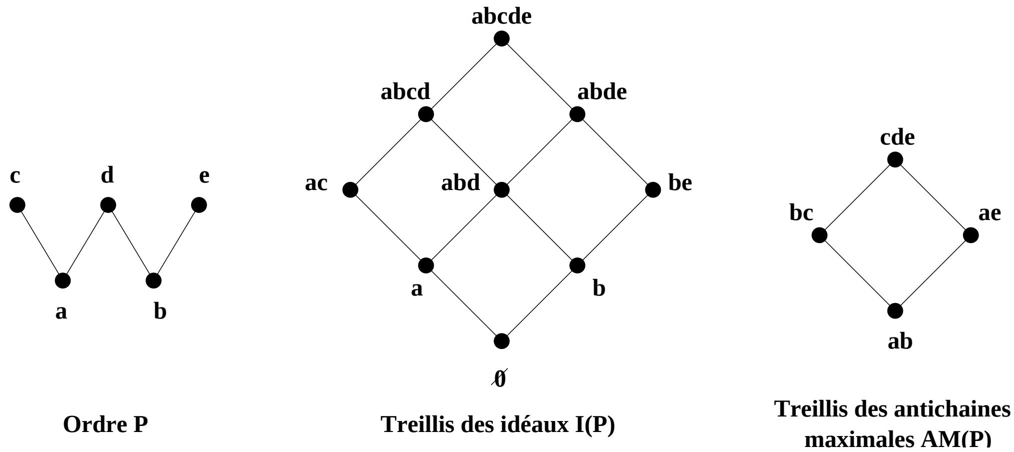


FIG. 1.4 – Treillis des idéaux $I(P)$ et des antichaines maximales $AM(P)$ d'un ordre P

correspond à l'union et borne inférieure à l'intersection. C'est le ***treillis des idéaux*** de P. On définit aussi $A(P)$ l'ensemble des antichaines de P et $AM(P)$ l'ensemble des antichaines de P maximales pour l'inclusion. Soit A et B deux antichaines de P, on peut définir la relation $A \leq_{anti} B$ si $\forall x \in A$, $\exists y \in B$, $x \leq_P y$. L'ensemble $A(P)$ muni de la relation $\leq_{anti}$ est un treillis complet appelé le ***treillis des antichaines*** de P. Il est isomorphe au treillis des idéaux de P. L'ensemble $AM(P)$ muni de la relation $\leq_{anti}$ est aussi un treillis complet appelé le ***treillis des antichaines maximales*** de P.

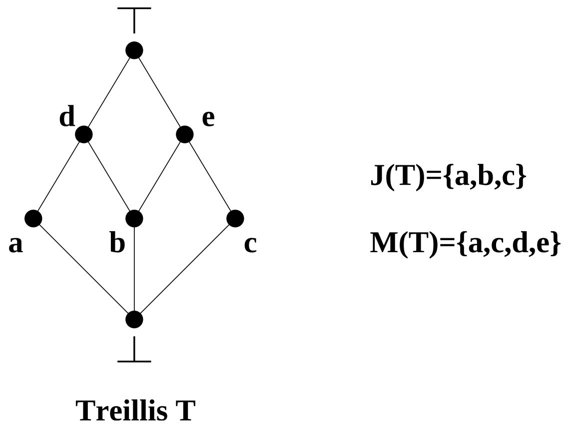


FIG. 1.5 – Les inf-irréductibles et sup-irréductibles d'un treillis T

Eléments irréductibles

Certains éléments d'un treillis jouent un rôle très important : ils permettent de décrire complètement la structure du treillis, de le reconstruire à l'aide des opérations $\wedge$ et $\vee$, ce sont les éléments inf-irréductibles et sup-irréductibles. Soit $T = (X, \leq_T)$ un treillis complet. Un élément $j \in X$ est appelé ***sup-irréductible*** si $j \neq \bot$ et $\forall S \subseteq X$, $j = \bigvee_P S$ implique $j \in S$. Si T est fini, c'est équivalent au fait que $|ImPred(j)| = 1$. De manière duale, un élément $m \in X$ est appelé ***inf-irréductible*** si $m \neq \top$ et $\forall S \subseteq X$, $m = \bigwedge_P S$ implique $m \in S$. Si P est fini, c'est équivalent au fait que $|ImSucc(m)| = 1$. L'ensemble des sup-irréductibles de T est noté $J(T)$ et l'ensemble des inf-irréductibles de T est noté $M(T)$.

Les éléments sup-irréductibles (resp. inf-irréductibles) d'un treillis complet T forment une ***base génératrice*** de ce treillis pour l'opération $\bigvee_T$ (resp. $\bigwedge_T$) : tout élément de T est borne supérieure (resp. inférieure) d'une partie de $J(T)$ (resp. $M(T)$) et aucun des éléments de $J(T)$ (resp. $M(T)$) n'est borne supérieure d'autres éléments de $J(T)$ (resp. borne inférieure d'autres éléments de $M(T)$). Le lemme suivant précise ce résultat. On définit les ensembles $J_x(T) = \{j \in J(T) \mid j \leq_T x\}$ et $M_x(T) = \{m \in M(T) \mid x \leq_T m\}$ (notés J_x et M_x s'il n'y a pas d'ambiguité).

Lemme 1.1. *Soit T un treillis complet et x un élément de T. Alors on a les deux égalités suivantes :*

$$x = \bigvee J_x \quad \textit{et} \quad x = \bigwedge M_x$$

En quelque sorte, les ensembles J_x et M_x permettent chacun de représenter l'élément x. Du lemme 1.1, on déduit les résultats suivants.

Lemme 1.2. *Soit T un treillis complet, soient x et y deux éléments de T. Alors on a les équivalences suivantes :*

$$x \leq_T y \iff J_x \subseteq J_y$$

$$x \leq_T y \iff M_x \supseteq M_y$$

Ordres bipartis

Un ***ordre biparti*** (ou ***biparti***) $B = (X, Y, <_B)$ est un ordre défini sur l'union disjointe de X et Y tel que sa relation d'ordre strict $<_B$ vérifie : $x <_B y$ implique $x \in X$ et $y \in Y$. Le diagramme de Hasse d'un tel ordre correspond à un graphe biparti.

A partir d'un ordre P donné, on peut construire plusieurs ordres bipartis qui jouent un rôle important en théorie des ensembles ordonnés. Pour plus de détails sur ces bipartis et leur rôle, voir par exemple [Nou93]. ou [MN96]. En particulier, pour un ordre $P = (X, \leq_P)$, on peut définir les bipartis $B(P) = (X, X, \leq_P)$, $\overline{B}(P) = (X, X, \not\leq_P)$, $B^0(P) = (X, X, <_P)$ ou $\overline{B^0}(P) = (X, X, \not<_P)$. Pour un treillis $T = (X, \leq_T)$, on peut définir le biparti $Bip(T) = (J(T), M(T), \not\leq_T)$. Le théorème suivant est un exemple de lien entre structures ordonnées et bipartis.

Théorème 1.3 (Reuter [Reu91]). *Soit $T = (X, \leq_T)$ un treillis complet. Alors T est isomorphe à $AM(Bip(T))$.*

Un théorème du même genre, établi par Behrendt [Beh88], prouve que tout treillis T est isomorphe à $AM(\overline{B}(P))$.

Un ordre biparti peut se voir naturellement comme un graphe biparti (en considérant en fait le graphe de comparabilité). Pour plus de commodité, on laissera le contexte fixer si l'on se place du point de vue ordre ou graphe. Par exemple si B est un ordre biparti, si l'on parle de $AM(B)$ c'est que l'on considère l'ordre B, alors que si l'on parle d'un recouvrement des arêtes de B c'est que l'on considère le graphe B.

On dit d'un biparti $B = (X, Y, <_B)$ qu'il est ***complet*** si pour tout $x \in X$ et $y \in Y$, $x <_P y$.

Classes de treillis

Nous donnons ici quelques propriétés classifiant les treillis. Un treillis fini $T = (X, \leq_T)$ est dit ***gradué*** si toutes les chaines maximales (pour l'inclusion)

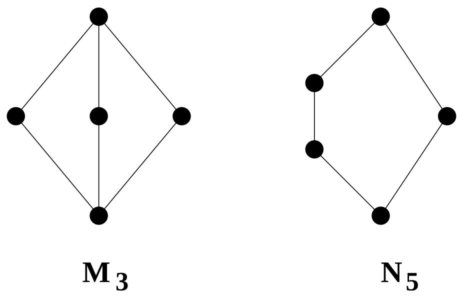


FIG. 1.6 – Les treillis M_3 et N_5

ont la même hauteur. Un treillis $T = (X, \leq_T)$ est dit ***distributif*** s'il satisfait la loi de distributivité :

$$\forall\ x, y, z \in X,\ \ x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

Ces treillis ont été très étudiés (voir par exemple [Bir67, DP91, Nou93, Nou00, HN93, HN96, Mar92, MN96]) et de nombreuses caractérisations en ont été données comme les suivantes.

Théorème 1.4. *Soit $T = (X, \leq_T)$ un treillis fini. Les propositions suivantes sont équivalentes :*

1. *T est distributif.*
2. *T ne contient pas de sous-treillis isomorphe à M_3 ou N_5 (voir figure 1.6).*
3. *T est le treillis des idéaux $I(P)$ d'un ordre P (dans ce cas P est isomorphe à l'ordre induit sur $J(T)$ lui-même isomorphe au dual de l'ordre induit sur $M(T)$).*
4. *T est gradué et $|J(T)| = |M(T)| = h(T)$.*

La proposition 3 du théorème est un résultat de Birkhoff [Bir67] et est parfois appelée théorème de représentation de Birkhoff.

Un treillis $T = (X, \leq_T)$ est $\wedge - extremal$ (resp. $\vee - extremal$) si $h(T) = |M(T)|$ (resp. $h(T) = |J(T)|$). Introduits par Markowski [Mar92], Morvan et Nourine ont montré que ces treillis possèdent la propriété du schéma d'élimination simplicial [MN96] donnant ainsi une nouvelle caractérisation et un algorithme de reconnaissance glouton.

Constructions

Soient $P_1 = (X_1, \leq_{P_1}), \ldots, P_n = (X_n, \leq_{P_n})$ des ensembles ordonnés.

Le ***produit cartésien*** ou ***produit direct*** $\Pi = P_1 \times \ldots \times P_n$ est l'ordre partiel $\Pi = (X_1 \times \ldots \times X_n, \leq_\Pi)$ tel que $(x_1, \ldots, x_n) \leq_\Pi (y_1, \ldots, y_n)$ si et seulement si $\forall i,\ 1 \leq i \leq n,\ x_i \leq_{P_i} y_i$. Soit P un ensemble ordonné, on notera P^n le produit cartésien $P \times \ldots \times P$ n fois.

Soient $P = (X, \leq_P)$ et $Q = (Y, \leq_Q)$ deux ensembles ordonnés.

L'***ordre lexicographique*** $P \times_{lexico} Q$ est défini sur $X \times Y$ par la relation $\leq_{lexico}$ où $(x_1, y_1) \leq_{lexico} (x_2, y_2)$ si $x_1 \leq_P x_2$ ou $(x_1 = x_2$ et $y_1 \leq_Q y_2)$.

L'***opération série*** associe à P et Q l'ordre $P \oplus Q$ défini sur l'union disjointe $X \cup Y$ par la relation $\leq_{P \oplus Q}$ où $u \leq_{P \oplus Q} v$ si $(u, v \in X$ et $u \leq_P v)$ ou $(u, v \in Y$ et $u \leq_Q v)$ ou $(u \in X$ et $v \in Y)$.

L'***opération parallèle*** associe à P et Q l'ordre $P \uplus Q$ défini sur l'union disjointe $X \cup Y$ par la relation $\leq_{P \uplus Q}$ où $u \leq_{P \uplus Q} v$ si $(u, v \in X$ et $u \leq_P v)$ ou $(u, v \in Y$ et $u \leq_Q v)$.

Si $X = Y$, l'***opération intersection*** associe à P et Q l'ordre $P \cap Q$ défini sur X par la relation $\leq_{P \cap Q}$ où $u \leq_{P \cap Q} v$ si $u \leq_P v$ et $u \leq_Q v$.

Si $X = Y$, l'***opération union*** associe à P et Q la relation bianire $P \cup Q$ définie sur X par la relation $\leq_{P \cup Q}$ où $u \leq_{P \cup Q} v$ si $u \leq_P v$ ou $u \leq_Q v$ (noter qu'il ne s'agit pas toujours d'une relation antisynétrique ou transitive).

La figure 1.7 illustre ces constructions.

Substitution et Décomposition modulaire

On définit maintenant l'***opération de substitution*** qui associe à un ordre $P = (X, \leq_P)$ avec n éléments $x_1, \ldots, x_n$ et n ordres $Q_1 = (X_1, \leq_{Q_1})$, ... , $Q_n = (X_n, \leq_{Q_n})$ le nouvel ordre $P[Q_1, \ldots, Q_n]$ défini sur l'union disjointe $\bigcup_{1 \leq i \leq n} X_i$ par la relation $\leq_{mod}$:

1. si $u \in X_i$ et $v \in X_j$ avec $i \neq j$, alors $u \leq_{mod} v$ si et seulement si $x_i \leq_P x_j$.
2. si $u, v \in X_i$, alors $u \leq_{mod} v$ si et seulement si $u \leq_{Q_i} v$.

Inversement on peut décomposer tout ordre $P = (X, \leq_P)$ en partionnant X en k parties X_1,...,X_k (appelées “modules forts”), en notant Q_1,...,Q_k les ordres induits, telles qu'il existe un ordre Q qui redonne P par substitution, c'est à dire $P \equiv Q[Q_1, \ldots, Q_k]$, et qui est soit une antichaine (décomposition ***parallèle***), soit une chaine (décomposition ***série***), soit un ordre premier c'est à dire indécomposable de manière non triviale (décomposition ***premier***). Avec les bonnes contraintes, on peut imposer que cette décomposition soit

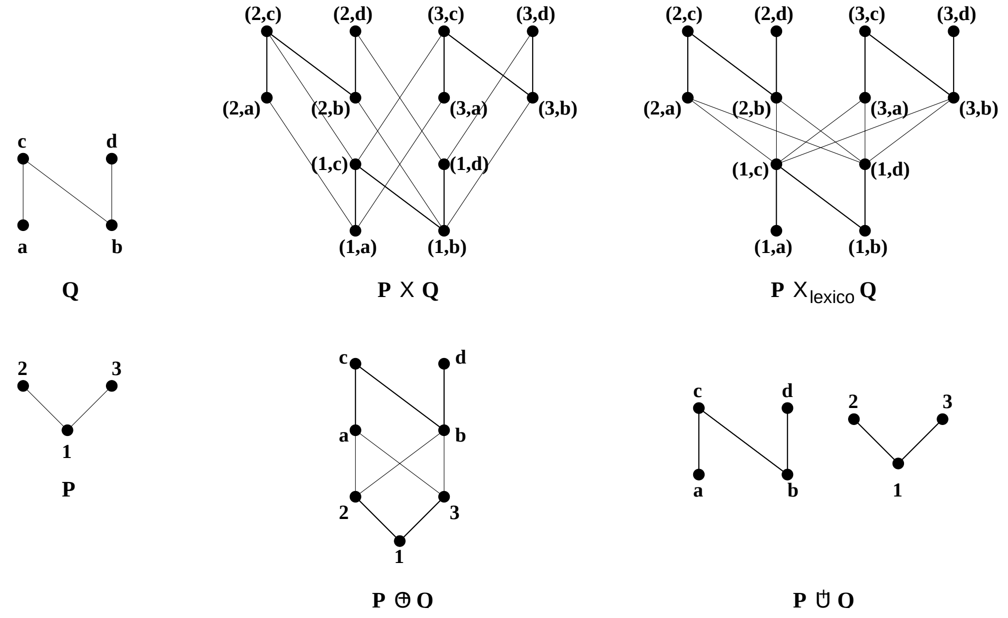


FIG. 1.7 – Exemples de constructions à partir des ordres P et Q

unique et en l'appliquant récursivement sur les modules, on construit un arbre de décomposition. Pour plus détails et notamment les aspects algorithmiques de cette décomposition appelée ***décomposition modulaire***, le lecteur pourra par exemple consulter la thèse de Christian Capelle [Cap96] ou celle de Christophe Paul [Pau98].

Morphismes

On donne maintenant les notions de morphisme pour les ordres, puis pour les treillis.

Définition 1.5. *Soient $P = (X, \leq_P)$ et $Q = (Y, \leq_Q)$ deux ordres. Alors une application $\phi : X \to Y$ est appelée :*

1. ***croissante ou morphisme d'ordre*** *si*

$$\forall x, y \in X,\ x \leq_P y \Rightarrow \phi(x) \leq_Q \phi(y).$$

2. ***plongement ou codage d'ordre*** *si*

$$\forall x, y \in X,\ x \leq_P y \iff \phi(x) \leq_Q \phi(y).$$

3. ***isomorphisme d'ordre*** *si ϕ est un plongement bijectif.*

Remarque 1.6. *Un plongement est toujours injectif. Par contre, une application croissante injective n'est pas toujours un plongement (par exemple toute application injective d'une antichaine à n éléments dans une chaine à n éléments est croissante mais n'est pas un plongement). Notons $\phi(P) = \{y \in Y \mid \exists x \in X,\ \phi(x) = y\}$ l'image de P. Si ϕ est un plongement, alors ϕ est un isomorphisme d'ordre entre P et $\phi(P)$. Un plongement d'ordre est un isomorphisme d'ordre si et seulement si il est surjectif.*

Définition 1.7. *Soient $T = (X, \leq_T)$ et $L = (Y, \leq_L)$ deux treillis. alors une application $\phi : X \to Y$ est appelée :*

1. ***$\vee$-morphisme ou morphisme de sup-demi-treillis*** *si*
$$\forall x, y \in X,\ \phi(x \vee_P y) = \phi(x) \vee_Q \phi(y).$$
2. ***$\wedge$-morphisme ou morphisme de inf-demi-treillis*** *si*
$$\forall x, y \in X,\ \phi(x \wedge_P y) = \phi(x) \wedge_Q \phi(y).$$
3. ***homomorphisme de treillis*** *si ϕ est à la fois un $\vee$-morphisme et un $\wedge$-morphisme.*
4. ***$\vee$-plongement*** *si ϕ est à la fois un $\vee$-morphisme et un plongement d'ordre.*
5. ***$\wedge$-plongement*** *si ϕ est à la fois un $\wedge$-morphisme et un plongement d'ordre.*
6. ***plongement ou codage de treillis*** *si ϕ est à la fois un morphisme de treillis et un plongement d'ordre.*

Remarque 1.8. *Les notions de morphisme de treillis et de plongement d'ordre sont a priori indépendantes (toute application constante est un morphisme de treillis). Mais dans le cas des treillis, l'injectivité de ϕ suffit à assurer la condition de plongement : ϕ est un plongement de treillis si et seulement si ϕ est un morphisme de treillis injectif.*

*On n'a pas défini la notion d'***isomorphisme de treillis***. On aurait pu le définir comme un morphisme de treillis bijectif. On vérifie alors que ϕ est un isomorphisme de treillis si et seulement si ϕ est un isomorphisme d'ordre.*

Ordres de grandeurs

Pour finir cette section d'introduction, on rappelle le sens des notations $\mathcal{O}$, Ω, Θ utilisées pour donner des ordres de grandeurs de fonctions. Soient f et g deux fonctions définies sur les entiers,

- $f(n) = \mathcal{O}(g(n))$ si $\exists\ c, n_0,\ \forall n \geq n_0,\ f(n) \leq cg(n)$.
- $f(n) = \Omega(g(n))$ si $\exists\ c, n_0,\ \forall n \geq n_0,\ f(n) \geq cg(n)$.
- $f(n) = \Theta(g(n))$ si $\exists\ c_1, c_2, n_0,\ \forall n \geq n_0,\ c_1 g(n) \leq f(n) \leq c_2 g(n)$.

Complexités

Autant que possible, tous les algorithmes sont donnés avec leurs complexités en espace et en temps dans différents contextes : pire des cas, complexité amortie, complexité en moyenne (pour une distribution des données en entrée). Par défaut, on donnera les complexités dans le pire des cas et si on omet la complexité en espace, c'est qu'elle est en $\mathcal{O}(\max(entree, sortie))$, où *entree* est la taille des données en entrée et *sortie* est la taille des données en sortie.

Nous seront aussi amenés à étudier des problèmes d'optimisation, c'est pourquoi nous faisons maintenant quelques rappels sur les problèmes $\mathcal{NP}$-complets, les problèmes d'approximation et les transformations pour passer d'un problème à un autre (aussi appelées réductions). Pour des ouvrages généraux sur la complexité, le lecteur est renvoyé aux livres de Papadimitriou [Pap94], Ausiello et al [ACG+99] ou Lassaigne et de Rougemont [LdR96] ainsi qu'au livre de Garey et Johnson [GJ79] qui reste une référence sur la $\mathcal{NP}$-complétude.

Les définitions suivantes ont été en partie empruntées au livre de Lassaigne et de Rougemont “Logique et Complexité” [LdR96] et au livre “Complexity and Approximation” [ACG+99]. Une liste de résultats tirés de ce livre est consultable sur le Web à l'adresse

http ://www.nada.kth.se/∼viggo/wwwcompendium/.

Problèmes d'optimisation et ratios d'approximation

Définition 1.9. *Un* ***problème d'optimisation*** *est défini comme un triplet* (Sol, val, but) *tel que :*

1. *Pour chaque instance* x, $Sol(x)$ *est l'ensemble des solutions possibles.*
2. *La fonction val associe à chaque solution possible une valeur entière.*
3. *Il s'agit soit de maximiser (*$but = \max$*) soit de minimiser (*$but = \min$*).*

Un ***optimum*** *du problème d'optimisation est un élément* $y^* \in Sol(x)$ *tel que* $val(y^*) = \max\{val(y) \mid y \in Sol(x)\}$ *si* $but = \max$*, respectivement* $val(y^*) = \min\{val(y) \mid y \in Sol(x)\}$ *si* $but = \min$*. La valeur de la fonction val pour l'optimum est notée* $opt(x)$*.*

Pour la suite, on utilise la notation $|x|$ pour désigner la taille de la donnée x.

Définition 1.10. *Un problème d'optimisation est dans la classe $\mathcal{NPO}$ si :*

1. *Il existe un polynôme p tel que pour tout $y \in Sol(x)$, $|y| \leq p(|x|)$.*
2. *Vérifier que $y \in Sol(x)$ est décidable en temps polynomial.*
3. *La fonction val est calculable en temps polynomial.*

Définition 1.11. *Pour un problème de minimisation et une instance x, le **ratio d'approximation** (ou erreur relative) d'une solution $y \in Sol(x)$ est défini par $R(x,y) = val(y)/opt(x)$. Pour un problème de maximisation, on le définit par $R(x,y) = opt(x)/val(y)$.*

Dans les deux cas, minimisation ou maximisation, ce ratio mesure la distance de la solution à une solution optimale. La solution est d'autant meilleure que ce ratio est proche de 1.

Définition 1.12. *Soient A un problème d'optimisation $\mathcal{NPO}$ et T un algorithme qui calcule pour toute instance x de A une solution $T(x) \in Sol(x)$. Etant donnée une application $r : \mathbb{N} \to [1, \infty[$, on dit que T un **algorithme de** $r(n)$**-approximation** ou que T est un **algorithme d'approximation à** $r(n)$ **près** si pour toute instance x de A, le ratio associé à $T(x)$ vérifie $R(x, T(x)) \leq r(|x|)$.*

*Un problème d'optimisation $\mathcal{NPO}$ pour lequel il existe un algorithme de $r(n)$-approximation est dit $r(n)$**-approximable** ou **approximable à** $r(n)$ **près**.*

Dans cette définition, la variable n est utilisée pour désigner la taille de la donnée d'entrée ($n = |x|$). Par la suite, si r n'est pas une fonction constante, on précisera toujours dans la notation $r(n)$ ce que désigne la variable n car dans certains cas le ratio d'approximation peut s'exprimer non pas en fonction de toute la taille de la donnée d'entrée mais seulement d'une partie de cette donnée (par exemple pour un graphe $G = (V, E)$, n peut désigner le nombre de sommets $|V|$ ou bien la taille de la matrice d'adjacence $|V|^2$). Dans ce cas, pour une instance x, le ratio associé à l'algorithme vérifie $R(x, T(x)) \leq r(n)$, où n est la taille du paramètre que l'on a précisé.

Définition 1.13. *Un problème d'optimisation $\mathcal{NPO}$ est dans la classe APX s'il est approximable à ϵ près, pour une constante $\epsilon > 1$. Soit A un problème $\mathcal{NPO}$. Un algorithme T est un **schéma d'approximation** pour A si pour toute instance x de A et tout rationnel $\epsilon > 0$, $T(x, \epsilon)$ renvoie une solution* *pour x à ϵ près. Un problème d'optimisation $\mathcal{NPO}$ est dans la classe $PTAS$ s'il admet un schéma d'approximation dont la complexité en temps est bornée par $q(|x|)$ où q est un polynôme pour $|x|$. Dans le cas, où la complexité en temps est bornée par $q(|x|, 1/(\epsilon-1))$ avec q un polynôme, on parle de* ***schéma complet d'approximation*** *et le problème est dans la classe $FPTAS$.*

Réductions entre problèmes

Nous introduisons maintenant une notion de ***réduction*** entre deux problèmes d'optimisation qui conserve les approximations. La réduction la plus classique est la ***L-réduction*** ou ***réduction linéaire***, définie dans [PY91]. D'autres réductions générales ont été introduites par la suite, en particulier la ***$PTAS$-réduction*** de [ACP95]. C'est celle que nous commençons par décrire. Pour distinguer les différents problèmes, on note $Sol_A(x)$ et $R_A(x, y)$ les solutions et ratios associés à x pour le problème A.

Définition 1.14. *Soient A et B deux problèmes $\mathcal{NPO}$. On dit que A est* ***$PTAS$-réductible*** *à B, ce qui est noté $A \leq_{PTAS} B$, s'il existe trois fonctions f, g et c vérifiant :*

1. *Pour toute instance x de A et tout rationnel $\epsilon > 1$, $f(x, \epsilon)$ est une instance de B calculable en temps polynomial par rapport à $|x|$.*
2. *Pour toute instance x de A, pour tout rationnel $\epsilon > 1$ et pour toute solution $y \in Sol_B(f(x, \epsilon))$, $g(x, y, \epsilon) \in Sol_A(x)$ et cette solution est calculable en temps polynomial par rapport à $|x|$ et $|y|$.*
3. *La fonction c est une fonction de $[1, \infty[$ dans $[1, \infty[$ calculable en temps polynomial et inversible.*
4. *Pour toute instance x de A, pour tout rationnel $\epsilon > 1$ et pour toute solution $y \in Sol_B(f(x, \epsilon))$,*

$$R_B(f(x, \epsilon), y) \leq c(\epsilon) \Rightarrow R_A(x, g(x, y, \epsilon)) \leq \epsilon$$

Si les fonctions f et g ne dépendent pas de ϵ, on dira que A est ***L-réductible*** *à B, ce qui est noté $A \leq_L B$.*

Cette définition décrit de manière très formelle les propriétés nécessaires pour pouvoir approximer les optima pour le problème A en se ramenant à des approximations pour B, tout en conservant un certain contrôle du ratio d'approximation et de la complexité en temps.

La principale utilité de ces réductions est énoncée dans la proposition suivante. Noter que les relations $\leq_{PTAS}$ et $\leq_L$ sont des relations de préordre, c'est à dire réflexives et transitives.

Proposition 1.15. *Soient A et B deux problèmes $\mathcal{NPO}$. Si $A \leq_{PTAS} B$ et $B \in APX$ (resp. $B \in PTAS$), alors $A \in APX$ (resp. $A \in PTAS$).*

Pour revenir à la description des réductions, il est quelques fois intéressant de préciser encore les propriétés des fonctions f, g et c pour pouvoir transférer des résultats d'approximation de B à A, en particulier quand $B \notin PTAS$.

Définition 1.16. *Soient A et B deux problèmes $\mathcal{NPO}$. Etant donné une L-réduction de A vers B avec les fonctions f, g et c de la définition 1.14, on dit qu'il s'agit d'une* ***réduction continue*** *s'il existe une constante $\alpha \geq 1$ et un polynôme p tels que :*

1. *Pour toute instance x de A, l'instance $f(x)$ de B vérifie $|f(x)| \leq p(|x|)$.*
2. *La fonction c est définie par $c(\epsilon) = \epsilon/\alpha$ (pour $\epsilon \geq \alpha$).*

S'il existe une telle réduction de A vers B, alors on le note $A \xrightarrow{\alpha} B$.

S'il existe une réduction continue de A vers B pour tout $\alpha > 1$, alors on le note $A \xrightarrow{1^+} B$.

Ce sont des réductions "économiques" (l'instance de B crée à partir de l'instance initiale de A est du même ordre de grandeur que cette instance initiale) et conservant le ratio d'approximation à une constante près.

Proposition 1.17. *Soit r une fonction croissante sur les entiers, soient A et B deux problèmes $\mathcal{NPO}$ tels que $A \xrightarrow{\alpha} B$ avec les notations précédentes. Si B est approximable à $r(n)$ près, alors A est approximable à $\alpha \times r(p(n))$ près, avec n repésentant la taille des instances.*

On peut écrire des variantes de cette proposition quand n désigne un autre paramètre des instances.

Première partie

Codages d'ordres

Chapitre 2

Introduction aux codages d'ordres

Cette synthèse regroupe les principales méthodes connues pour coder les ordres. L'objectif est de traiter le plus efficacement possible des opérations usuelles sur les ordres. Par efficace, on entend avoir des structures de données et des algorithmes associés pour représenter un ordre et traiter les opérations sur cet ordre avec de bonnes complexités en espace et en temps. De même qu'il existe des méthodes de compression de texte ou d'images, on pourra parler ici de compressions d'ordres puisqu'on essaiera en général de minimiser l'espace de stockage de l'ordre tout en conservant assez d'informations pour répondre vite aux requêtes sur l'ordre.

Cette synthèse est organisée en trois chapitres. Le premier rappelle les opérations usuelles sur les ordres et les trois structures de données couramment utilisées pour les ordres. Il précise ce qu'on entend par méthode de codage et donne les critères d'évaluation communément considérés. Plusieurs grandes catégories de méthodes sont identifiées et on présente comment sera organisée le deuxième chapitre. Ce deuxième chapitre est une liste de différentes méthodes présentes dans la littérature, mises ici sous forme de fiches techniques résumant leurs principales caractéristiques et avec des illustrations. Le troisième chapitre dresse un premier bilan sur cette synthèse et sur les méthodes de codage.

Un aspect de cette synthèse est que, pour pouvoir véritablement comparer ces méthodes, analyser leurs performances, exhiber des analogies, nous n'avons pas pu nous contenter d'une formalisation de haut niveau, nous nous sommes aussi attachés à étudier les implantations possibles et à étudier soigneusement les complexités correspondantes, ce qui a bien sûr nécessité de bien préciser le modèle de calcul dans lequel on se place.

Ce travail rassemble des résultats épars dans la littérature. Il présente sûrement des imperfections et ne se veut pas exhaustif, en particulier parmi les opérations à traiter nous nous intéresserons essentiellement à la comparaison entre éléments. Quoiqu'il en soit, nous proposons une classification originale que nous espérons utile pour l'analyse et la comparaison des méthodes de codage d'ordres ainsi que pour l'élaboration de nouvelles méthodes.

Le lecteur est bien sûr renvoyé aux nombreuses références pour plus de détails sur chacune des méthodes présentées.

2.1 Opérations usuelles sur les ordres

Etant donné un ordre $P = (X, \leq_P)$, voici une liste non exhaustive d'opérations classiques :

1 Soient $x, y \in X$, est-ce que $x \leq_P y$? (opération de ***comparaison***)

2 Soit $x \in X$, donner l'ensemble de ses successeurs $Succ_P(x)$ (resp. de ses prédécesseurs $Pred_P(x)$).

3 Soient $x, y \in X$, calculer $Maj_P(x, y) = \{z \in X \mid x \leq_P z \text{ et } y \leq_P z\}$ l'ensemble des majorants de x et y (resp. $Min_P(x, y) = \{z \in X \mid z \leq_P x \text{ et } z \leq_P y\}$ l'ensemble des minorants de x et y).

4 Si P est un treillis, soient $x, y \in X$, calculer la borne supérieure $x \vee_P y$ (resp. la borne inférieure $x \wedge_P y$) (opération de ***borne supérieure***, resp. ***borne inférieure***).

On peut bien sûr penser à d'autres exemples d'opérations :

5 Etendre les premières opérations aux parties de X, c'est à dire étant données $A, B \subseteq X$, vérifier si $A \leq_P B$ (signifiant $\forall a \in A$, $\forall b \in B$, $a \leq_P b$), calculer la borne supérieure $\vee_P A$ (resp. inférieure $\wedge_P A$).

6 Vérifier qu'une partie de X a certaines propriétés dans l'ordre (par exemple vérifier s'il s'agit d'un convexe).

7 Pour deux éléments $x, y \in X$ tels que $x \leq_P y$, trouver une chaine maximale (pour l'inclusion) de l'intervalle $[x, y]$, autrement dit qui relie x à y.

8 Calculer les points fixes d'une application de X dans X avec des propriétés pour l'ordre (par exemple si elle est monotone).

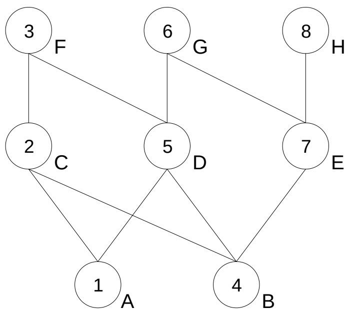
La plupart du temps les opérations un peu complexes sur les ordres comme les quatre dernières opérations font appel à certaines des quatre opérations élémentaires présentées au début, en particulier à l'opération de comparaison.

2.2 Structures classiques pour les ordres

Pour commencer, nous donnons ici les trois structures de données classiques utilisées pour représenter un ordre : la matrice d'incidence, les ensembles de successeurs et les ensembles de successeurs immédiats. Elles sont chacune illustrées par un exemple. La ***taille*** du codage est le nombre de bits nécessaires par élément.

Matrice d'incidence (BM)

Parmi les méthodes les plus classiques de codage, il y a d'abord la ***matrice d'incidence*** qui consiste à stocker un ordre $P = (X, \leq_P)$ à n éléments x_1,...,x_n dans une matrice M de taille $n \times n$ telle que $M[i, j] = 1$ si $x_i \leq_P x_j$ et $M[i, j] = 0$ sinon (aussi appelé ***matrice de la fermeture transitive*** ou simplement méthode de la ***matrice binaire***). Avec ce codage, les comparaisons entre éléments s'effectuent en temps constant. Il nécessite un espace total de n^2 bits, autrement dit la taille de codage vaut n.



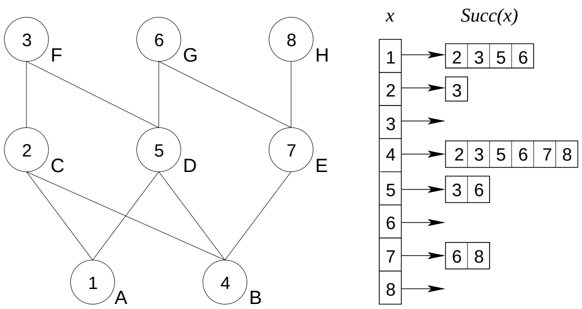
	1	2	3	4	5	6	7	8
1	1	1	1	0	1	1	0	0
2	0	1	1	0	0	0	0	0
3	0	0	1	0	0	0	0	0
4	0	1	1	1	1	1	1	1
5	0	0	1	0	1	1	0	0
6	0	0	0	0	0	1	0	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	0	0	0	1

Ensembles de successeurs (TC)

Une autre méthode consiste à stocker les ensembles $Succ_P(x)$ pour tout $x \in X$. Une implantation classique consiste à représenter ces ensembles par des listes chainées. Il s'agit d'un codage de taille inférieure à la matrice binaire en général mais les tests de comparaisons sont en temps non constant puisqu'on doit parcourir une liste chainée à n éléments dans le pire des cas. Ici dans le pire des cas, la taille de codage est en $\Theta(n)$ et la complexité en temps d'un test de comparaison en $\mathcal{O}(n)$. On peut améliorer cette complexité en stockant chaque ensemble dans un tableau trié (on trie les identificateurs des éléments), le test est alors une recherche dichomotique en temps $\mathcal{O}(\log_2(n))$.

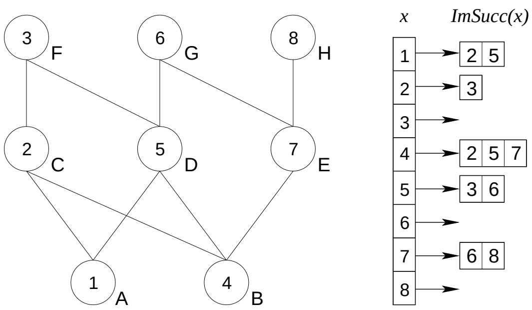
En notant $| <_P |$ le nombre de comparabilités dans l'ordre (c'est à dire $\sum_{x \in X} |Succ_P(x)|$), l'espace total de stockage de l'ordre est $\mathcal{O}(| <_P | \lceil \log_2(n) \rceil)$

bits.



Ensembles de successeurs immédiats (TR)

On peut encore essayer de réduire la taille de codage en stockant plutôt $ImSucc_P(x)$ pour tout $x \in X$ c'est à dire juste les arcs de la réduction transitive, mais un test de comparaison $x \leq_P y$? revient alors à effectuer un parcours (en profondeur ou largeur) du graphe de la réduction transitive à partir de x, ce qui augmente la complexité en temps. En utilisant des listes chainées, soit $| \prec_P |$ le nombre d'arcs de la réduction transitive de P (c'est à dire $\sum_{x \in X} |ImSucc_P(x)|$), l'espace total de stockage de l'ordre est $\mathcal{O}(| \prec_P | \lceil \log_2(n) \rceil)$ bits et un test de comparaison s'exécute en temps $\mathcal{O}(| \prec_P |)$.



Entre ces trois méthodes, on trouve ici la nécessité de choisir des compromis, par exemple en privilégiant plutôt un gain en temps ou plutôt un gain en espace.

Le passage de la représentation TR à TC ou à BM s'appelle le calcul de la fermeture transitive. Inversement, le passage de TC ou BM à TR s'appelle le calcul de la réduction transitive. On reviendra sur ces calculs dans la section 2.5. Le passage de BM à TC est évident : en parcourant la matrice d'incidence, on construit en temps $\mathcal{O}(n^2)$ les ensembles de successeurs. Le passage de TC à BM peut aussi s'effectuer de manière simple en initialisant la matrice d'incidence avec des zéros partout puis en parcourant les ensembles de successeurs pour remplir les valeurs de la matrice, soit un temps total en $\mathcal{O}(n^2)$. Notons qu'il est possible d'éviter la phase d'initialisation en ajoutant à la simple matrice $n \times n$ une petite structure de données complémentaire qui, tout en restant en espace $\Theta(n^2)$, permet d'effectuer les tests sur la matrice en temps constant sans avoir à initialiser cette matrice (voir [AHU71]). Dans ce cas, le passage s'effectue en temps $\mathcal{O}(n + | <_P |)$.

2.3 Qu'est-ce qu'une méthode de codage ?

Une ***méthode de codage d'ordres*** est une manière de stocker en mémoire des informations sur cet ordre **et** un ensemble d'algorithmes permettant de répondre efficacement à des requêtes concernant cet ordre. Pour nous, il faudra au moins que la méthode permette de répondre à toutes les questions de comparaisons "$x \leq_P y$?". Les trois représentations décrites dans la section précédente sont donc bien des méthodes de codage d'ordres.

Une méthode de codage est bien sûr intimement liée au modèle de calcul choisi, avec les contraintes sur le stockage en mémoire et sur la structure des algorithmes qu'il impose, sans même parler des mesures de complexité. A priori, on se situera dans le modèle ***Word-RAM***. Pour rappel, ce modèle séquentiel est une variante du modèle RAM de Cook et Reckhow [CR73] avec deux différences majeures :

1. Restrictions au niveau de la mémoire. Comme dans le modèle RAM, la mémoire est organisée en registres, mais ici ils ont tous une taille bornée : ils contiennent chacun un mot mémoire de w bits, w étant le paramètre important du modèle.
2. Enrichissement du jeu d'instructions. A coté des instructions RAM classiques (transferts de donnée, sauts conditionnels et inconditionnels, utilisation de constantes, adressages direct et indirect), on ajoute l'addition, la soustraction, les décalages variables à gauche et à droite, ainsi que les opérations booléennes bit à bit `OU`, `ET` et `NOT`. C'est ce qu'on appelle le jeu d'instructions restreint (sans autres intructions AC^0 et sans multiplication). L'exécution de chacune de ces instructions est supposée en temps constant et les opérandes sont bien sûr des mots de w bits.

Pour une synthèse sur le modèle Word-RAM et ses variantes (avec plus d'instructions), consulter l'article de Hagerup [Hag98]. Si on choisit plutôt ce modèle, c'est qu'il se rapproche un peu plus des machines réelles, en particulier sur les contraintes de stockage en mémoire, sur le jeu consistant à stocker plusieurs informations sur un seul mot mémoire, sur le parallélisme intrinsèque des opérations sur les mots mémoires comme l'addition ou les opérations booléennes bit à bit. Dans ce chapitre, on sera amené à donner des complexités en $\mathcal{O}$, Θ ou Ω. Quand elles dépendront de w, on fera apparaitre le terme en w sans le masquer dans les constantes cachées.

On n'insistera pas plus sur cette question de modèle de calcul, mais il faut savoir que certains auteurs travaillant sur processeur RISC se sont par exemple attachés à bien structurer les instructions RISC de leurs algorithmes de comparaison pour plus d'efficacité [VHK97].

Dans le texte, on distinguera le terme "***méthode de codage***" qui désigne la description de la méthode (dont la description formelle, les implantations possibles, la génération de codages) et le terme "***codage***" qui est l'implantation de la méthode pour un ordre fixé (c'est l'ensemble de bits associés à l'ordre considéré).

Une fois la méthode choisie, les problèmes qui lui sont liés sont la génération du codage d'un ordre à partir de la donnée d'un autre type de codage (notamment à partir des trois structures classiques) et la mise à jour du codage si l'ordre évolue (aspects dynamiques).

On peut identifier trois grandes composantes dans une méthode de codage, comprenant :

1. Des ***informations globales*** pour l'ordre considéré dans son ensemble. Cela peut aller du nombre n de sommets à la matrice d'incidence $n \times n$ définie précédemment, qui contient les réponses à toutes les requêtes de comparaisons possibles, en passant par différents paramètres propres à la méthode utilisée.
2. Des ***informations locales*** associées à chaque élément de l'ordre. C'est par exemple la liste des successeurs d'un élément dans la méthode des ensembles de successeurs.
3. Des ***algorithmes*** pour chacune des opérations à effectuer sur l'ordre, utilisant les informations globales et locales qui ont été stockées.

On peut faire quelques commentaires sur ces trois composantes. D'abord on n'a pas défini formellement la frontière entre informations globales et locales, sachant qu'on peut s'arranger pour faire passer des informations globales en local, et inversement. Par exemple, associer à chaque élément sa ligne dans la matrice d'incidence fait passer cette matrice "en local".

Dans la littérature, une catégorie de codage est plus fréquemment utilisée, il s'agit de ce qu'on appelle les ***représentations implicites***. Cela désigne des méthodes où les opérations sont résolues juste avec les informations locales des opérandes. C'est par exemple le cas si pour un ordre fixé P, pour déterminer si $x \leq_P y$ on a un algorithme qui n'utilise que les informations locales de x et les informations locales de y. Avec cette définition, la représentation par ensembles de successeurs est implicite, alors que la représentation par ensembles de successeurs immédiats ne l'est pas car pour tester si $x \leq_P y$ le parcours de graphe utilise les informations locales d'autres éléments que x et y. En ce qui concerne le cas de l'opération de comparaison, on trouve dans la littérature une définition des "codages" reprise par plusieurs auteurs [Bou84, HHN95] : pour un ordre $P = (X, \leq_P)$, cela consiste à trouver un ensemble Y, une application f de X dans Y et une relation $\rho : Y \times Y \to \{0, 1\}$ tels que pour tout $x, y \in X$, $x \leq_P y$ si et seulement si $\rho(f(x), f(y)) = 1$ et tels que cette dernière égalité se traite bien informatiquement. L'esprit de cette formulation correspond aux représentations implicites, où l'application f associe à chaque élément ses informations locales et ρ représente une relation décrite à la fois par les informations globales et par l'algorithme de comparaison. En général, on trouve $Y = \{0, 1\}^*$ l'ensemble des vecteurs de bits, mais Y pourrait tout aussi bien être un ensemble de structures de données plus complexes.

Cette définition des représentations implicites se base essentiellement sur la distinction, faite dans la description de la méthode, entre informations locales d'une part, et informations globales et algorithmes d'autre part. Il existe aussi une définition plus restrictive, très liée à l'implantation de la méthode. Elle est par exemple suggérée par Talamo *et al* [NTV99] qui définissent comme "implicites" des méthodes n'utilisant pas de pointeurs. Plutôt que de parler de pointeurs, nous pouvons donner une définition dans le modèle Word-RAM : pour une opération donnée, une méthode est dite ***fortement implicite*** si pour tout ordre il existe une implantation de l'opération consistant à d'abord charger les informations locales des opérandes dans des registres, puis à effectuer des instructions ne comprenant aucun adressage indirect (chargement ou stockage). Parmi les exemples étudiés dans la section 3.2, la méthode de la *dimension booléenne* est fortement implicite, la méthode de la *partition en chaines* est implicite dans ses deux versions mais seule sa version *plongement* est fortement implicite, sa version *partitionnement* ne l'est pas.

Quant aux algorithmes utilisés pour effectuer les opérations, leur structure peut dépendre peu, beaucoup ou pas du tout de l'ordre codé. Pour les trois méthodes classiques présentées dans la section 2.2, les algorithmes pour la comparaison ne dépendent pas de l'ordre codé. A l'opposé, si on s'intéresse à compresser au maximum l'espace de stockage d'un ordre pour répondre aux comparaisons, on pourrait imaginer une méthode consistant à associer à cet ordre la plus petite machine de Turing ou le plus petit algorithme Word-RAM pouvant générer sa matrice d'incidence. Une comparaison pourrait s'effectuer en énumérant la matrice jusqu'à la réponse à la requête. On n'aborde même pas la complexité d'une telle méthode, mais on pourrait alors dire que toutes les informations concernant l'ordre ont été placée dans l'algorithme traitant la comparaison, cet algorithme dépendant complètement de l'ordre considéré. On retrouve ici des questions liées à la problématique de la ***compilation***.

Nous allons maintenant préciser un peu ces différents points en présentant les mesures communément considérées pour évaluer les performances d'une méthode de codage.

2.4 Quels sont les critères d'évaluation ?

Les méthodes de codage d'ordres sont souvent jugées selon une liste de critères qui sont exposés ci-dessous. Parmi les travaux traitant de ces critères d'évaluation, on pourra consulter entre autres les articles [Cap94], [HHN95], [VHK97], [Fal98] et le plus récent [ZG01] très clair dont nous reprenons en partie la présentation. Ces critères sont des mesures de complexité qu'il s'agit d'optimiser simultanément si possible. Lorsqu'on parlera des opérations à traiter, il s'agit essentiellement de l'opération de comparaison même si tout ceci pourrait s'appliquer à d'autres opérations élémentaires.

Espace de stockage

Espace nécessaire pour stocker les identificateurs des éléments de l'ordre et les informations utilisées ensuite pour traiter les opérations. Il peut s'agir d'informations ***globales*** c.à.d. associées à la relation d'ordre en général ou ***locales*** c.à.d. consistant à associer à chaque élément un ensemble de données qui lui sont propres. Pour tout prendre en compte, on considère l'***espace total de stockage*** utilisé pour l'ordre et mesuré en bits. Quitte à diviser par le nombre d'éléments de l'ordre, on obtient la ***taille du codage*** qui est le nombre moyen de bits par élément.

Pour une méthode donnée et un ordre donné, on est souvent confronté à des problèmes d'optimisation pour choisir certains paramètres du codage et ainsi minimiser la taille du codage. Cette taille est d'autant plus importante que les deux points suivants (nombre d'instructions/opération et temps/opération) sont parfois proportionnel à la taille du codage.

Nombre d'instructions pour l'opération à traiter

Dépendant du modèle de calcul considéré, ce critère compte le nombre d'instructions de l'algorithme utilisé pour le traitement des opérations sur l'ordre (le nombre d'intructions du modèle Word-RAM ou par exemple le nombre d'instructions machines sur un processeur RISC précis).

Il faut en particulier distinguer les codages où ce nombre d'instructions dépend peu, beaucoup ou pas du tout des paramètres de l'ordre codé. Une méthode de codage est dit ***uniforme*** s'il est possible d'implanter les opérations à traiter avec un programme dont le nombre d'instructions ne dépend pas de l'ordre à coder.

Il faut aussi remarquer que dans certains contextes, par exemple le code compilé d'un logiciel écrit dans un langage orienté objets et faisant de nombreux tests de comparaison (pour le sous-typage), l'espace utilisé pour stocker les intructions de tests du logiciel peut dépasser l'espace de stockage de l'ordre sur les classes.

Temps d'exécution de l'opération à traiter

Puisque le principal objectif du codage de l'ordre est de pouvoir l'utiliser pour traiter les opérations de manière répétée, c'est une mesure de complexité fondamentale. Il est ainsi courant de rechercher des codages pour traiter les opérations en temps constant. Si pour une méthode, il existe une implantation des opérations où chacune s'exécute en temps constant indépendant de l'ordre, on parlera de méthode ***en temps constant***. L'algorithme de traitement des opérations peut contenir des boucles et donc sa complexité en temps peut ne pas être constante même pour une méthode uniforme. Cependant pour un ordre fixé il est toujours possible de dérouler une boucle en ajoutant des instructions, une telle implantation en “temps presque constant” (pour peu qu'on ajoute peu d'instructions) mais plus vraiment uniforme permet parfois de tirer partie du parallélisme d'instructions de certains processeurs.

Par exemple, les trois représentations classiques de la section 2.2 sont uniformes, mais seule la méthode de la matrice binaire est en temps constant.

Temps de génération du codage entier

En général, un ordre est initialement donné sous un format classique qui est soit la matrice de la relation binaire soit les listes chainées de tous les successeurs (ou prédécesseurs) des éléments, soit les listes des successeurs (ou prédécesseurs) immédiats. On mesure ici le temps nécessaire pour générer les informations du codage de l'ordre à partir de ces données initiales.

Temps et espace de mise à jour du codage (aspects dynamiques)

On mesure ici les ressources utilisées pour mettre à jour les informations du codage quand l'ordre est modifié. On cherche ainsi à gérer l'ajout ou la supression d'éléments ou de comparabilités dans l'ordre. Cette gestion dynamique est l'un des aspects les plus difficiles à traiter efficacement, en particulier quand on cherche à continuer à minimiser l'espace de stockage et/ou les temps d'exécution.

Ces critères se déclinent en fait selon de nombreuses nuances. Par exemple, pour l'espace de stockage, on mesure l'espace nécessaire pour stocker l'ensemble des informations du codage sans compter l'espace nécessaire pour stocker les instructions des algorithmes traitant ensuite les opérations, mais a priori il n'est pas forcément négligeable (par exemple si on transfère des informations à l'intérieur de ces algorithmes, ce qu'on évite en général pour avoir les algorithmes de traitement les plus simples possibles). Par exemple Gambosi, Nesestril et Talamo [GNT87a, GNT87b, GNT90] ont proposé une méthode de codage (*dimension booléenne*) qu'on pourrait qualifier de paradoxale, où tous les efforts sont placés dans le choix de l'algorithme de traitement des opérations. Le nombre d'instructions dépend alors fortement de l'ordre à coder. Pour de telles méthodes de codage non uniformes, il faut bien sûr ajouter l'espace de stockage des algorithmes à l'espace total de stockage.

Certaines problématiques privilégient parfois certaines méthodes de codages. Par exemple supposons que les tests de comparaisons pour un ordre P soient utilisés dans un programme à compiler. L'ordre P est supposé connu et on suppose que dans les tests de la forme $x \leq_P y$? effectués par le programme, y est connu à la compilation alors que x est une variable. Dans ce cas, on peut associer deux informations à chaque élément, une à utiliser lorsqu'il est à gauche dans le test $\leq_P$? ("information à gauche") et l'autre quand il est à droite ("information à droite"). Avec nos hypothèses, dans le programme machine généré à la compilation, on peut stocker pour chaque test l'information du type à droite dans les instructions. Si beaucoup de variables sont générés dynamiquement, ce stockage des informations "à droite" deviendra négligeable devant la somme des informations "à gauche" des objets générés. Il s'agit alors de minimiser la taille des informations "à gauche", celle des informations "à droite" étant moins cruciale. Ces considérations sont par exemple abordées dans [VHK97] et [ZG01], elles interviennent pour les programmes en langages orientés objets.

Bornes inférieures

Etant donnée une méthode de codage utilisée sur les ordres à n éléments, alors il existe des ordres dont l'espace total de codage (on ajoute l'espace pris par l'algorithme de comparaison si le codage n'est pas uniforme) est en $\Omega(n^2)$ bits. En effet le nombre d'ordres sur n éléments vaut $2^{\Theta(n^2)}$: rien qu'en dénombrant les ordres bipartis avec $n/2$ éléments minimaux fixés et $n/2$ éléments maximaux, on compte $2^{n^2/4}$ ordres. Cette borne est d'ailleurs assez proche du nombre total d'ordres puisque ce nombre P_n d'ordres sur n éléments vérifie $\log_2(P_n) = n^2/4 + 3n/2 + \mathcal{O}(\log(n))$ [KR75].

En fait cette borne inférieure dans le pire des cas est renforcée par un résultat plus général concernant l'espace total de codage en moyenne. Sans perte en espace, le codage d'un ordre (algorithmes de traitement des opérations compris) peut s'exprimer sous forme d'un vecteur de bits, l'espace de stockage est alors la longueur du vecteur. Pour une méthode de codage fixée, les codages de deux ordres différents correspondent à deux vecteurs de bits différents. Il suffit alors d'appliquer le lemme 2.1 très simple avec l'évaluation du nombre P_n pour obtenir le résultat suivant : étant donnée une méthode de codage, l'espace total de codage (algorithmes compris) pour les ordres à n éléments est en moyenne en $\Omega(n^2)$ bits, pour une distribution uniforme de ces ordres.

Lemme 2.1. *Soit $\mathcal{C}$ un ensemble et ϕ une application injective de $\mathcal{C}$ dans l'ensemble des mots sur $\{0,1\}$. Alors, en notant $|\phi(x)|$ la longueur du mot $\phi(x)$ pour $x \in \mathcal{C}$,*

$$\frac{1}{|\mathcal{C}|} \sum_{x \in \mathcal{C}} |\phi(x)| \geq \log_2(|\mathcal{C}|) - 2$$

Preuve. L'application injective ϕ qui minimise $1/|\mathcal{C}| \sum_{x\in\mathcal{C}} |\phi(x)|$ est clairement celle dont l'image est l'ensemble de tous les mots de longueur strictement inférieure à k plus des mots de longueur k pour compléter. L'entier k correspondant est le plus petit qui vérifie $1+2+\ldots+2^k \geq |\mathcal{C}|$, autrement dit c'est $k = \lceil \log_2(|\mathcal{C}|+1) - 1 \rceil$. Pour cette application ϕ, $1/|\mathcal{C}| \sum_{x\in\mathcal{C}} |\phi(x)| \geq 1/|\mathcal{C}| \sum_{i=0}^{k-1} i.2^i = 1/|\mathcal{C}|((k-1)2^{k+1}+2)$. Comme $|\mathcal{C}| \leq 2^{k+1}$ et $k \geq \log_2(|\mathcal{C}|) - 1$, la somme vérifie bien $1/|\mathcal{C}| \sum_{x\in\mathcal{C}} |\phi(x)| \geq \log_2(|\mathcal{C}|) - 2$. ■

Par conséquent, asymptotiquement la méthode de la matrice binaire est optimale en espace. Ce qui ne remet pas en cause toutes les autres techniques de codages qui obtiennent de bien meilleurs espaces de codage en moyenne pour d'autres distributions que la distribution uniforme.

Concernant les informations locales (qui peuvent chacune s'exprimer sous forme d'un vecteur de bits de longueur pas nécessairement constante), si chacun des n éléments reçoit des informations distinctes, alors nécessairement la somme de ces informations locales représente un espace en $\Omega(n\lceil\log_2(n)\rceil)$ bits.

Classes d'ordres

On peut aussi ne s'intéresser à coder qu'une classe restreinte d'ordres. Certains ordres admettent naturellement des codages très performants du fait de leur définition : ordres d'intervalles, ordres d'inclusion de figures géométriques, treillis booléens, ... (pour d'autres classes connues, voir par exemple [Moh89]) Etant donnée une classe d'ordres $\mathcal{C}$, une méthode de codage est dite ***optimale pour*** $\mathcal{C}$ si les codages lui correspondant vérifient les points suivants :

1. La taille maximum de l'espace total de stockage d'un ordre de $\mathcal{C}_n$ est en $\mathcal{O}(\log_2|\mathcal{C}_n|)$, où $\mathcal{C}_n$ désigne les ordres de $\mathcal{C}$ à n éléments.
2. Les tests de comparaisons sont en temps constant.

En particulier si l'espace total de stockage est en $\mathcal{O}(n\lceil\log_2(n)\rceil)$, que tous les éléments reçoivent des informations locales différentes et que les tests de comparaisons s'effectuent en temps constant, on a affaire à une méthode de codage optimale. Il faut noter qu'il existe beaucoup de classes pour lequelles il existe une méthode de codage optimale mais où générer un tel codage optimal est un problème difficile ($\mathcal{NP}$-complet ou plus difficile). C'est notamment le cas de classes issues de méthodes de codages dont certains paramètres sont bornés (par exemple les ordres de dimension 3). Pour introduire cette nuance et aussi la notion de reconnaissance de la classe, on définit la notion de méthode ***polynomialement optimale*** pour une classe $\mathcal{C}$: c'est une méthode optimale pour $\mathcal{C}$ pour laquelle on connait un algorithme reconnaissant en temps polynomial si un ordre appartient à $\mathcal{C}$ et si c'est le cas, générant pour cet ordre en temps polynomial en n (à partir de des représentations BM, TC ou TR) un codage vérifiant les points 1 et 2.

Liens entre ordres et ordres bipartis

En théorie des ordres, il est fréquent d'associer à un ordre $P = (X, \leq_P)$ les ordres bipartis $B(P) = (X, X, \leq_P)$, $\overline{B}(P) = (X, X, \not\leq_P)$, $B^0(P) = (X, X, <_P)$

ou $\overline{B^0}(P) = (X, X, \not<_P)$. Il est connu que certaines propriétés de P se lisent bien sur ces bipartis. C'est pourquoi pour étudier ou élaborer des méthodes de codage, il est parfois conseillé de se placer sur l'un de ces quatre bipartis. C'est par exemple utile pour la méthode de la *2-dimension* (voir Chapitre 5), mais pas vraiment intéressant pour une méthode comme la *représentation par ensembles de successeurs immédiats*.

Dans le cas particulier de l'opération de comparaison, il y a un lien fort entre coder un ordre P et coder les ordres bipartis $B(P)$, $\overline{B}(P)$, $B^0(P)$ et $\overline{B^0}(P)$. A partir d'un codage de P, on peut construire facilement un codage pour chacun de ces ordres bipartis. Par exemple, pour coder $B(P)$, on conserve le codage de P, on associe à chaque élément x de $B(P)$ l'élément $x' \in X$ lui correspondant dans P ainsi qu'un bit $b(x)$ fixé à 0 si l'élément est minimal et 1 s'il est maximal. Les tests sont résolus avec $x <_{B(P)} y$ si et seulement si $x' \leq_P y'$ et $b(x) < b(y)$. Inversement à partir du codage d'un des quatre ordres bipartis, on peut construire facilement un codage de P. Par exemple pour coder P, on peut conserver le codage de $B(P)$ et associer à chaque élément de x les deux éléments x' et x'' lui correspondant dans $B(P)$, x' étant le minimal et x'' le maximal. Les tests sont résolus avec $x \leq_P y$ si et seulement si $x' \leq_{B(P)} y''$. On constate que, dans les deux sens, ces transformations de codages ajoutent au plus un temps constant aux tests de comparaisons et conservent les ordres de grandeurs en espace des codages. Ici on se place donc sur un biparti pour appliquer une méthode de codage au biparti, générer un codage de ce biparti et ensuite transformer ce codage en un codage de P. Remarquer quand même que dès lors que l'on s'intéresse uniquement aux codages issus d'une méthode ***fixée***, alors rien ne dit que le comportement de la méthode (par exemple la valeur optimale d'un paramètre) sera le même pour P et ses ordres bipartis $B(P)$, $\overline{B}(P)$, $B^0(P)$ et $\overline{B^0}(P)$.

Le précédent paragraphe s'applique presque exclusivement à l'opération de comparaison. Pour d'autres opérations comme la borne supérieure de deux éléments, on n'a pas nécessairement ce schéma de transformation du codage d'un des quatre bipartis vers un codage de l'ordre initial.

2.5 Liens avec d'autres problèmes

Accessibilité dans les graphes orientés - Fermeture transitive

Le problème de l'***accessibilité*** dans un graphe non orienté consiste à déterminer pour un graphe orienté $G = (V, E)$ et deux sommets $x, y \in V$, s'il existe ou non un chemin de x vers y. Pour un graphe orienté G, être capable de répondre à toutes les requêtes d'accessibilité entre deux sommets est équivalent au calcul de la ***fermeture transitive*** du graphe.

Pour un graphe sans circuit $G = (V, E)$, la fermeture transitive correspond à un ordre $P = (V, \leq)$ et il existe dans G un chemin d'un sommet x vers un sommet y si et seulement si $x \leq y$. Inversement pour un ordre P, il existe plusieurs graphes orientés dont la fermeture transitive redonne P et en particulier le plus petit (pour l'inclusion des arcs) est la réduction transitive de P.

Pour un graphe possédant des circuits $G = (V, E)$, on peut se ramener à l'accessibilité dans les graphes sans circuit en quotientant le graphe par ses composantes fortement connexes (tous les sommets d'une composante fortement connexe sont accessibles les uns par rapport aux autres). Le calcul des composantes fortement connexes et du graphe quotient peut s'effectuer en temps et espace $\mathcal{O}(|V|+|E|)$ avec par exemple l'algorithme de Tarjan [Tar72].

Par conséquent, les méthodes de codage d'ordres peuvent être utilisées pour stocker les graphes orientés et répondre efficacement aux questions d'accessibilité (moyennant un quotient par les composantes fortement connexes), et inversement.

On notera que si un codage d'ordre effectue les comparaisons en temps constant, générer ce codage à partir de la réduction transitive fournit le calcul de la matrice de la fermeture transitive.

Concernant le calcul de la fermeture transitive d'un graphe orienté, on peut rappeler qu'il existe des algorithmes matriciels, des algorithmes de graphes et des algorithmes hybrides mélant les deux catégories précédentes. Les meilleurs algorithmes matriciels sont basés sur la multiplication de matrices : soit n le nombre de sommets d'un graphe qui est stocké dans une matrice d'incidence $n \times n$, soit $\alpha > 1$, on peut calculer la fermeture transitive en temps $\mathcal{O}(n^\alpha)$ si et seulement si on peut calculer le produit de deux matrices $n \times n$ en temps $\mathcal{O}(n^\alpha)$ [Mun71, FM71, AHU74] (à ce jour, le meilleur α pour la multiplication de matrices vaut 2.376 [CW87]). Si les bornes asymptotiques des algorithmes matriciels sont très intéressantes, elles masquent des facteurs constants importants et nécessitent d'avoir la matrice d'incidence en entier. Les algorithmes de graphes passent par des parcours de graphes, comme des tris topologiques si le graphe est sans circuit, ils sont particulièrement adaptés quand la donnée en entrée est la liste des arcs du graphe. On peut citer en particulier l'algorithme de Goralcikova et Koubek [GK79] qui, pour un graphe sans circuit à n sommets, calcule l'ordre $P = (X, \leq_P)$ qui est sa fermeture transitive en temps $\mathcal{O}(n + \sum_{x \in X} \sum_{y \in ImSucc_P(x)} |Succ_P(y)|)$, ce qu'on peut majorer par $\mathcal{O}(n| \prec_P |)$. Le lecteur intéressé peut par exemple consulter les premiers chapitres de la thèse de Nuutila [Nuu95].

D'un point de vue très théorique, l'accessibilité dans les graphes orienté (parfois appelée problème ***GAP***) joue un rôle central en théorie de la complexité [Pap94, LdR96, Weg87] : le complément de GAP est dans NL ce qui fournit une preuve du théorème de Immerman-Szelepcsenyi sur les classes non déterministes en espace, le problème GAP est NL-complet, il est aussi dans AC^1 ce qui permet de simuler de l'espace séquentiel par du temps parallèle, il existe aussi une borne inférieure sur la profondeur des circuits booléens résolvant GAP donnée par un théorème de Karchmer et Wigderson ... La problématique de GAP n'est a priori pas exactement la même que notre problématique de “compression” : pour GAP, on reçoit en entrée un graphe et deux de ses sommets et on recherche les ressources nécessaires pour résoudre directement l'accessibilité entre ces deux sommets, alors que dans notre problématique, il s'agit plutôt de transformer le codage du graphe pour avoir ensuite besoin de moins de ressources pour résoudre l'accessibilité. Cela n'empèche pas que certains résultats de complexité pour GAP puissent être exploités pour notre problématique.

Voisinage dans un graphe non orienté

Le problème de ***voisinage*** dans un graphe non orienté consiste à déterminer pour un graphe non orienté $G = (V, E)$ et deux sommets $x, y \in V$ si x et y sont adjacents ou pas. Plusieurs auteurs se sont intéressé à des codages de graphes permettant de répondre efficacement à ces questions, sans passer par la matrice d'adjacence ou les listes d'adjacences. En particulier, des ***représentations implicites*** ont été proposés pour certaines classes de graphes non orientés : chaque sommet reçoit des informations locales sous forme d'un vecteur de bits et on dispose d'un algorithme prenant en entrée juste les informations locales de deux sommets pour déterminer s'ils sont adjacents. Cette notion de représentation implicite pour traiter l'adjacence dans les graphes non orientés a été mise en avant, formalisée et étudiée par Muller [Mul88] et Kannan, Naor et Rudich [KNR92]. Un important travail de synthèse a été réalisé par Spinrad qui a replacé dans ce cadre de nombreux travaux sur les graphes [Spi97]. L'utilisation de représentations implicites pour l'adjacence a par ailleurs été généralisée au calcul de distance entre deux sommets dans les travaux de Gavoille *et al* [GPPR01, GKK$^+$01].

Les méthodes sur les graphes non orientés peuvent être utilisées pour coder les ordres de la manière suivante. Etant donné un ordre $P = (X, \leq_P)$, on construit son graphe de comparabilité $G_{comp}(P)$ (ce qui nécessite qu'on ait sa fermeture transitive) et on code ce graphe non orienté pour traiter le problème de voisinage. Il suffit alors de calculer une extension linéaire L de P, ce qui est réalisable en temps et espace $\mathcal{O}(n + |<_P|)$ avec $n = |X|$. Chaque élément de P reçoit son indice dans l'extension linéaire, ce qui revient seulement à ajouter $\lceil \log_2(n) \rceil$ bits d'informations locales pour un élément. On résout alors une comparaison $x \leq_P y$ en vérifiant si x et y sont adjacents dans $G_{comp}(P)$ et $x \leq_L y$ (ce dernier test s'effectue en temps constant). Un des atouts de cette méthode est que pour traiter le problème de voisinage dans un graphe non orienté, on peut aussi bien coder le graphe que son complémentaire, ce qui est parfois plus intéressant. Par exemple, Talamo *et al* ont proposé une méthode (*dimension booléenne*) qui s'avère obéir à ce schéma de codage (voir [GNT90], théorème 2.4).

La similitude entre les codages d'ordres et le problème de voisinage est particulièrement vraie dans le cas des ordres bipartis pour lesquels la construction du graphe de comparabilité et la donnée d'une extension linéaire sont immédiates. Toujours dans [GNT90], Talamo *et al* qui basent une de leurs méthodes sur le codage des bipartis, passent par ce genre de considérations (voir [GNT90], page 256).

2.6 Proposition de classification

Il existe de nombreuses approches pour coder les ordres. Cependant il existe aussi des traits communs entre plusieurs des méthodes connues et on peut essayer d'identifier des catégories de codages. C'est ce que nous présentons ici. Les catégories ci-dessous ne sont pas disjointes, certaines méthodes apparaissent dans plusieurs catégorie suivant le point de vue adopté.

Souvent les méthodes utilisent les classes particulières d'ordres qu'on sait bien coder et aussi les constructions classiques sur les ordres comme le produit direct ou le produit lexicographique.

Parcours de graphe

Se ramener à des parcours de graphe, comme c'est le cas pour la méthode des *ensembles de successeurs immédiats* qui revient au problème d'accessibilité dans un graphe orienté sans circuits. Remarquer que des opérations comme la comparaison sont alors coûteuses en temps.

Plongements

Construire un plongement d'ordre dans un ordre qu'on sait bien coder. Très souvent il s'agit d'un plongement dans un treillis, ce choix présente les atouts suivants :

1. Si on sait coder un ordre P, on sait coder son complété de Dedekind-MacNeille $DM(P)$: on sait que s'il existe un plongement d'ordre de P dans un treillis T, alors il existe un plongement d'ordre de $DM(P)$ dans T (voir [Bou71] ou [DP91] pour les aspects constructifs).
2. Si l'ordre à coder est un treillis, on peut imposer que le plongement soit aussi un $\vee$-morphisme ou un $\wedge$-morphisme, pour pouvoir traiter efficacement les opérations de borne supérieure ou borne inférieure. En particulier les $\wedge$-plongements et $\vee$-plongements dans des treillis distributifs ont des propriétés intéressantes [Bou71]. Pour une étude des $\wedge$-plongements et $\vee$-plongements, voir aussi [Mar80].
3. Soient A et B deux parties d'un ordre P, vérifier que $A \leq_P B$, c'est à dire $\forall a \in A$, $\forall b \in B$, $a \leq_P b$, nécessite $|A| \times |B|$ comparaisons. Supposons qu'on ait un plongement d'ordre ϕ dans un treillis T où on sait bien traiter les bornes supérieures et inférieures, alors $A \leq_P B$ si et seulement si $\bigvee_{a \in A} \phi(a) \leq_T \bigwedge_{b \in B} \phi(b)$, ce qui nécessite $|A| + |B| - 1$ opérations en tout.

En général, les problèmes d'optimisation associés à de telles méthodes consistent à minimiser le nombre d'éléments de l'ordre dans lequel on plonge l'ordre initial.

Pour le traitement des bornes inférieures (resp. supérieures) dans un treillis, l'utilisation d'un $\wedge$-plongement ϕ (resp. $\vee$-plongement) du treillis initial $P = (X, \leq_P)$ dans un treillis T, pose parfois un problème supplémentaire. En effet, soient $x, y \in X$, pour calculer $x \wedge_P y$, on calcule son code $\phi(x \wedge_P y) = \phi(x) \wedge_T \phi(y)$, mais on peut considérer que le calcul n'est pas terminé si on n'a pas retrouvé à quel élément correspond le code calculé, ce qui revient à implanter ϕ^{-1}. Plus généralement, on peut se poser la question du traitement de la correspondance entre les éléments et leurs informations locales. On est alors ramené à des recherches d'élément dans un ensemble à partir des informations de l'élément. De très nombreuses techniques existent pour résoudre ce problème (hachage, arbres de recherche ...), le lecteur est renvoyé à [CLR94] pour des structures de données, des algorithmes et leurs complexités.

Représentation par intersection ou par union

Construire une famille d'extensions (resp. réductions) de l'ordre à coder dont l'intersection (resp. l'union) redonne l'ordre et telle que chaque extension (resp. réduction) est un ordre qu'on sait bien coder. On parle aussi de ***codage par recouvrement*** pour la représentation par union [Wes85].

De telles méthodes justifient l'étude pour un ordre de ses extensions (resp. réductions) appartenant à une classe particulière d'ordres (voir le Chapitre 6). En particulier toute classe d'ordres $\mathcal{C}$ contenant toutes les chaines permet de définir une représentation par intersection en prenant les extensions dans $\mathcal{C}$ (car tout ordre est intersection de ses extensions linéaires [Szp30]). De même toute classe d'ordres contenant tous les ordres avec une seule comparabilité permet de définir une représentation par union.

En gros, on ajoute ou on retire des comparaisons pour obtenir des ordres qu'on sait bien coder. En général, les problèmes d'optimisation associés à de telles méthodes consistent à minimiser le nombre d'ordres dans la famille utilisée pour représenter l'ordre à coder.

La génération de tels codages peut souvent se réinterpréter en terme de recouvrement des arêtes d'un biparti par des bipartis d'une classe précise. Pour un ordre $P = (X, \leq_P)$, le biparti à recouvrir est souvent l'un des quatre suivants : $B(P) = (X, X, \leq_P)$, $\overline{B}(P) = (X, X, \not\leq_P)$, $B^0(P) = (X, X, <_P)$ ou $\overline{B^0}(P) = (X, X, \not<_P)$.

Compression d'ensembles

On part du principe que tester si $x \leq_P y$ est équivalent à tester si $y \in Succ_P(x) \cup \{x\}$. Il existe alors plusieurs manières de représenter les ensembles $Succ_P(x)$ ou $Succ_P(x) \cup \{x\} = \uparrow_P x$ et de traiter ensuite l'opération $\in$. L'objectif reste bien sûr de minimiser l'espace de stockage en gardant la vitesse de traitement. Cela passe en général par une bonne numérotation des éléments.

On peut bien sûr faire la même chose avec les ensembles de prédécesseurs, car $x \leq_P$ si et seulement si $x \in Pred_P(y) \cup \{y\} = \downarrow_P y$.

Partitionnement

Exploiter le fait que certains éléments ont des ensembles de successeurs (ou prédécesseurs) qui se ressemblent beaucoup, ou bien que certains éléments "voient" (c.à.d. ont dans leur ensemble de successeurs ou prédécesseurs) des structures régulières (dans le sens facile à décrire). Ce genre de configurations suggère des compressions possibles de l'ordre. Il s'agit donc de regrouper certains éléments entre eux, en partitionnant l'ensemble sous-jacent à l'ordre, pour faire apparaître des propriétés exploitables pour coder l'ordre. Plus formellement, étant donné un ordre $P = (X, \leq_P)$, pour résoudre les tests $x \leq_P y$, on peut partitionner X avec une première famille d'ensemble $\{S_i\}_{1 \leq i \leq k}$ et avec une deuxième famille d'ensembles $\{T_j\}_{1 \leq j \leq l}$ telles qu'on sache bien traiter, pour i et j fixés, les tests $x \leq_P y$? avec $x \in S_i$ et $y \in T_j$. Cette technique est donc à marier avec d'autres méthodes de codage pour traiter ces tests quand i et j sont fixés (par exemple des méthodes de compression d'ensembles ou des méthodes optimales pour certaines classes d'ordres).

Les partitions $\{S_i\}_{1 \leq i \leq k}$ et $\{T_j\}_{1 \leq j \leq l}$ ne sont pas nécessairement identiques. Il existe ainsi trois cas particuliers importants :

Type 1. Les partitions $\{S_i\}_{1 \leq i \leq k}$ et $\{T_j\}_{1 \leq j \leq l}$ sont identiques.

Type 2. La partition $\{S_i\}_{1 \leq i \leq k}$ est quelconque, mais $\{T_j\}_{1 \leq j \leq l}$ est réduite à $\{X\}$.

Type 3. La partition $\{T_j\}_{1 \leq j \leq l}$ est quelconque, mais $\{S_i\}_{1 \leq i \leq k}$ est réduite à $\{X\}$ (peut se voir comme le cas 2 appliqué au dual de l'ordre).

Un très bon exemple pour le type 1 de partitionnement est donné par la *décomposition modulaire* d'un ordre. En effet, soit $P = (X, \leq_P)$ un ordre et $\{S_i\}_{1 \leq i \leq k}$ sa partition en modules forts, alors pour i et j fixés, si $i \neq j$, on peut résumer facilement les comparaisons entre éléments de S_i et éléments de S_j : soit pour tout $x \in S_i$ et $y \in S_j$ on a $x \leq_P y$, soit pour tout $x \in S_i$ et $y \in S_j$ on a $x \not\leq_P y$. On peut ainsi proposer le codage suivant : on associe à chaque élément x le numéro $s(x)$ du module fort auquel il appartient et aussi l'ensemble de ses successeurs dans $S_{s(x)}$, on conserve comme informations globales l'ordre quotient sur les modules forts (représenté par exemple par ses ensembles de successeurs). Ce codage économise clairement de l'espace par rapport à la représentation par ensembles de successeurs de l'ordre initial et les tests de comparaisons sont aussi accélérés, car les tailles des ensembles que l'on conserve sont plus petites. La décomposition modulaire se marie donc bien avec la représentation par ensembles de successeurs, mais aussi avec beaucoup d'autres méthodes de codage (voir par exemple la *dimension* [Tro91]).

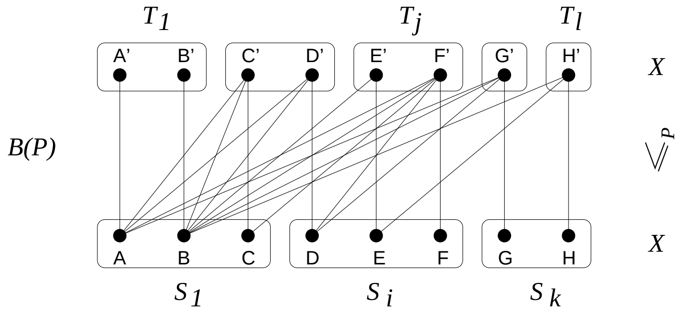
Les types 2 et 3 s'associent bien avec des méthodes de compression d'ensembles. On va par exemple s'intéresser à la compression par rapport aux ensembles de prédécesseurs, c.à.d. se basant sur $y \leq_P x$ si et seulement si $y \in Pred_P(x) \cup \{x\} = \downarrow_P x$. Il existe alors deux schémas importants de méthodes utilisant partitionnement et compression d'ensembles : le schéma A correspond au type 2 de partitionnement, le schéma B au type 3. On s'efforce ainsi de trouver des partitions telles que :

Schéma A. $\forall i, \forall x \in X$, l'ensemble $S_i \cap \downarrow_P x$ est facile à décrire dans S_i (on a un moyen efficace de traiter la question d'appartenance d'un élément de S_i à l'ensemble $S_i \cap \downarrow_P x$).

Schéma B. $\forall j, \forall x \in T_j$, l'ensemble $X \cap \downarrow_P x$ est facile à décrire.

On a laissé $X \cap \downarrow_P x$ (qui vaut bien sûr $\downarrow_P x$) dans la dernière définition pour mettre en valeur la symétrie entre les schémas A et B. Plusieurs des méthodes présentées dans la section 3.2 appartiennent à l'un de ces schémas. Dans le schéma A, on peut compter l'*algorithme de Cohen* et la *partition en antichaines* ($S_i \cap \downarrow_P x$ est vide ou réduit à un singleton), ainsi que la *partition en chaines* ($S_i \cap \downarrow_P x$ est une chaine). Dans le schéma B, il y a les *PQ-encodings* où on partitionne en $\{T_j\}_{1 \leq j \leq l}$ tel que pour tout j, il existe une numérotation de X telle que pour tout $x \in T_j$, $\downarrow_P x$ est un intervalle d'entiers. Les descriptions précises de ces méthodes sont données dans la section 3.2. Noter que dans le schéma A, il est conseillé d'autoriser que $S_i \cap \downarrow_P x$ puisse être vide (sinon on risque d'avoir des contraintes difficiles à réaliser).

Comme on utilise deux partitions (une pour la gauche du test de comparaison et une pour la droite), ces techniques de partitionnement se visualisent très naturellement sur les bipartis $B(P)$ ou $B^0(P)$ (la partition $\{S_i\}_{1 \leq i \leq k}$ concerne les minimaux du biparti et $\{T_j\}_{1 \leq j \leq l}$ les maximaux). La figure suivante représente cette façon de voir les partitionnements.



Pourquoi partitionner plutôt que simplement recouvrir X ? On peut justifier ce choix par la volonté d'avoir des opérations rapides, si possible en temps constant, et dans ce cas il est bon de pouvoir accéder en temps constant à la bonne partie à considérer pour chaque opérande.

Du point de vue de l'implantation des méthodes utilisant le partitionnement, on associe souvent à un élément x le numéro s_x (resp. s'_x) de la partie à laquelle il appartient dans $\{S_i\}_{1\leq i\leq k}$ (resp. $\{T_j\}_{1\leq j\leq l}$), un identifiant id_x (resp. id'_x) de x dans cette partie (par exemple un entier), et des informations $(t_x[1],\ldots,t_x[k])$ (resp. $(t'_x[1],\ldots,t'_x[l])$) où $t_x[i]$ désigne l'information locale de x par rapport à S_i (utilisée pour les tests $y \leq_P x$? où x est à droite et $y \in S_i$) et $t'_x[j]$ désigne l'information locale de x par rapport à T_j (utilisée pour les tests $x \leq_P y$? où x est à gauche et $y \in T_j$). Ces dernières informations locales peuvent être par exemple un entier, un uplet d'entiers, un vecteur de bits ou bien d'autres structures de données. Noter qu'on n'a parfois besoin que d'une partie de ces informations, en particulier pour les types 2 et 3 de partitionnement.

On peut aussi voir cette catégorie comme de la représentation par union puisque l'ordre P est union des ordres $P_{i,j} = (X, \leq_{i,j})$ avec $x \leq_{i,j} y$ si et seulement si $x \leq_P y$ et $x \in S_i$ et $y \in T_j$ (on vérifie facilement que c'est bien une relation d'ordre). Mais c'est une représentation par union très particulière avec de fortes contraintes puisque les réductions $P_{i,j}$ sont issues de partitions de X, alors que les représentations par union habituelles sont issues de recouvrements des comparabilités, une de ces contraintes est que pour un test $x \leq_P y$? on sait immédiatement quelle réduction $P_{i,j}$ regarder.

Compression logique (théorie de l'information)

C'est une catégorie large pour englober les travaux qui, pour un ordre donné, jouent autant sur les informations locales et globales que sur la structure des algorithmes de traitement des opérations. Ces algorithmes dépendent alors très fortement de l'ordre à coder.

Par exemple, les auteurs de [GNT87a, GNT87b, GNT90] se sont intéressés à une méthode (*dimension booléenne*) où chaque élément x reçoit un vecteur de bits représentant un k-uplet d'entiers $(x_1,\ldots,x_k)$. Ils cherchent alors une formule boolénne aussi courte que possible, vraie si et seulement si $x \leq_P y$, qui utilise `OU`, `ET`, `NON` et $\leq$ sur les entiers, avec les variables x_1,...,x_k,y_1,...,y_k correspondant aux éléments x et y. Par exemple, considérons l'ordre $N = (X, \leq_N)$ défini sur $X = \{a,b,c,d\}$ et dont les seules comparabilités sont $a \leq_N c$, $b \leq_N c$ et $b \leq_N d$. Un codage possible de N consiste à associer à chaque élément x un couple d'entier $\phi(x) = (x_1, x_2)$, en prenant $\phi(a) = (3,1)$, $\phi(b) = (1,2)$, $\phi(c) = (4,3)$ et $\phi(d) = (2,4)$, puis à se servir de la formule $(x1 \leq y_1)$ `ET` $(x_2 \leq y_2)$ vraie si et seulement si $x \leq_N y$ (la vérification est laissée au lecteur). Mais on aurait pu aussi bien proposer le codage suivant :

on associe toujours des couples d'entiers en prenant cette fois $\phi(a) = (0,0)$, $\phi(b) = (0,1)$, $\phi(c) = (1,0)$ et $\phi(d) = (1,1)$ et on effectue les comparaisons avec la formule $(x_1 \leq y_1)$ `ET` `NON`$(y_1 \leq x_1)$ `ET` `NON`$(y_2 \leq x_2)$ vraie si et seulement si $x \leq_N y$.

On pourrait aussi imaginer une méthode consistant pour un ordre donné à associer à chacun de ses éléments un vecteur de bits de même taille et générant un circuit booléen prenant en entrée deux vecteurs et donnant en sortie la réponse au test de comparaison. On pourrait alors chercher à optimiser la taille des vecteurs et différents paramètres du circuit booléen (taille, profondeur). Une telle méthode serait tout à fait appropriée dans le cadre du modèle de calcul parallèle par circuits booléens.

Pour ce genre de méthodes clairement non uniformes, l'espace de stockage des algorithmes est à ajouter à l'espace total de stockage.

D'autres approches

De nombreux articles font l'énumération de méthodes de codage connues, mais peu d'entre eux font un véritable effort de synthèse. Une véritable synthèse sur le sujet a été rédigée par Fall dans [Fal98]. Une version allégée de cette synthèse se trouve dans [Fal95a], et son article [Fal95b] développe une des idées présentée dans [Fal98] : l'utilisation de la décomposition modulaire ou d'autres décompositions pour construire de nouveaux codages (ces idées de décompositions se placent complètement dans le cadre du Partitionnement décrit au-dessus). Nous allons dire quelques mots sur cette synthèse de Fall.

Cette synthèse choisit de se placer à un haut niveau de formalisation avec l'objectif suivant : avoir un cadre commun de description des méthodes connues permettant de les analyser en distinguant d'une part une notion combinatoire de complexité de la méthode par rapport à l'ordre à coder et d'autre part les implantations de la méthode. Un des buts de cette distinction est de pouvoir comparer les méthodes en s'abstrayant le plus possible des implantations.

Pour cela, la synthèse part de l'hypothèse que les méthodes de codage d'ordres peuvent se décomposer en deux composantes. La première consiste à avoir pour tout ordre $P = (X, \leq_P)$, une technique de recouvrement de X par une famille de parties $\mathcal{S} = \{S_1, \ldots, S_k\}$ telle que l'application ϕ de X dans le treillis $I(\mathcal{S})$ (treillis des idéaux de l'ordre $(\mathcal{S}, \subseteq)$) définie par $\phi(x) = \{S_i \mid x \in S_i\}$, est un plongement d'ordre. C'est la composante "combinatoire" où on cherche des familles $\mathcal{S}$ avec certaines contraintes. Ensuite il s'agit de représenter au mieux les idéaux $\phi(x)$ pour implanter efficacement les tests d'inclusion entre ces idéaux.

En fait cette hypothèse est trop restrictive par rapport à la variété des approches pour le problème de codage. Toutes les méthodes que nous présentons dans ce chapitre ne rentrent pas dans le cadre proposé dans [Fal98] (prendre par exemple la *représentation par ensembles de successeurs immédiats* ou la *dimension booléenne*), ou alors elles y rentrent au prix d'une reformulation qui fait perdre de vue les vraies complexités de ces méthodes (par exemple si on cherche à tout prix à exhiber la fonction ϕ). Nous avons fait un choix différent en mettant en avant les critères d'espace, de temps d'exécution des opérations, de nombre d'instructions pour ces opérations, de temps de génération et de mise à jour des codages. Ces aspects de complexités sont peu traités dans [Fal98], seule la compression en espace est une conséquence sûre des optimisations qui y sont proposées. Nous avons choisi pour notre part tous ces critères car ils reflètent à la fois les efforts possibles au niveau de l'implantation (choix des opérations élémentaires utilisées, choix des structures de données) et les efforts possibles à un niveau plus combinatoire (décompositions de l'ordre, problèmes d'optimisation).

Cependant cette synthèse de Fall est riche sur plusieurs points :

- En s'abstrayant des aspects d'implantations, elle identifie clairement des techniques utiles pour coder les ordres, basées uniquement sur des notions de décomposition (à placer dans les techniques de Partionnement)
- Son étude des plongements d'ordre dans le treillis $I(\mathcal{S})$, donc dans un treillis booléen, fournit des résultats de complexité sur la méthode de *représentation par des ensembles* (*2-dimension*), notamment des résultats de $\mathcal{NP}$-complétude.
- Plusieurs "techniques d'implantation" y sont présentées de manière très claire (ce qui est appelé "technique d'implantation" dans [Fal98] est présenté ici comme "méthode de codage", il s'agit par exemple de la méthode de *partition en chaines* ou de *partition en antichaines*).

Chapitre 3

Une liste de méthodes de codage d'ordres

Nous allons présenter dans la section 3.2 plusieurs méthodes de codage sous forme de fiches techniques, un peu à la manière de Garey et Johnson. Ces méthodes sont essentiellement axées autour du problème de ***comparaison***. Pour commencer, la section 3.1 détaille le contenu de ces fiches.

3.1 Organisation des fiches

Pour chacune des méthodes, les fiches seront organisées autour des rubriques suivantes.

NOM (CLASSE) : nom de la méthode. Si cette méthode ne s'applique qu'à une classe particulière d'ordres, celle-ci est indiquée entre parenthèses.

CATEGORIE : catégorie(s) où la méthode se situe (voir section 2.6). Si la méthode est une généralisation ou une variante d'une autre méthode connue, cela sera indiqué ici.

AUTEURS : noms des auteurs à l'origine de la méthode. Les origines ne sont pas toujours claires, certaines méthodes ayant été redécouvertes indépendamment, dans différents contextes. Dans ce cas, on donnera ici une des références les plus significatives soit par la clarté de sa présentation de la méthode, soit pour le travail d'analyse qui y est effectué.

METHODE

DESCRIPTION : description des informations globales et locales, ainsi que de la technique utilisée pour traiter les tests de comparaison.

PARAMETRES : ce sont les paramètres importants de la méthode tels que la taille des informations locales ou d'autres paramètres influant sur l'espace de stockage et le traitement des opérations.

IMPLANTATIONS : implantations possibles pour le test de comparaison et performances, c'est à dire complexités de ce test.

L'espace total de stockage ou l'espace moyen par élément du codage (taille du codage) sont donnés en bits. Un exemple important est le ***stockage de k-uplets d'entiers*** $(x_1, \ldots, x_k)$. Si on sait que tous les entiers des k-uplets sont bornés par un entier n, alors on peut stocker un k-uplet dans un tableau de k mots mémoires (si $\lceil \log_2(n) \rceil \leq w$) pour pouvoir accéder aux entiers en temps constant. Si n est fixé et que $\lceil \log_2(n) \rceil \ll w$, alors on peut s'arranger pour découper les mots mémoires de taille w en sous-mots de taille $\lceil \log_2(n) \rceil$, tout en conservant un accès aux entiers en temps constant. C'est pourquoi on annoncera en général une complexité en espace en $\mathcal{O}(k\lceil \log_2(n) \rceil)$ bits pour chaque k-uplet (plutôt que $\mathcal{O}(kw)$). On parlera de la ***version tableau*** pour une méthode de codage stockant ses k-uplets d'entiers ainsi.

On peut encore compresser si on connait pour tout i, $1 \leq i \leq k$, n_i le maximum des x_i sur tous les k-uplets à stocker. Une fois les entiers n_i fixés, on peut accoler les entiers x_1, ..., x_k d'un k-uplet dans les mêmes mots mémoires, tout en conservant un accès à chaque entier en temps constant. Pour chaque k-uplet, la complexité en espace est alors en $\mathcal{O}(\sum_{1 \leq i \leq n} \lceil \log_2(n_i) \rceil)$ bits. On parlera de la ***version pseudo-tableau*** pour une méthode de codage stockant ses k-uplets d'entiers ainsi.

En général, il est plus facile de mettre en place la version tableau car il est plus facile de trouver et fixer un majorant commun à tous les entiers considérés. Remarquer aussi que pour un codage d'ordre fixé, par exemple pour une opération de comparaison, l'accès dans un tableau à un entier dont l'indice ne dépend que de l'ordre (et donc est fixé) sera traité plus rapidement que l'accès à entier dont l'indice dépend des éléments à comparer (même si avec notre jeu d'instructions, on reste en temps constant dans les deux cas).

OPTIMISATION

PARAMETRES : paramètres que l'on cherche à optimiser pour améliorer l'espace total de stockage et les performances des opérations. Si elle est connue, on donnera la complexité du calcul des optima.

LIENS : liens forts avec d'autres problèmes d'optimisation (dans la même classe de complexité, avec des transferts de ratios d'approximabilité ou de non-approximabilité).

BORNES : bornes sur les paramètres. Eventuellement comportement en moyenne des paramètres pour une distribution sur les ordres en général ou sur certaines classes d'ordres.

GENERATION

On donne ici des algorithmes de génération du codage (a priori à partir des trois structure de données classiques données dans la section 2.2). Pour beaucoup de méthodes, l'optimisation des paramètres correspond à des problèmes $\mathcal{NP}$-complets et il s'agit alors d'heuristiques de génération.

Pour chaque algorithme, on donne les complexités de la génération du codage (à partir d'une des trois structures de données classiques : BM pour la matrice binaire, TC pour les listes de successeurs, TR pour les listes de successeurs immédiats), ainsi que les résultats d'optimisation (optima atteints ou distances aux optima).

Dans certains cas, on aborde aussi la reconnaissance de classes d'ordres obtenues en fixant certains paramètres de la méthode de codage.

REFERENCES

Les références des principaux travaux étudiant la méthode de codage, avec en particulier ceux à l'origine de la méthode.

Les rubriques suivantes sont **en option**, suivant ce qui existe dans la littérature.

MISES A JOUR

Ce sont les aspects dynamiques de la méthode ou comment mettre à jour le codage après ajout/suppression d'éléments/comparaisons. En général, c'est plutôt l'apect incrémental (ajout d'un élément) qui a été étudié.

VARIANTES

On exposera des variantes de la méthode : soit une autre méthode très proche, soit la même mais avec d'autres contraintes sur les paramètres, par exemple pour pouvoir traiter d'autres opérations que la comparaison. C'est le cas quand on imposera à un plongement d'ordre d'être un $\vee$-plongement pour pouvoir traiter efficacement l'opération de borne supérieure.

ATOUTS & INCONVENIENTS

Caractéristiques intéressantes ou défauts de la méthode par rapport à ses performances ou par rapport à d'autres opérations à traiter sur les ordres. Premières comparaisons avec d'autres méthodes.

Certains points à compléter ou problèmes ouverts seront signalisés par un “ ?”.

3.2 Quelques méthodes de codage

L'ordre à coder est $P = (X, \leq_P)$, n désigne le nombre d'éléments de P, $| <_P |$ le nombre de relations de comparabilités autrement dit $\sum_{x \in X} |Succ_P(x)|$, $| \leq_P |$ le nombre de comparabilités plus le nombre d'éléments autrement dit $n + \sum_{x \in X} |Succ_P(x)|$, $| \prec_P |$ le nombre d'arcs de la réduction transitive autrement dit $\sum_{x \in X} |ImSucc_P(x)|$.

Par rapport au modèle Word-RAM considéré pour les implantations, on supposera que tous les entiers x que nous auront à stocker vérifieront $\lceil \log_2(x) \rceil \leq w$, pour qu'ils tiennent sur un seul mot mémoire. En particulier, on supposera que $\lceil \log_2(n) \rceil \leq w$.

La notation $[a, b]$ où a et b sont deux entiers, désignera toujours l'intervalle d'entiers $\{i \mid a \leq i \leq b\}$.

Une des techniques les plus utilisées pour parcourir les éléments d'un ordre consiste à suivre une extension linéaire de l'ordre du plus petit élément vers le plus grand (parcours ***extension linéaire***) ou du plus grand vers le plus petit (***extension linéaire inversée***, qui revient à une extension linéaire du dual). On rappelle que le calcul d'une extension linéaire de P se fait en temps $\mathcal{O}(n + | \prec_P |)$ à partir de TR, $\mathcal{O}(| \leq_P |)$ à partir de TC et $\mathcal{O}(n^2)$ à partir de BM (aussi appelé ***tri topologique*** [CLR94]).

Nous illustrons chacune des méthodes par un codage pour l'ordre de la figure 3.2. Noter que le codage proposé ne sera pas nécessairement optimal pour les paramètres de la méthode considérée. Les dessins sont inspirées de ceux de l'article de Zibin et Gil [ZG01], où le lecteur pourra trouver d'autres exemples d'illustrations.

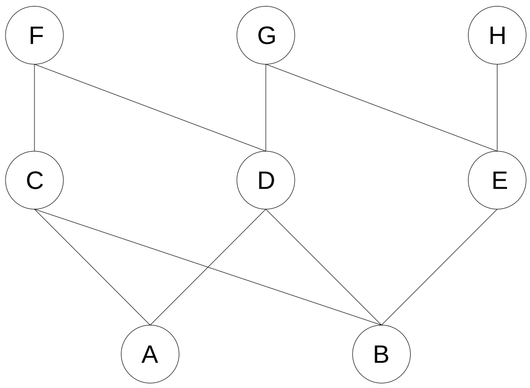
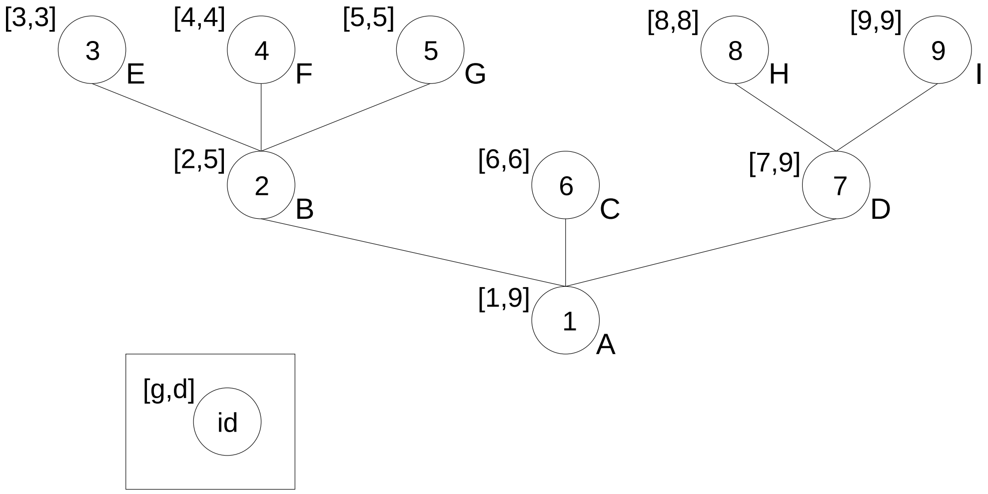


FIG. 3.1 – L'ordre utilisé pour illustrer les méthodes de codage

Les méthodes présentées sont les suivantes :

1. ***Relative Numbering***, ou ***méthode de Schubert***, pour la classe des arbres.
2. ***Range Compression***.
3. ***Union d'ordres d'intervalles*** (et en variante, ***Intersection d'ordres d'intervalles***).
4. ***Algorithme de Cohen*** pour la classe des arbres.
5. ***Partition en antichaines***.
6. ***Partition en chaines***.
7. ***PQ-Encoding***.
8. ***Dimension***.
9. ***2-Dimension*** ou ***Représentation par ensembles***.
10. ***E-Dimension***
11. ***Dimension booléenne***.
12. ***Inclusion de figures géométriques***.
13. ***Représentation par union d'intervalles***.

Relative Numbering (méthode de Schubert)



NOM : Relative Numbering (ARBRES)

CATEGORIE : Compression d'ensembles.

AUTEURS : Schubert *et al* [SPT83].

METHODE

- ▷ **DESCRIPTION :** Numéroter les éléments de 1 à n tel que pour tout $x \in X$, en notant id_x le numéro de x, l'ensemble $\{id_y \mid y \in Succ_P(x) \cup \{x\}\}$ soit un intervalle $[g_x, d_x]$.
  Test de comparaison : $x \leq_P y$ si et seulement si $g_x \leq id_y \leq d_x$.
- ▷ **PARAMETRES :** Aucun.
- ▷ **IMPLANTATIONS :** Chaque élément x reçoit les trois entiers id_x, g_x, d_x. Taille du codage en $\mathcal{O}(\lceil \log_2(n) \rceil)$. Test en temps constant.

OPTIMISATION

- ▷ **PARAMETRES :** Rien à optimiser.
- ▷ **LIENS :**
- ▷ **BORNES :**

GENERATION

Il suffit de faire un parcours en profondeur de l'arbre et de numéroter les éléments selon un ordre préfixe (ou postfixe). Génération en temps et espace $\mathcal{O}(n)$ à partir de TR.

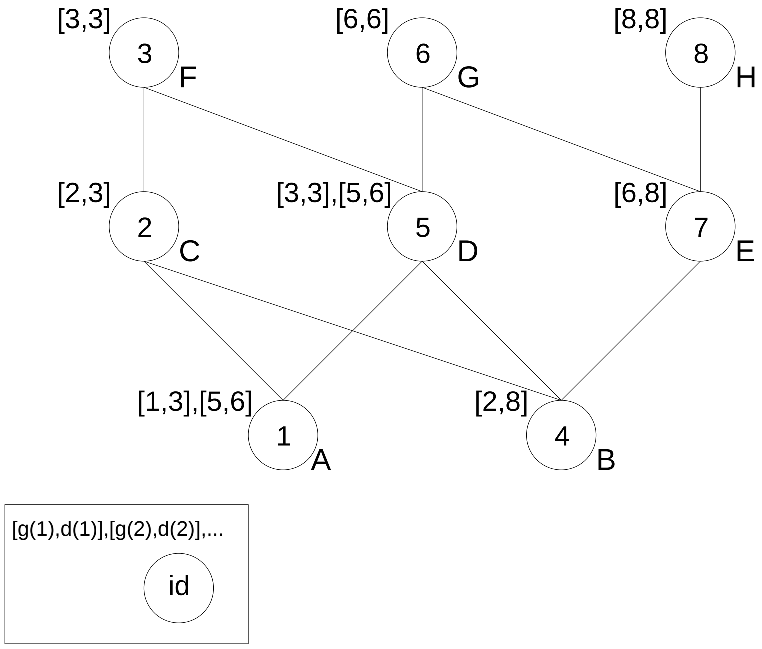
ATOUTS & INCONVENIENTS

Méthode optimale pour la classe des arbres (il y en a d'autres pour les arbres, par exemple le codage associé à la *dimension*).

REFERENCES

Connu aussi sous le nom de codage de Schubert [SPT83].

$\boxed{\text{Range Compression}}$



NOM : Range Compression.

CATEGORIE : Compression d'ensembles. Généralisation du Relative Numbering.

AUTEURS : Agrawal *et al* [ABJ89a, ABJ89b].

METHODE

▷ **DESCRIPTION :** Numéroter les éléments, puis en notant id_x le numéro de x, réécrire l'ensemble $\{id_y \mid y \in Succ_P(x) \cup \{x\}\}$ sous forme d'une union de k_x intervalles $[g_x(1), d_x(1)] \cup \ldots \cup [g_x(k_x), d_x(k_x)]$. Par exemple $\{1, 2, 4, 5, 6, 9\}$ devient $[1, 2] \cup [4, 6] \cup [9, 9]$.
Test de comparaison : $x \leq_P y$ si et seulement si $g_x(i) \leq id_y \leq d_x(i)$ pour un indice i, $1 \leq i \leq k_x$.

▷ **PARAMETRES :**
- $MaxInt$ = nombre maximum d'intervalles pour un élément.
- $NbInt$ = nombre total d'intervalles pour tout l'ordre, c.à.d. $\sum_{x \in X} k_x$.

▷ **IMPLANTATIONS :** Chaque élément x reçoit un tableau de $2k_x$ entiers ($g_x(1)$,$d_x(1)$,...,$g_x(k_x)$,$d_x(k_x)$), son nombre d'intervalles k_x et son numéro id_x. Espace total de stockage en $\mathcal{O}(NbInt.\lceil \log_2(n) \rceil)$. Test en temps $\mathcal{O}(\log_2(MaxInt))$ au pire (tableaux triés), et en temps moyen $\mathcal{O}(\log_2(NbInt/n))$ (si les tests sont équiprobables). Analyse plus poussée du temps moyen dans [ABJ89b].

OPTIMISATION

▷ **PARAMETRES :**
- Minimiser $NbInt$. Problème d'optimisation $\mathcal{NP}$-complet (réduction à partir de Consecutive Block Minimization [ABJ89b]). Approximations ?
- Minimiser $MaxInt$. Le minimum est noté $range(P)$. Problème $\mathcal{NP}$-complet ?

▷ **LIENS :** Nombre d'intervalles $i(P)$ de la représentation par union d'intervalles.

▷ **BORNES :** ?

GENERATION

▷ Labeling Algorithm [ABJ89b] : algorithme suivant la stratégie qui consiste à construire une réduction T de P qui soit un arbre (ajouter un minimum à P si nécessaire) puis à numéroter les éléments selon un *Relative Numbering* de T. Implanté alors en associant à chaque $x \in X$ l'intervalle $[g_x, d_x]$ des numéros de x et ses successeurs dans T, puis en propageant à chaque $x \in X$ les intervalles de ses successeurs (en prenant que ceux maximaux pour l'inclusion). L'algorithme de [ABJ89b] est optimal (minimise $NbInt$) pour cette stratégie.

▷ Reconnaissance des ordres tels que $MaxInt = 1$ équivalente à déterminer si $MaxInt = 1$ pour le biparti $B(P)$, c.à.d. si c'est un biparti convexe. Reconnaissance et bonne numérotation en temps et espace $\mathcal{O}(| \leq_P |)$ à partir de TC [BL76, Pau98]. Noter que les bipartis convexes sont des ordres de contiguïté [BHJR95]. La classe des ordres tels que $MaxInt = 1$ est strictement incluse dans la classe des ordres de *dimension* ≤ 2.

VARIANTES

▷ Variante en remplaçant $Succ_P(x) \cup \{x\}$ par $Succ_P(x)$. Le test devient : $x \leq_P y$ si et seulement si $id_x) = id_y$ ou $g_x(i) \leq id_y \leq d_x(i)$ pour un indice i, $1 \leq i \leq k_x$. Dans ce cas, reconnaissance des ordres tels que $MaxInt = 1$ équivalente à déterminer si $B^0(P)$ est un biparti convexe.

▷ Variante en remplaçant $Succ_P(x)$ par $Pred_P(x)$.

ATOUTS & INCONVENIENTS

Remarquer que pour cette méthode, un ordre ne se comporte pas comme son dual : valeurs différentes des paramètres, par exemple la classe des ordres tels que $MaxInt = 1$ n'est pas stable par passage au dual, et plus généralement $|range(P) - range(P^d)|$ n'est pas borné par une constante (considérer par exemple les arbres ternaires complets et leurs duals). Redonne la méthode de *Schubert* dans le cas des arbres.

REFERENCES

Méthode proposée et analysée par Agrawal *et al* [ABJ89a, ABJ89b].

NOM : Union d'ordres d'intervalles.

CATEGORIE : Représentation par union.

AUTEURS : Capelle [Cap94].

METHODE

▷ **DESCRIPTION :** Construire une famille $\{I_1, \dots, I_k\}$ d'ordres d'intervalles sur X tels que $x \leq_P y$ si et seulement si $x \leq_{I_1} y$ ou ... ou $x \leq_{I_k} y$. Pour chaque $x \in X$, $[g_x(i), d_x(i)]$ est l'intervalle associé à x dans I_i.
Test de comparaison : $x \leq_P y$ si et seulement si $x \leq_{I_i} y$ pour un indice i, $1 \leq i \leq k$ (c.à.d. $d_x(i) \leq g_y(i)$).

▷ **PARAMETRES :** $NbInt$ = le nombre d'ordres d'intervalles de la famille.

▷ **IMPLANTATIONS :** Chaque élément x reçoit un tableau de $2k$ entiers $(g_x(1),d_x(1),...,g_x(k),d_x(k))$. Taille du codage en $\mathcal{O}(NbInt\lceil\log_2(n)\rceil)$. Test en temps $\mathcal{O}(NbInt)$.

OPTIMISATION

▷ **PARAMETRES :** Minimiser $NbInt$, le minimum est noté $ch(P)$. Problème $\mathcal{NP}$-complet dans le cas des ordres bipartis (réduction à partir de la *dimension* d'un ordre [Yan82]).

▷ **LIENS :** Dimension d'un ordre.

▷ **BORNES :** $dim(P) \geq ch(\overline{B}(P)))$

GENERATION

▷ Algorithme IRR [Cap94] : recouvrement glouton des comparabilités par une succession de réductions intervallaires construites sur la base de l'algorithme de reconnaissance des ordres d'intervalles de Baldy et Morvan [BM93]. Donne bien $NbInt = 1$ si P est un ordre d'intervalles.

VARIANTES

▷ Représentation par intersection d'ordres d'intervalles : nombre minimum d'ordres dans la famille appelé *dimension intervallaire*, noté $idim(P)$ [FHM94] (problème d'optimisation $\mathcal{NP}$-complet [Yan82], réduction à partir de la *dimension*, inégalité claire $idim(P) \leq dim(P)$, égalités $dim(P) = idim(B(P))$ et $idim(P) = idim(B^0(P)) = ch(B^0(P))$).

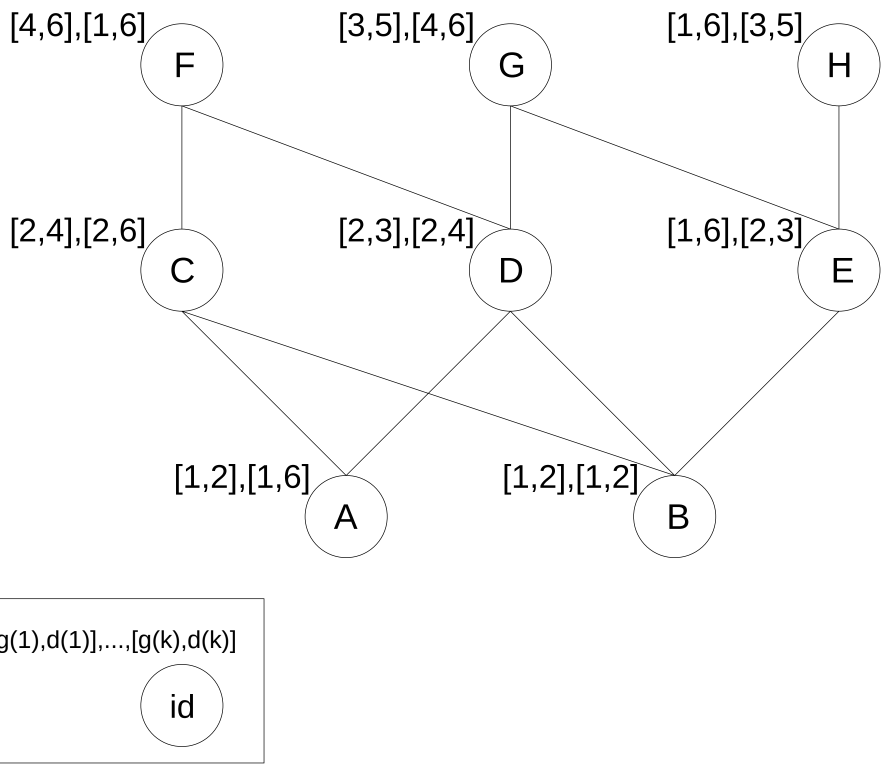
ATOUTS & INCONVENIENTS

D'après les expérimentations, bons taux de compression en espace (petit $NbInt$) pour les ordres denses ($| <_P | = \Theta(n^2)$) [Cap94].

REFERENCES

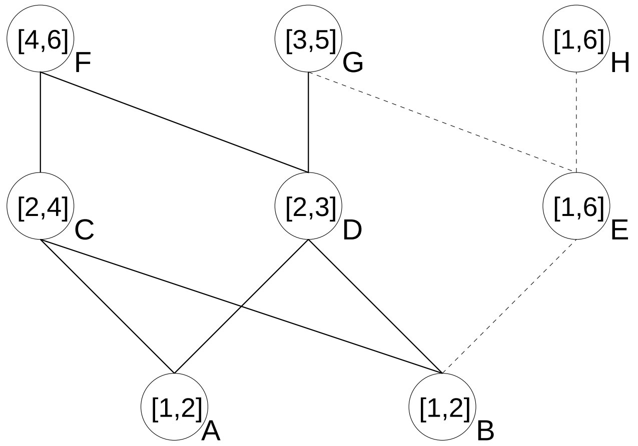
Méthode présentée par Capelle [Cap94]. Etude des extensions et réductions intervallaires d'un ordre par Felsner, Gusted et Morvan [FGM99]. Etude de la dimension intervallaire par Felsner, Habib et Möhring [FHM94].

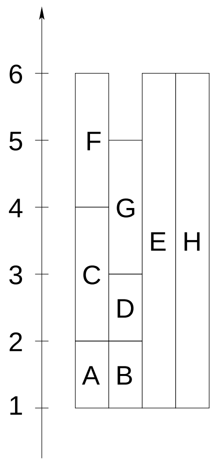
Union d'ordres d'intervalles



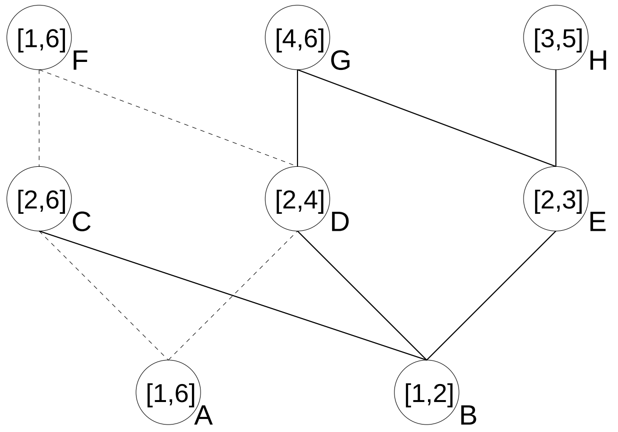
Union d'ordres d'intervalles

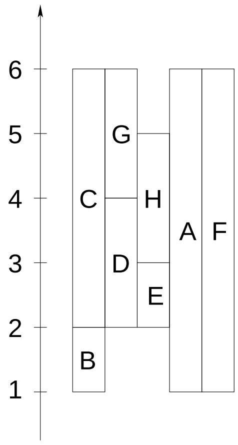
Première réduction intervallaire I_1



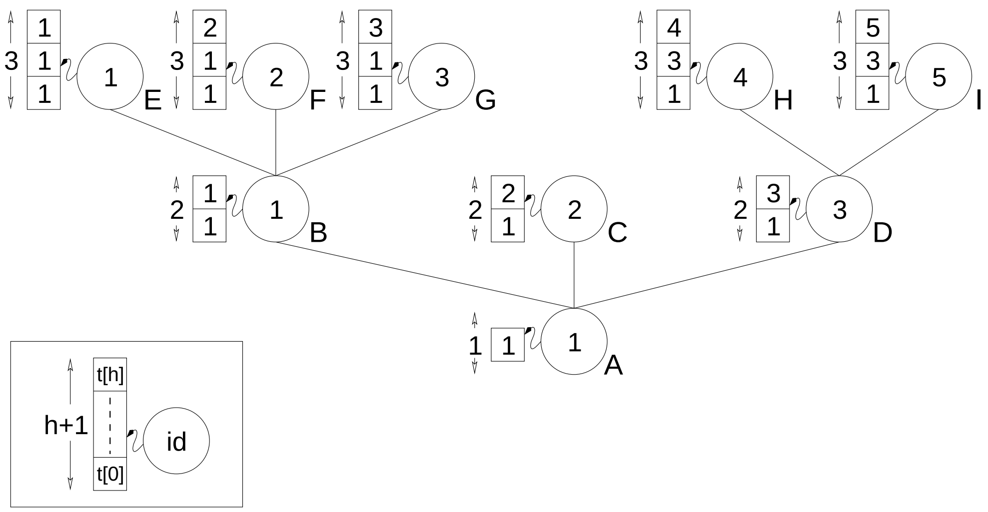


Deuxième réduction intervallaire I_2





Algorithme de Cohen



NOM : Algorithme de Cohen (ARBRES)

CATEGORIE : Partitionnement.

AUTEURS : Cohen [Coh91].

METHODE

- ▷ **DESCRIPTION :** Partitionner P en ses $h(P)+1$ niveaux $S_0,\ldots,S_{h(P)}$ de cardinal $n_0,\ldots,n_{h(P)}$ (S_i est l'ensemble des éléments de hauteur i). Pour tout i, $0 \leq i \leq h(P)$, numéroter les éléments de S_i de 1 à n_i. Associer à chaque $x \in X$ son numéro id_x, sa hauteur h_x et le $h(x)+1$-uplet $(t_x[0],\ldots,t_x[h_x])$ tel que $t_x[i] = id_y$ avec $\{y\} = S_i \cap \downarrow_P x$.
  Test de comparaison : $x \leq_P y$ si et seulement si $h_x \leq h_y$ et $t_y[h_x] = id_x$.
- ▷ **PARAMETRES :** Aucun.
- ▷ **IMPLANTATIONS :** Chaque élément x reçoit les entiers id_x et h_x et un tableau avec $(t_x[0],\ldots,t_x[h_x])$. Espace total de stockage en $\mathcal{O}(n.h(P).\lceil\log_2(n)\rceil)$. Test en temps constant.

OPTIMISATION

- ▷ **PARAMETRES :** Rien à optimiser
- ▷ **LIENS :**
- ▷ **BORNES :**

GENERATION

Construction avec un parcours de l'arbre (profondeur ou largeur) à partir de TR. Génération en temps $\mathcal{O}(n.h(P))$ (pour remplir les tableaux).

ATOUTS & INCONVENIENTS

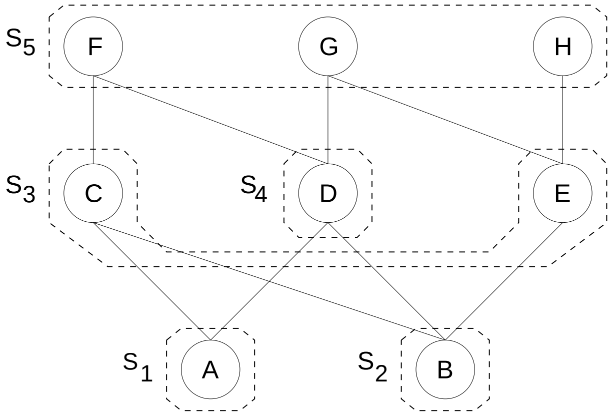
Contrairement à la méthode de *Schubert*, ce n'est pas une méthode optimale pour la classe des arbres.

REFERENCES

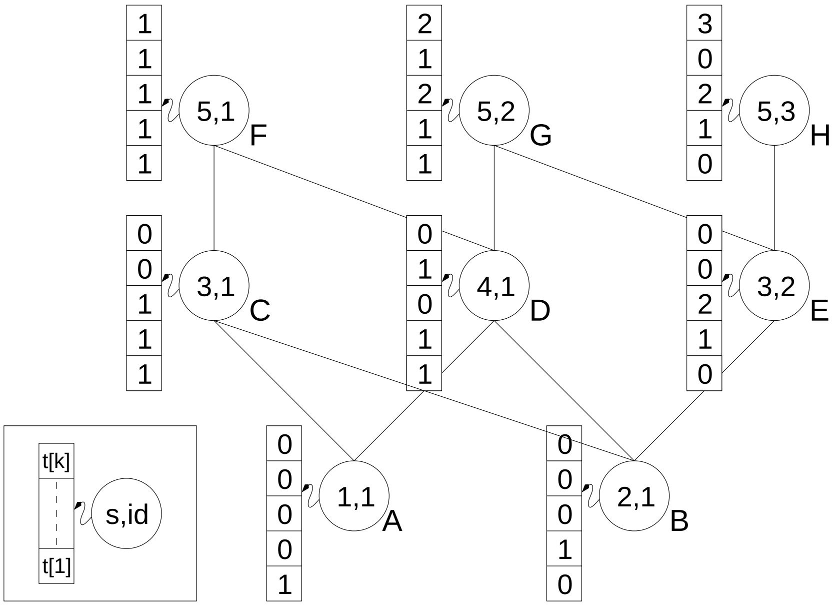
Méthode proposée par Cohen [Coh91]. Peut être vu comme une variante de la méthode de “displays” de Dijkstra [Dij60].

$\boxed{\text{Partition en Antichaines}}$

Partition de X en $\{S_1, S_2, S_3, S_4, S_5\}$



Codage correspondant à cette partition



NOM : Partition en Antichaines.

CATEGORIE : Partitionnement. Généralisation de l'algorithme de Cohen.

AUTEURS : Fall [Fal98].

METHODE

▷ **DESCRIPTION :** Partitionner P en k parties $S_1,\ldots,S_k$ de cardinal $n_1,\ldots,n_k$ telles que pour tout $x \in X$ et $1 \leq i \leq k$, $|S_i \cap \downarrow_P x| \leq 1$ (implique que tous les S_i sont des antichaines). Pour tout i, $1 \leq i \leq k$, numéroter les éléments de S_i de 1 à n_i. Associer à chaque $x \in X$ son numéro id_x, le numéro s_x de la partie à laquelle il appartient et le k-uplet d'entiers $(t_x[1], \ldots, t_x[k])$ tel que si $S_i \cap \downarrow_P x = \{y\}$, alors $t_x[i] = id_y$, sinon $S_i \cap \downarrow_P x = \emptyset$ et alors $t_x[i] = 0$.
Test de comparaison : $x \leq_P y$ si et seulement si $t_y[s_x] = id_x$.

▷ **PARAMETRES :**
- $NbSet$ = nombre de parties dans la partition.
- $Taille$ = nombre de bits d'informations pour un élément (en négligeant id_x et s_x), soit $\sum_{1 \leq i \leq k} \lceil \log_2(n_i) \rceil$.

▷ **IMPLANTATIONS :** Chaque élément x reçoit les entiers id_x et s_x et un tableau avec $(t_x[1], \ldots, t_x[NbSet])$. Test en temps constant.
- Version tableau : taille du codage en $\mathcal{O}(NbSet.\lceil \log_2(n) \rceil)$.
- Version pseudo-tableau : taille du codage en $\mathcal{O}(Taille)$.

OPTIMISATION

▷ **PARAMETRES :**
- Minimiser $NbSet$. Equivalent à colorer le graphe de conflit sur X avec x adjacent à y si et seulement si $\uparrow_P x \cap \uparrow_P y \neq \emptyset$ (chaque couleur correspond à un S_i). Problème d'optimisation $\mathcal{NP}$-complet [Fal98], non-approximable comme la coloration de graphe.
- Minimiser $Taille$. Problème $\mathcal{NP}$-complet ?

▷ **LIENS :** Coloration de graphe.

▷ **BORNES :** $NbSet \geq \max_{x \in X} |\downarrow_P x| \geq h(P) + 1$.

GENERATION

▷ Packed Encoding [VHK97] (retranscrit, car la version de l'article correspond à la variante présentée en dessous) : Partitionne l'ordre en deux : $A = \{x \in X | \uparrow_P x \text{ est un arbre}\}$ et $B = X \backslash A$. Puis partition de l'ordre par une coloration gloutonne des éléments de B selon l'inverse d'une extension linéaire, suivie d'une coloration gloutonne des éléments de A selon une extension linéaire. Essaie de tirer profit des sous-ordres qui sont des arbres.

- ▷ Bit-Packed Encoding [VHK97] : Version pseudo-tableau du Packed Encoding.

VARIANTES

- ▷ Remplacer les $\downarrow_P$ par des $\uparrow_P$, et inversement.

ATOUTS & INCONVENIENTS

Pour cette méthode, un ordre ne se comporte pas comme son dual : par exemple pour un arbre P, $NbSet$ minimum vaut $h(P)+1$, alors que pour son dual, l'anti-arbre P^d, $NbSet$ minimum vaut n.

REFERENCES

Présentation claire de la méthode par Fall [Fal98]. Heuristique de génération proposée par Vitek *et al* [VHK97].

NOM : Partition en Chaines.

CATEGORIE : Partitionnement. Peut aussi être vu comme un plongement dans un treillis.

AUTEURS : Bouchet [Bou71].

METHODE

▷ **DESCRIPTION :** Partitionner P en k chaines $C_1,\ldots,C_k$ avec respectivement $n_1,\ldots,n_k$ éléments. Pour chaque chaine C_i, numéroter de 1 à n_i les éléments de la chaine dans l'ordre croissant. En notant id_x le numéro reçu par x, associer à chaque $x \in X$ le k-uplet d'entiers $(t_x[1],\ldots,t_x[k])$ où $t_x[i] = \max\{id_y \mid y \leq_P x \text{ et } y \in C_i\}$, avec $\max(\emptyset) = 0$. Associer aussi à x le numéro c_x de la chaine le contenant.
Test de comparaison :
- Version Partitionnement : $x \leq_P y$ si et seulement si $t_y[c_x] \geq id_x$.
- Version Plongement : $x \leq_P y$ si et seulement si $t_x[i] \leq t_y[i]$ pour tout i, $1 \leq i \leq k$.

▷ **PARAMETRES :**
- $NbChaines$ = nombre de chaines de la partition.
- $Taille$ = nombre de bits d'informations pour un élément, soit $\sum_{1\leq i\leq k} \lceil \log_2(n_i+1) \rceil$.

▷ **IMPLANTATIONS :** Chaque élément x reçoit les entiers id_x et c_x et un tableau avec $(t_x[1],\ldots,t_x[NbChaines])$. Deux versions pour la complexité en espace et deux versions pour la complexités en temps :
- Version tableau : taille du codage en $\mathcal{O}(NbChaines.\lceil \log_2(n) \rceil)$.
- Version pseudo-tableau : taille du codage en $\mathcal{O}(Taille)$.
- Version Partitionnement : test en temps constant.
- Version Plongement : test en temps $\mathcal{O}(NbChaines)$.

OPTIMISATION

▷ **PARAMETRES :**
- Minimiser $NbChaines$, le minimum est la largeur $w(P)$ (théorème Min-Max de Dilworth [Dil50]).
- Minimiser $Taille$. Problème d'optimisation polynomial ?

▷ **LIENS :**

▷ **BORNES :** $NbChaines \geq w(P)$.

GENERATION

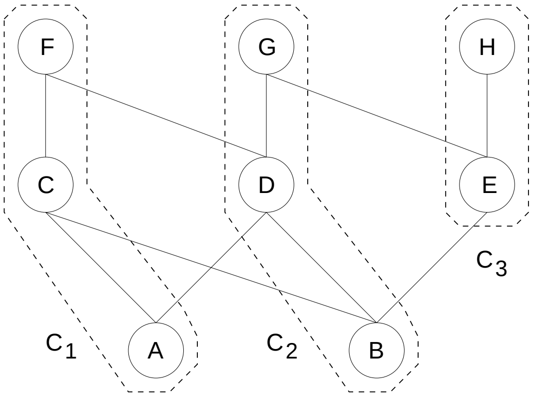
Calcul d'une partition en $w(P)$ chaines à partir d'un couplage maximum de $B^0(P)$, en temps $\mathcal{O}(n^3)$ avec un algorithme simple à base de séquences alternées [BFK90] ou en temps $\mathcal{O}(n^{1/2}| \leq_P |)$ avec l'algorithme de Hopcroft et Karp [HK73].

REFERENCES

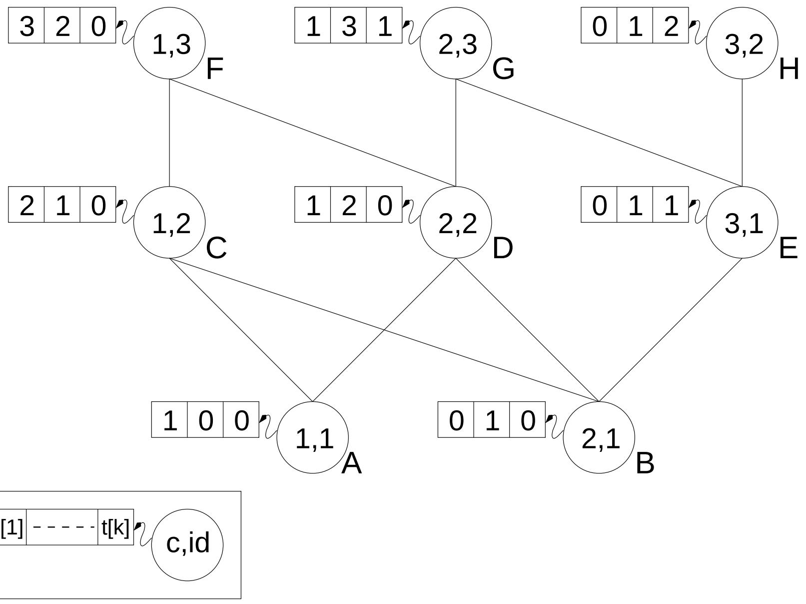
Présentation claire de la méthode par Fall [Fal98]. Utilisée par différents auteurs [Jag90, Meh84, Sim88]. Utilisée dans les systèmes distribués [Mat89, Bal94, Jou95] Déjà présente dans la thèse de Bouchet [Bou71]. Pour un rappel sur le théorème de Dilworth et les algorithmes de partionnement en $w(P)$ chaines, voir par exemple l'article de Felsner, Raghavan et Spinrad [FRS99].

$\boxed{\text{Partition en Chaines}}$

Partition de X en $\{C_1, C_2, C_3\}$

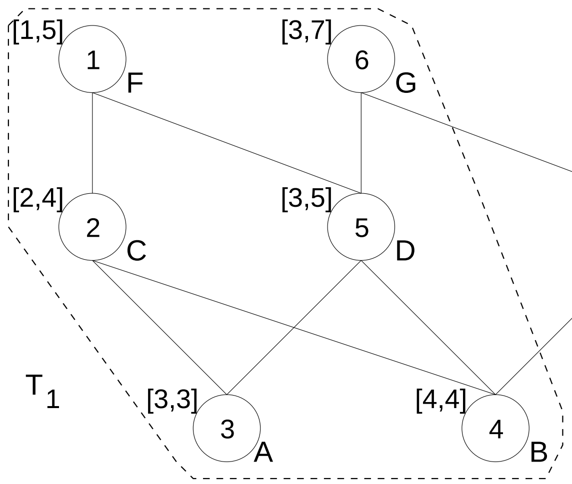


Codage correspondant à cette partition

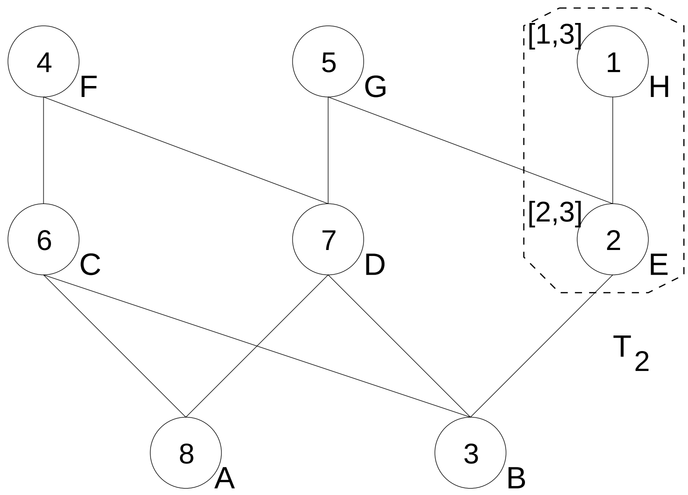


$\boxed{\text{PQ-Encoding}}$

Bonne numérotation de X pour la partie T_1



Bonne numérotation de X pour la partie T_2



NOM : PQ-Encoding.

CATEGORIE : Partitionnement

AUTEURS : Zibin et Gil [ZG01].

METHODE

- ▷ **DESCRIPTION :** Partitionner P en l parties T_1,…,T_l en sorte que pour tout $1 \leq j \leq l$, il existe une numérotation de 1 à n des ééments de X telle que pour tout $x \in T_j$, en notant $t_y[j]$ le numéro donné à $y \in X$, l'ensemble $\{t_y[j] \mid y \in \downarrow_P x\}$ est un intervalle $[g_x, d_x]$. Associer à chaque $x \in X$ le numéro s_x de la partie à laquelle il appartient, les entiers g_x et d_x, et le l-uplet d'entiers $(t_x[1], \ldots, t_x[l])$.
  Test de comparaison : $x \leq_P y$ si et seulement si $g_y \leq t_x[s_y] \leq d_y$.
- ▷ **PARAMETRES :** $NbSet$ = nombre de parties dans la partition.
- ▷ **IMPLANTATIONS :** Chaque élément x reçoit les entiers s_x, g_x et d_x et un tableau avec $(t_x[1], \ldots, t_x[NbSet])$. Taille du codage en $\mathcal{O}(NbSet.\lceil \log_2(n) \rceil)$. Test en temps constant.

OPTIMISATION

- ▷ **PARAMETRES :** Minimiser $NbSet$. Problème d'optimisation $\mathcal{NP}$-complet ?
- ▷ **LIENS :** ?
- ▷ **BORNES :** ?

GENERATION

- ▷ Algorithme de Zibin et Gil [ZG01] : construction gloutonne de la partition à partir d'un algorithme linéaire de reconnaissance des bipartis convexes basé sur les PQ-trees [BL76].
- ▷ Reconnaissance des ordres tels que $NbSet = 1$ équivalente à déterminer si $B(P)$ est un biparti convexe (voir la méthode de *Range compression*).

VARIANTES

- ▷ Remplacer les $\downarrow_P$ par des $\uparrow_P$, et inversement.

ATOUTS & INCONVENIENTS

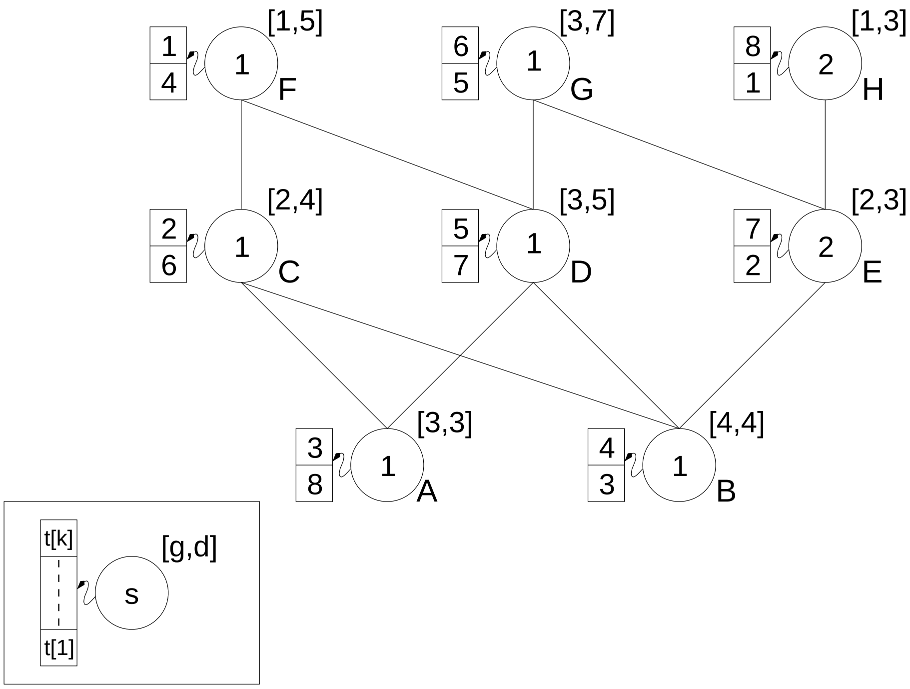
Remarquer que pour cette méthode, un ordre ne se comporte pas comme son dual (par exemple la classe des ordres tels que $MaxInt = 1$ n'est pas stable par passage au dual). Méthode optimale pour les anti-arbres : la variante redonne la méthode de *Schubert* dans le cas des arbres. Très bonnes performances sur des ordres issus des langages orientés objets [ZG01].

REFERENCES

Méthode proposée par Zibin et Gil [ZG01] (avec des techniques supplémentaires d'implantation pour améliorer les performances dans le contexte des langages orientés objets).

PQ-Encoding

Codage correspondant aux numérotations présentées pour T_1 et T_2



NOM : Dimension.

CATEGORIE : Représentation par intersection. Peut aussi être vu comme plongement dans un treillis.

AUTEURS : Dushnik et Miller [DM41].

METHODE

- $\triangleright$ **DESCRIPTION :** Construire une famille $\{L_1, \ldots, L_k\}$ de k extensions linéaires de P dont l'intersection redonne P. Associer à chaque $x \in X$ le k-uplet d'entiers $(v_x[1], \ldots, v_x[k])$ où $v_x[i]$ désigne la hauteur de x plus 1 dans L_i.
  Test de comparaison : $x \leq_P y$ si et seulement si $v_x[i] \leq v_y[i]$ pour tout i, $1 \leq i \leq k$.
  Equivalent à un plongement d'ordre dans le produit de k chaines $[1, n] \times \ldots \times [1, n]$.
- $\triangleright$ **PARAMETRES :** $NbExt$ = nombre d'extensions linéaires dans la famille.
- $\triangleright$ **IMPLANTATIONS :** Chaque élément x reçoit le tableau d'entiers $(v_x[1], \ldots, v_x[NbExt])$. Taille du codage en $\mathcal{O}(NbExt.\lceil\log_2(n)\rceil)$. Test en temps $\mathcal{O}(NbExt)$.

OPTIMISATION

- $\triangleright$ **PARAMETRES :** Minimiser $NbExt$, le minimum est appelé *dimension* de P, noté $dim(P)$. Problème d'optimisation $\mathcal{NP}$-complet : pour $d \geq 3$, tester si $dim(P) \leq d$ est $\mathcal{NP}$-complet [Yan82]. Approximations ?
- $\triangleright$ **LIENS :** Coloration d'hypergraphe [Tro91].
- $\triangleright$ **BORNES :**
  - Bornes supérieures : $dim(P) \leq n/2$ et $dim(P) \leq w(P)$ [Hir55].
  - Comportement en moyenne : soit X un ensemble fixé à n éléments, soit $\mathcal{PO}(X)$ l'ensemble des ordres sur X muni de la distribution uniforme, alors il existe $c_1, c_2 > 0$ telles que $(1 - c_1/\log_2(n))n/4 < dim(P) < (1 - c_2/\log_2(n))n/4$ pour presque tous les ordres $P \in \mathcal{PO}(X)$ [EKT91].

GENERATION

- $\triangleright$ Reconnaissance et codage des ordres P tels que $dim(P) = 2$ en temps et espace $\mathcal{O}(| \leq_P |)$ à partir de TC [MS97].

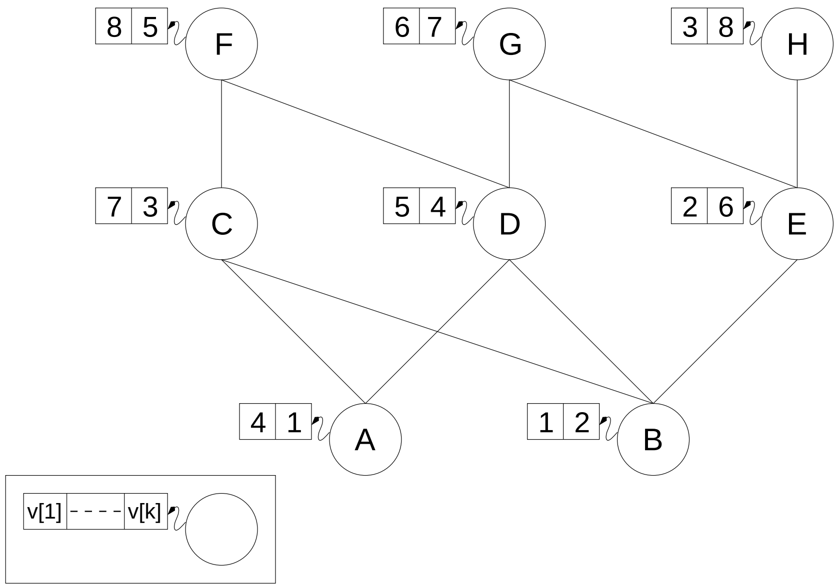
ATOUTS & INCONVENIENTS

De nombreuses classes particulières d'ordres ont leur dimension bornée par une constante (voir [Tro91] pour des exemples).

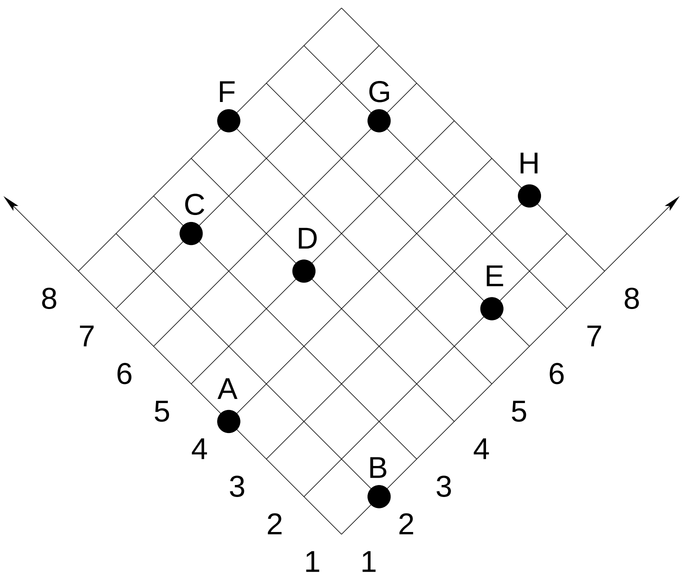
REFERENCES

Définition de la dimension par Dushnik et Miller [DM41]. Equivalence avec les plongements dans un produit de chaines par Ore [Ore62]. Livre de référence par Trotter [Tro91]. Listes de problèmes ouverts d'algorithmique sur la dimension par Spinrad [Spi94, Spi].

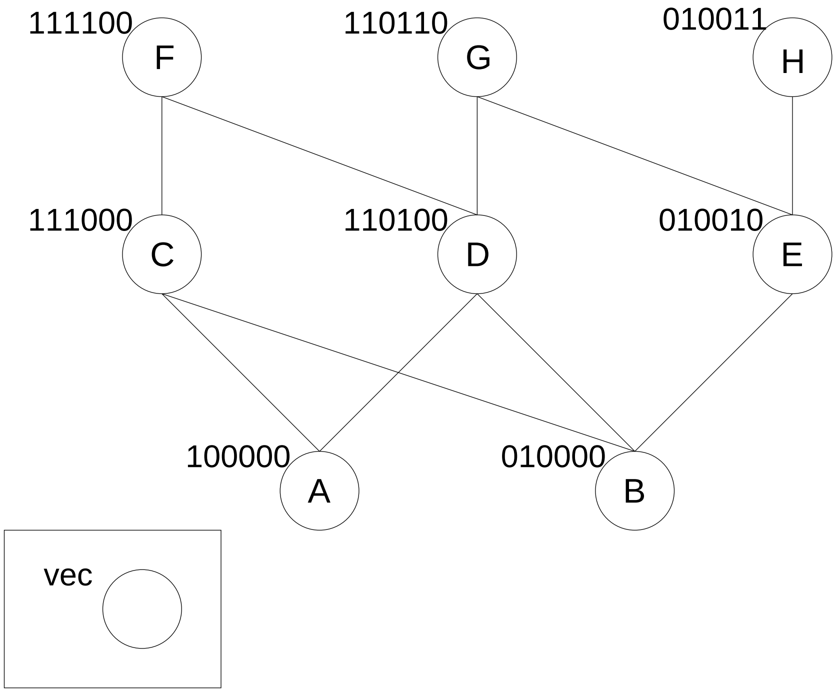
$\boxed{\text{Dimension}}$



Codage vu comme un plongement d'ordre dans $[1, 8] \times [1, 8]$



2-Dimension


111100
F
110110
G
010011
H
111000
C
110100
D
010010
E
100000
A
010000
B
vec

NOM : Représentation par ensembles, 2-Dimension.

CATEGORIE : Plongement dans un treillis.

AUTEURS : Novak [Nov63].

METHODE

- ▷ **DESCRIPTION :** Construire un plongement d'ordre ϕ de P dans un treillis booléen $\underline{2}^S$ où S est un ensemble fini. Associer à chaque $x \in X$ l'ensemble $\phi(x) \subseteq S$
  Test de comparaison : $x \leq_P y$ si et seulement si $\phi(x) \subseteq \phi(y)$.
- ▷ **PARAMETRES :** $Taille$ = le cardinal de S.
- ▷ **IMPLANTATIONS :** En prenant $S = \{1, \ldots, Taille\}$, chaque élément x de X reçoit un vecteur de bits $vec_x = (v_x[1], \ldots, v_x[Taille])$ où $v_x[i] = 1$ si $i \in \phi(x)$ et $v_x[i] = 0$ sinon. Taille du codage en $\mathcal{O}(Taille)$. Test en temps $\mathcal{O}(Taille/w)$ en testant si vec_x `OU` $vec_y = vec_y$ avec l'opération bit à bit `OU` sur ces vecteurs de bits.

OPTIMISATION

- ▷ **PARAMETRES :** Minimiser $Taille$, le minimum est appelé *2-dimension*, noté $dim_2(P)$. Problème d'optimisation $\mathcal{NP}$-complet, non-approximable à une constante près (voire pire) à moins que $\mathcal{P}$=$\mathcal{NP}$ (voir Chapitre 5).
- ▷ **LIENS :** Recouvrement d'un biparti par des bipartis complets, coloration de graphe, base d'une famille d'ensembles (voir Chapitre 5).
- ▷ **BORNES :** $dim_2(P) \geq h(P)$.

GENERATION

- ▷ Coloration du graphe de conflit de Skorsky [Sko92] (voir Chapitre 5).
- ▷ Simple coloration [Cas93] : relaxation du problème consistant à chercher un plongement ϕ tel que $\forall\ x \in X,\ |\phi(x) \setminus \bigcup_{y \in Pred_P(x)} \phi(y)| \leq 1$. Equivalent à colorer un graphe de conflit (plus petit que celui de Skorsky) [CHNR99]. Comprend le codage de Ait-Kaci *et al* [AKBLN89] et les codages atomiques [HHN95].
- ▷ Simple coloration avec "équilibrage" [KVH97] : variante consistant à commencer par modifier l'ordre en rajoutant des éléments pour obtenir ensuite une simple coloration plus petite. Bons résultats en pratique, mais absence d'analyse (performances de l'équilibrage ?).
- ▷ Codage dichotomique (ARBRES) [RT01] : algorithme proposant une stratégie d'"équilibrage" optimale pour les arbres. 4-approximation de la 2-dimension des arbres (voir Chapitre 5).

VARIANTES

Variantes dans l'implantation : stocker les ensembles $\phi(x)$ sous une autre forme que des vecteurs de bits (n'importe quelle structure de données permettant de traiter efficacement l'inclusion entre ensembles).

ATOUTS & INCONVENIENTS

Apparemment bonne compression pour les treillis et les ordres tels que le ratio hauteur/largeur est petit (prévisible vu la forme d'un treillis booléen).

REFERENCES

Cas particulier de la k-dimension introduite par Novak [Nov63]. Etudié d'un point du vue mathématique par Bouchet [Bou71], par Trotter [Tro75, Tro91], par Markowski [Mar80]. Article de synthèse par Habib et Nourine [HN94].

NOM : E-Dimension.

CATEGORIE : Plongement dans un treillis. Généralise les codages liés à la dimension et ceux liés à la 2-dimension. Contient les codages par partitions en chaines.

AUTEURS : Habib, Huchard et Nourine [HHN95].

METHODE

▷ **DESCRIPTION :** Construire un plongement d'ordre ϕ de P dans un produit de k chaines $[1, n_1] \times \ldots \times [1, n_k]$. Associer à chaque $x \in X$ le k-uplet d'entiers $\phi(x) = (v_x[1], \ldots, v_x[k])$.
Test de comparaison : $x \leq_P y$ si et seulement si $v_x[i] \leq v_y[i]$ pour tout i, $1 \leq i \leq k$.

▷ **PARAMETRES :**
- $NbChaines$ = nombre de chaines du produit.
- $Taille$ = nombre de bits d'informations pour un élément, soit $\sum_{1 \leq i \leq k} \lceil \log_2(n_i) \rceil$.

▷ **IMPLANTATIONS :** Chaque élément x reçoit le tableau d'entiers $(v_x[1], \ldots, v_x[Nbchaines])$. Test en temps $\mathcal{O}(NbChaines)$.
- Version tableau : taille du codage en $\mathcal{O}(NbChaines.\lceil \log_2(n) \rceil)$.
- Version pseudo-tableau : taille du codage en $\mathcal{O}(Taille)$.

OPTIMISATION

▷ **PARAMETRES :**
- Minimiser $Taille$, le minimum est appelé *encoding-dimension*, noté $edim(P)$. Problème $\mathcal{NP}$-complet ? Approximations ?
- Minimiser $NbChaines$ revient exactement au calcul de la *dimension* $dim(P)$.

▷ **LIENS :** ?

▷ **BORNES :** $dim(P) \leq edim(P) \leq \lceil \log_2(n) \rceil dim(P)$.

GENERATION

▷ Algorithme optimal (minimise $Taille$) pour les ordres tels que $dim(P) = 2$ et en se fixant $NbChaines = 2$ [dlHN97]. Complexité en temps $\mathcal{O}(| \leq_P |)$. Passe par une décomposition modulaire.

VARIANTES

▷ Si on impose $n_1 = \ldots = n_k = t$, le nombre minimum de chaines où se plonge P est appelé *t-dimension*, noté $dim_t(P)$. Problème d'optimisation $\mathcal{NP}$-complet ? (pour t constant, fonction de n ou d'autres paramètres de l'ordre)

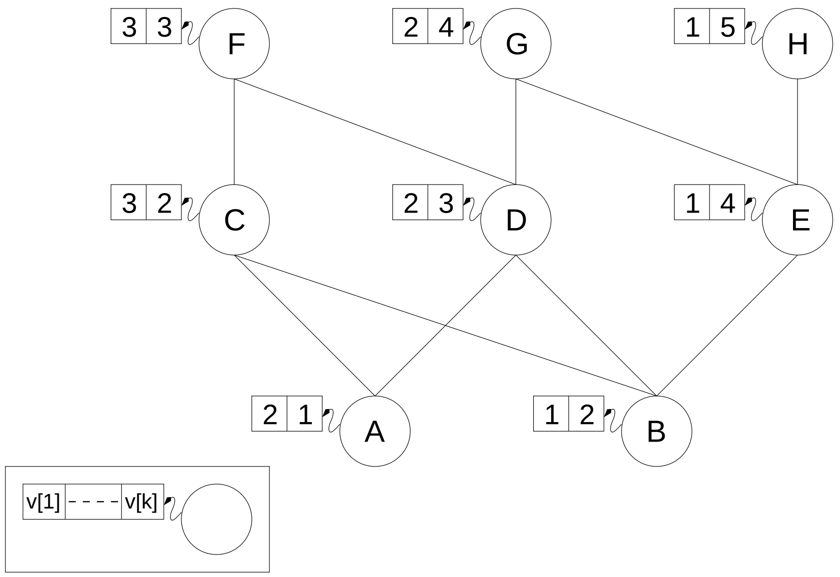
▷ Variante où P est un treillis et le plongement est un $\wedge$-morphisme (resp. $\vee$-morphisme).

- Il existe un $\wedge$-plongement (resp. $\vee$-plongement) de P dans $[1, n_1] \times \ldots \times [1, n_k]$ si et seulement si il existe une partition de $(J(P), \leq_P)$ (resp. $(M(P), \leq_P)$) en k chaines avec respectivement $\leq n_1 - 1$ éléments, ..., $\leq n_k - 1$ éléments [Bou71].
- Construction d'un $\wedge$-plongement ϕ de P dans $[0, n_1 - 1] \times \ldots \times [0, n_k - 1]$ à partir d'une partition de $(J(P), \leq_P)$ en k chaines avec respectivement $n_1 - 1$ éléments, ..., $n_k - 1$ éléments [Bou71] :
  1. Construire le codage par partition en chaines de $(J(P), \leq_P)$. Chaque $j \in J(P)$ reçoit le k-uplet d'entiers $\phi(j) = (v_j[1], \ldots, v_j[k])$.
  2. Etendre ce codage à X en associant à tout $x \in X$, la borne supérieure dans $[1, n_1] \times \ldots \times [1, n_k]$ des $\phi(j)$, $j \in J_x$, autrement dit $(\max_{j \in J_x} v_j[1], \ldots, \max_{j \in J_x} v_j[k])$.
- Opération de borne inférieure : le code de $x \wedge_P y$ est $(\min(v_x[1], v_y[1]), \ldots, \min(v_x[k], v_y[k]))$.
- Construction d'un $\vee$-plongement : passer au dual.
- Optimisation :
  - Minimiser $Taille$. Polynomial ? Avec un théorème Min-Max ?
  - Minimiser $NbChaines$. Equivalent à recouvrir $J(P)$ (resp. $M(P)$) avec le minimum de chaines, c.à.d. $w(J(P))$ chaines (resp. $w(M(P))$). Polynomial (voir Partition en Chaines).
  - Si on fixe $n_1 = \ldots = n_k = t$, les deux optimisations sont équivalentes et on note $\wedge - dim_t(P)$ (resp. $\vee - dim_t(P)$) le nombre minimum de chaines. Cas polynomiaux ? On sait que $\wedge - dim_2(P) = |J(P)|$ et $\wedge - dim_n(P) = w(J(P))$ (remplacer par $M(P)$ pour les $\vee$-plongements). De plus $\wedge - dim_3(P)$ est polynomial par couplage, mais le recouvrement minimum d'un ordre par des chaines avec $\leq t - 1$ éléments est $\mathcal{NP}$-complet pour $t \geq 4$ [ST91].

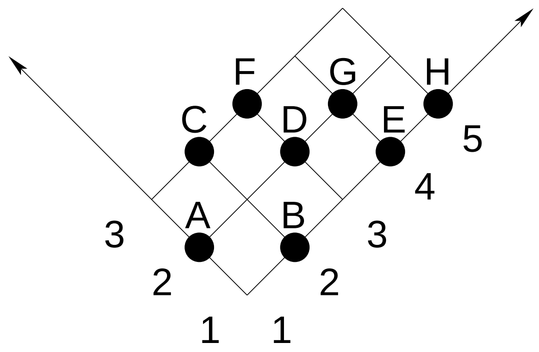
REFERENCES

Définition de la *encoding-dimension* par Habib, Huchard et Nourine [HHN95]. Redécouvert dans [GS01] sous le nom de *string-dimension* qui est la *encoding-dimension* de l'ordre quotienté par la relation "éléments jumeaux" (mêmes prédecesseurs et mêmes successeurs). Thèse de Bouchet sur les $\wedge$-plongements et $\vee$-plongements [Bou71]. Propriétés structurelles des produits de chaines dans le livre de Anderson [And87].

$\boxed{\text{E-Dimension}}$

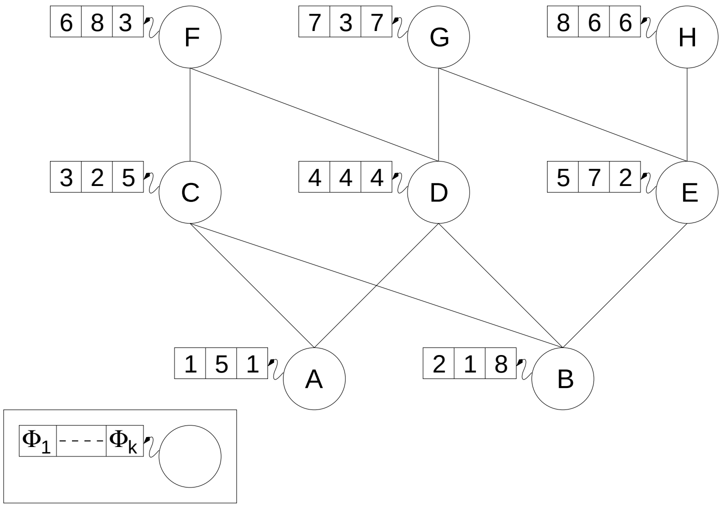


Codage vu comme un plongement d'ordre dans $[1, 3] \times [1, 5]$

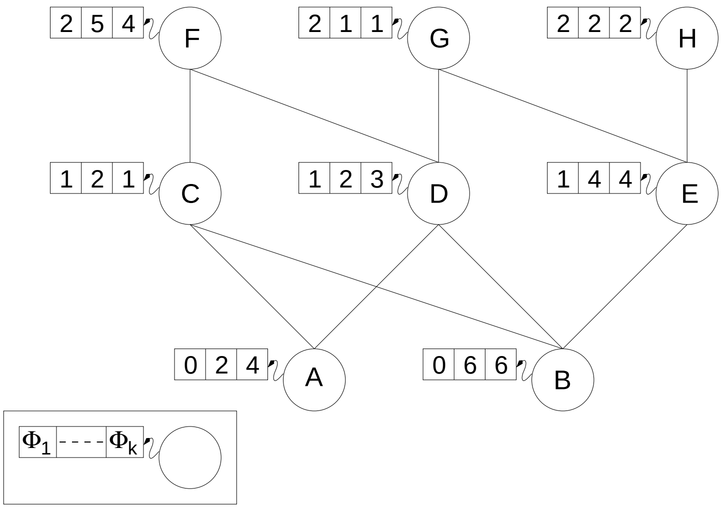


$\boxed{\text{Dimension Booléenne}}$

Un codage pour la formule $x_1 \leq y_1$ `ET NON`$(x_2 \leq y_2$ `ET` $x_3 \leq y_3)$.



Un codage pour une des variantes avec la formule
$x_1 \leq y_1$ `ET NON`$(y_1 \leq x_1)$ `ET NON`$(x_2 = y_3$ `OU` $x_3 = y_2)$.



NOM : Dimension Booléenne.

CATEGORIE : Compression logique.

AUTEURS : Gambosi, Nesestril et Talamo [GNT87b].

METHODE

▷ **DESCRIPTION :** Trouver k bijections ϕ_1,...,ϕ_k de X dans $[1, n]$ (équivalent à numéroter k fois l'ensemble X) et une formule booléenne $\mathcal{F}$ utilisant `NON`, `ET`, `OU` et les termes $x_i \leq y_i$ pour $1 \leq i \leq k$, telles que en prenant x_i (resp. y_i) égal à $\phi_i(x)$ (resp. $\phi_i(y)$), alors $x <_P y$ si et seulement si $\mathcal{F}$ est vraie. Par conséquent, associer à l'ordre P la formule $\mathcal{F}$ et à chaque élément x le k-uplet d'entiers $(\phi_1(x), \ldots, \phi_k(x))$.
Test de comparaison : $x <_P y$ si et seulement si $\mathcal{F}$ est vraie en assignant $\phi_i(x)$ (resp. $\phi_i(y)$) à la variable x_i (resp. y_i) pour tout i, $1 \leq i \leq k$.

▷ **PARAMETRES :**
- $NbNum$ = nombre de bijections.
- *Longueur* = nombre d'opérations $\leq$, `NON`, `ET`, `OU` dans la formule $\mathcal{F}$.

▷ **IMPLANTATIONS :** Chaque élément x reçoit le tableau d'entiers $(\phi_1(x), \ldots, \phi_{NbNum}(x))$. On stocke aussi la formule $\mathcal{F}$. Espace total de codage en $\mathcal{O}(n.NbNum.\lceil \log_2(n) \rceil + Longueur.\lceil \log_2(n) \rceil)$. Test en temps $\mathcal{O}(Longueur)$.

OPTIMISATION

▷ **PARAMETRES :**
- Minimiser $NbNum$. Le minimum est appelé *dimension booléenne*, noté $dim_B(P)$. Complexité ?
- Minimiser *Longueur*. Complexité ?
- Minimiser $n.NbNum.\lceil \log_2(n) \rceil + Longueur.\lceil \log_2(n) \rceil$. Complexité ?

▷ **LIENS :** Dimension.

▷ **BORNES :**
- $dim_B(P) \leq dim(P)$, pour tout k, $1 \leq k \leq 3$, $dim_B(P) = k$ si et seulement si $dim(P) = k$ et il existe une constante $c > 0$ telle que pour tout ordre P, $dim_B(P) \leq c \log(n)$ [GNT90].
- $dim_B(P) \leq Longueur$. Autres bornes inférieures intéressantes sur *Longueur* ?
- Liens entre $NbNum$ et *Longueur* ? Par exemple, bornes inférieures pour $n.NbNum + Longueur$?

GENERATION

Existence d'un codage tel que $NbNum$ et $Longueur$ sont en $\mathcal{O}(\sqrt{|\leq_P|})$, prouvée dans [GNT90]. Complexité de la génération d'un tel codage ?

VARIANTES

- ▷ Autoriser dans la formule toutes les comparaisons $x_i \leq y_j$, pour i, j quelconques. Possibilités de gains importants sur $dim_B(P)$ et la longueur des formules ?
- ▷ Variante où l'application $x \to (\phi_1(x), \ldots, \phi_k(x))$ de X dans $[1, n]^k$ où pour tout $1 \leq i \leq k$, ϕ_i est injective, devient une application de X dans $[1, n_1] \times \ldots \times [1, n_k]$ où $\phi_i(x)$ n'est pas forcément injective. Choix de la version tableau ou pseudo-tableau. Comme pour la *e-dimension* par rapport à la *dimension*, introduction d'un nouveau paramètre à minimiser : $Taille = \sum_{1 \leq i \leq k} \lceil \log_2(n_i) \rceil$
- ▷ Autoriser dans la formule les comparaisons $x_i \leq y_j$, mais aussi les comparaisons $x_i \geq y_j$ (qui ne s'obtiennent pas à partir d'une formule avec les $x_i \leq y_j$ et les `OU`, `ET`, `NON`, de longueur constante).
- ▷ Autoriser dans la formule l'utilisation de l'addition + sur les entiers et/ou l'utilisation de constantes entières.

ATOUTS & INCONVENIENTS

Potentiel de compression en espace très important, mais comme l'"information" passe dans l'algorithme de comparaison (c.à.d. la formule $\mathcal{F}$), on risque d'avoir des formules très longues donc couteuses en espace et en temps pour les tests. Certains codages issus des variantes peuvent s'interpréter géométriquement (inclusion de figures par exemple). En enrichissant encore d'instructions les variantes, on tend vers la définition des réprésentations fortement implicites.

REFERENCES

Méthode proposée et étudiée par Gambosi, Nesestril et Talamo [GNT87b, GNT87a, GNT90], mais reformulée dans cette fiche car les articles cités manquent de clarté sur certains points (en particulier sur les définitions et les bornes). La définition originale de [GNT87b, GNT87a, GNT90] impose que pour chaque i, l'ordre total induit par ϕ_i sur X suive une extension linéaire de P. Pour cette définition (non utilisée ensuite dans leurs textes), on peut prouver que $dim_B(P) = dim(P)$ et que $Longueur \geq 2dim(P) - 1$, l'optimum étant atteint par la formule $(x_1 \leq y_1)$ `ET` $\ldots$ `ET` $(x_{dim(P)} \leq y_{dim(P)})$. Autrement dit, l'optimum est le codage par la *dimension*.

NOM : Inclusion de figures géométriques.

CATEGORIE : Plongement d'ordre. Peut souvent être vu comme un cas particulier de la dimension booléenne.

AUTEURS : variés.

METHODE

- ▷ **DESCRIPTION :** Trouver un entier d, une famille $\mathcal{F}$ de parties de $\mathbb{R}^d$ et un plongement d'ordre ϕ de P dans $\mathcal{F}$ munie de la relation d'inclusion. Très nombreuses variantes en se fixant des conditions sur la famille $\mathcal{F}$: par exemple en fixant $d \leq 1$ et en prenant comme famille les cubes de $\mathbb{R}^d$ ou bien les sphères de $\mathbb{R}^d$ pour la distance euclidienne.
- ▷ **PARAMETRES :** $Taille$ = nombre moyen de bits pour stocker l'image d'un élément par ϕ.
- ▷ **IMPLANTATIONS :** Tester si $x \leq_P y$ est équivalent à tester si $\phi(x) \subseteq \phi(y)$. La résolution de ce test d'inclusion (et donc sa complexité en temps) dépend de la famille $\mathcal{F}$ (nécessite parfois l'opération de multiplication comme pour l'inclusion de sphères).

OPTIMISATION

- ▷ **PARAMETRES :** Nombreux problèmes d'optimisation (notamment pour minimiser $Taille$ et la vitesse du test d'inclusion) suivant les contraintes que l'on s'impose . Par exemple étant une famille $\mathcal{G}$ de figures géométriques, trouver le plus petit entier d tel que P se plonge dans $\mathcal{G} \cap \mathbb{R}^d$.
- ▷ **LIENS :** Dimension d'un ordre (par exemple $dim(P) \leq d$ ssi P se plonge dans la famille des cubes de $\mathbb{R}^{d-1}$).
- ▷ **BORNES :**

GENERATION

- ▷ Problèmes de reconnaissance des ordres pouvant se plonger dans une famille $\mathcal{F}$ fixé (si on impose beaucoup de conditions à $\mathcal{F}$, il n'existe pas nécessairement de plongement pour tout ordre dans $\mathcal{F}$), puis optimisation du plongement s'il en existe au moins un dans la famille $\mathcal{F}$.
- ▷ Introduction de la notion de *degré de liberté* de $\mathcal{F}$ par Alon et Scheinerman comme critère d'existence de plongements dans $\mathcal{F}$ en fonction de la dimension des ordres à coder [AS88].

VARIANTES

- ▷ $\mathcal{F}$ est la famille des sphères de $\mathbb{R}^d$ (pour la distance euclidienne). Si $dim(P) \leq 2$, alors il existe un plongement de P dans la famille des disques de $\mathbb{R}^2$, mais il existe un ordre de dimension 3 qui ne se plonge dans aucune famille de sphères [FFT99]. Si P est un ordre d'intervalle, il existe un plongement de P dans la famille des disques de $\mathbb{R}^2$.
- ▷ $\mathcal{F}$ est la famille des polygones du plan. Si $dim(P) \leq 3$, alors pour tout entier $c \geq 3$ il existe un plongement de P dans la famille des polygones réguliers à c cotés.
- ▷ $\mathcal{F}$ est l'ensemble des secteurs angulaires du plan. Tout ordre d'intervalle peut se plonger dans $\mathcal{F}$.

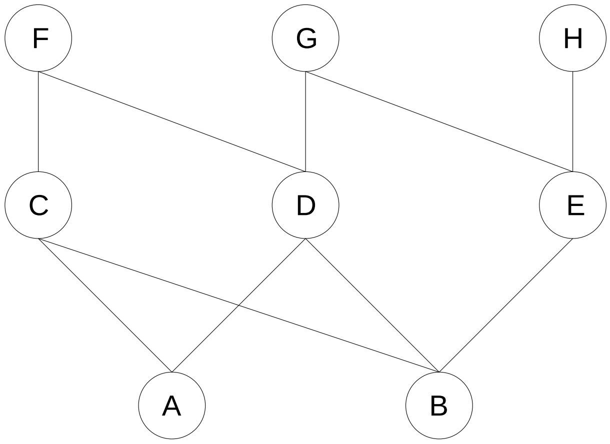
ATOUTS & INCONVENIENTS

Ces représentations fortement implicites sont particulièrement adaptées si les ordres à coder sont issus d'un processus qui fournit naturellement les plongements. Par contre pour un ordre quelconque, les problèmes d'existence de plongement (avant même l'optimisation) sont difficiles, la complexité des tests de comparaison peut être importante (temps non constant en général) et la compression en espace n'est pas toujours garantie.

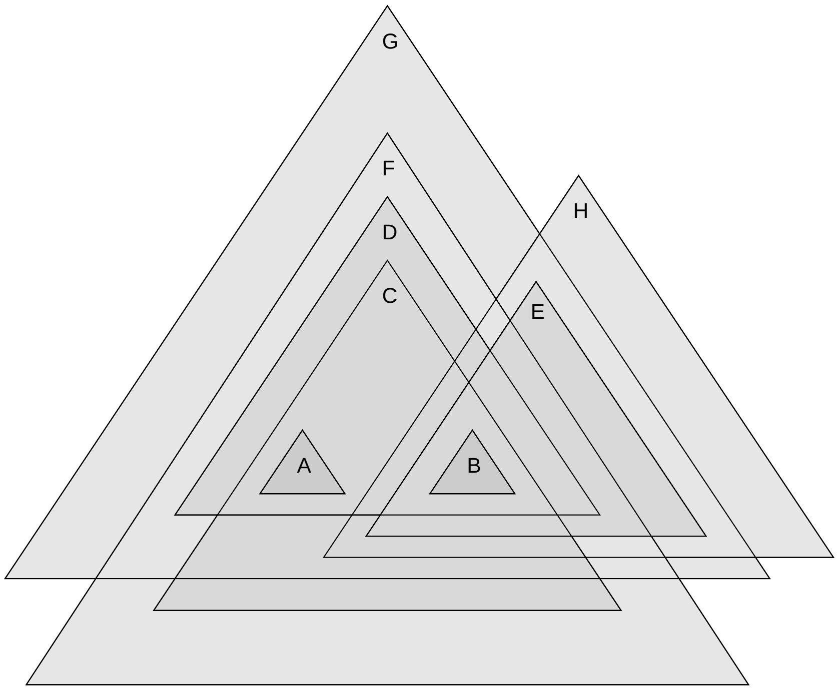
REFERENCES

De nombreuses références sont listées dans [Tro91, Tro95] et [FT99]. Consulter aussi [Cer00] à propos des interprétations géométriques des ordres.

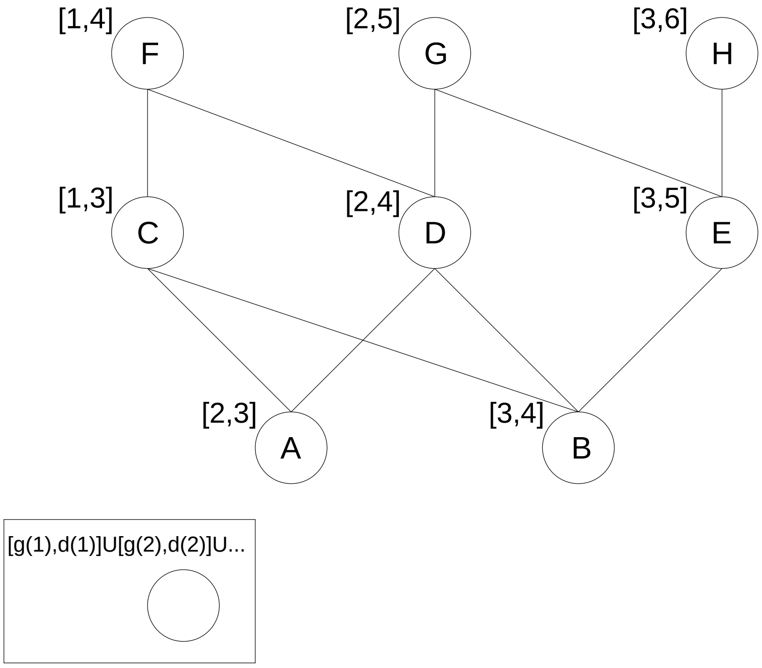
Inclusion de figures géométriques



Représentation par triangles équilatéraux, ordonnés par inclusion



Représentation par union d'intervalles



NOM : Représentation par union d'intervalles.

CATEGORIE : Plongement d'ordre. Exemple de codage par inclusion de figures géométriques. Contient les codages par range compression.

AUTEURS : Madej et West [MW91, MW95].

METHODE

▷ **DESCRIPTION :** Construire un plongement d'ordre ϕ de P dans la famille des unions finies d'intervalles d'entiers (munie de la relation d'inclusion). Associer à chaque $x \in X$ l'union d'intervalles $\phi(x) = [g_x(1), d_x(1)] \cup \ldots \cup [g_x(k_x), d_x(k_x)]$.
Test de comparaison : $x \leq_P y$ si et seulement si $\phi(x) \subseteq \phi(y)$.

▷ **PARAMETRES :**
- $MaxInt$ = nombre maximum d'intervalles pour un élément.
- $NbInt$ = nombre total d'intervalles pour tout l'ordre, c.à.d. $\sum_{x\in X} k_x$.
- $MaxSet$ = nombre d'entiers sur lesquels sont pris les intervalles, c.à.d. $\max_{x\in X} d_x(k_x) - \min_{x\in X} g_x(1)$ (en supposant les intervalles triés).

▷ **IMPLANTATIONS :** Chaque élément x reçoit un tableau de $2k_x$ entiers ($g_x(1)$,$d_x(1)$,...,$g_x(k_x)$,$d_x(k_x)$) et son nombre d'intervalles k_x. Espace total de stockage en $\mathcal{O}(NbInt.\lceil\log_2(MaxSet)\rceil)$. Test en temps $\mathcal{O}(MaxInt)$ au pire (tableaux triés).

OPTIMISATION

▷ **PARAMETRES :**
- Minimiser $MaxInt$. Le minimum est appelé *nombre d'intervalles* de P et noté $i(P)$. Problème d'optimisation $\mathcal{NP}$-complet : pour tout $d \geq 2$ fixé, tester si $i(P) \leq d$ est $\mathcal{NP}$-complet [MW91]. Approximations ?
- Minimiser $NbInt$. Problème $\mathcal{NP}$-complet ?
- Minimiser $NbInt.\lceil\log_2(MaxSet)\rceil$. Problème $\mathcal{NP}$-complet ?

▷ **LIENS :** Range compression, Dimension, Dimension intervallaire.

▷ **BORNES :**
- Bornes supérieures : $i(P) \leq range(P)$, $i(P) \leq \lceil dim(P)/2\rceil \leq \lceil n/4\rceil$ et $i(P) \leq \lceil\frac{3}{2}idim(P)\rceil$ [MW91].
- Si P est biparti, $i(P) \leq \lceil n/(\log_2 n - \log_2\log_2 n)\rceil + 1$ [MW95].
- Comportement en moyenne : soit X un ensemble fixé à n éléments, soit $\mathcal{PO}(X)$ l'ensemble des ordres sur X muni de la distribution uniforme, alors pour tout $\epsilon > 0$ fixé, $(1-\epsilon)n/(8\log_2 n) \leq i(P) \leq \frac{3}{2}\lceil n/(\log_2 n - \log_2\log_2 n)\rceil + 1$ pour presque tous les ordres

$P \in \mathcal{PO}(X)$ [MW95].

GENERATION

- ▷ Tout codage par range compression de P^d fournit un codage par union d'intervalles de P en prenant $\phi(x) = \{id_y \mid y \in Succ_{P^d}(x) \cup \{x\}\}$ (avec les notations de la méthode de *Range compression*).
- ▷ Reconnaissance des ordres tels que $MaxInt = 1$ équivalente à déterminer si $dim(P) = 2$ (polynomial).

ATOUTS & INCONVENIENTS

Remarquer que pour cette méthode, $i(P^d) \leq i(P) + 1$. Dans quels cas $i(P^d) = i(P)$? Par contre $range(P) - i(P)$ n'est pas borné par une constante (considérer par exemple les duals des arbres ternaires).

REFERENCES

Méthode proposée et analysée par Madej et West [MW91, MW95]. Une liste de problèmes ouverts est présentée dans [MW95].

Premier bilan

A ce stade, on peut faire quelques remarques sur les méthodes présentées, notamment pour qualifier le comportement des paramètres de ces méthodes. Soit $Param$ un paramètre d'une méthode, $ParamOpt(P)$ désignera la valeur optimale de ce paramètre pour un ordre P.

Symétrie

Une méthode est dite ***symétrique*** si elle se comporte similairement pour un ordre et son dual : pour tout paramètre $Param$ et tout ordre P, $ParamOpt(P^d) = ParamOpt(P)$. En fait, pour plusieurs des méthodes, les paramètres vérifient une propriété plus forte : ce sont des ***invariants de comparabilité***, autrement dit des paramètres $Param$ tels que si deux ordres P et Q ont le même graphe de comparabilité, alors $ParamOpt(P) = ParamOpt(Q)$.

Monotonie

Un paramètre $Param$ d'une méthode est dit ***monotone*** si pour tout ordre P et tout sous-ordre Q de P, $ParamOpt(Q) \leq ParamOpt(P)$. L'exemple courant est la monotonie de la taille du codage pour les représentations implicites des ordres.

Continuité

Un paramètre $Param$ d'une méthode est dit ***continu*** s'il existe une constante $c \geq 0$ telle que pour tout ordre $P = (X, \leq_P)$ et tout $x \in X$, l'ordre induit sur $X \backslash \{x\}$, noté $P - x$, vérifie $ParamOpt(P) \leq ParamOpt(P - x) + c$. On peut définir de même la notion de continuité pour l'ajout/suppression de comparaisons dans l'ordre.

Le tableau 3.1 précise le comportement des paramètres par rapport à ces trois propriétés. Il faut noter que plusieurs paramètres ont été présentés pour certaines méthodes, s'ils n'ont pas le même comportement, on précisera entre parenthèses le ou les paramètres examinés. On considère la continuité pour l'ajout/suppression d'un élément de l'ordre (sans ajout de comparabilités entre les autres éléments). Le tableau rappelle aussi quelles sont les méthodes en temps constant (colonne "$\mathcal{O}(1)$").

La monotonie et la continuité assurent que la méthode n'a pas un comportement imprévisible quand l'ordre évolue. C'est une indication pour la recherche d'algorithmes dynamiques de mises à jour du codage, même si cela ne fournit pas directement des algorithmes dynamiques efficaces.

Méthode	$\mathcal{O}(1)$	Symétrie	Monotonie	Continuité
Range compression ($MaxInt$)			X	X
Union d'ordres d'intervalles		X	X	X
Partition en antichaines	X		X	
Partition en chaines	X	X	X	X
PQ-encoding	X		X	X
Dimension		X	X	X
2-Dimension		X	X	X
E-Dimension		X	X	X
Dimension booléenne ($NbNum$)		X	X	X
Union d'intervalles ($MaxInt$)		?	X	X

TAB. 3.1 – Comportement des paramètres de méthodes de codage connues

D'autres méthodes

Il existe plusieurs autres méthodes de codage d'ordre dans la littérature qui n'ont pas été présentées dans notre liste, faute de temps. Parmi elles, on peut citer :

▷ ***Compact encoding*** (Vitek, Horspool, Krall [VHK97]) : méthode de codage présentée comme une technique de compression de matrice. Peut être vue comme de la composition de partitionnements.

▷ ***Logical term encoding*** (Dahl, Fall [DF93, Fal98]) : méthode de codage à base de compression d'ensembles. Utilise des termes logiques pour représenter les ensembles, la subsumption pour la comparaison et l'unification pour les bornes inférieures ou supérieures dans le cas d'un treillis. Peu analysée dans les références données ici.

▷ ***Arbre de codage des treillis distributif*** (Habib, Nourine [Nou93, HN96]) : utilisation de la structure de données d'arbre lexicographique pour représenter un treillis distributif $T = (X, \leq_T)$. Espace de stockage en $\mathcal{O}(|X|)$, comparaison en temps $\mathcal{O}(|J(T)|)$, bornes inférieures ou supérieures en temps $\mathcal{O}(|J(T)|)$. Génération du codage, c.à.d. de l'arbre de codage, en temps $\mathcal{O}(| \leq_P |)$ à partir de TC.

▷ ***Calcul de bornes inférieures dans un arbre (LCA)*** (Harel, Tarjan [HT84], Schieber, Vishkin [SV88], Bender, Farach-Colton [BFC00]) : les trois références présentent des méthodes de codage pour traiter l'opération de borne inférieure dans le cas particulier des arbres. Dans chaque cas, le codage est généré en temps $\mathcal{O}(n)$ à partir de TR, l'espace de stockage est en $\mathcal{O}(n)$ et le calcul de la borne inférieure de deux éléments s'effectue en temps $\mathcal{O}(1)$.

- ▷ ***Codages de treillis par arbres-doubles*** (Talamo, Vocca [TV97]) : méthode de codage pour la classe des treillis, basée sur une décomposition en idéaux puis en arbres-doubles. Espace en $\mathcal{O}(n\sqrt{n})$, comparaison en $\mathcal{O}(1)$. Algorithme de génération d'un codage en temps $\mathcal{O}(n^2)$ à partir de BM ou TC.
- ▷ ***Codages d'ordres par décomposition en bipartis réguliers*** (Nardelli, Talamo, Vocca [NTV99] : méthode de codage d'ordres annonçant comme performances en espace $\mathcal{O}(| \leq_P |. \log(n))$ et des comparaisons en $\mathcal{O}(1)$. Algorithme de génération d'un codage en $\mathcal{O}(n^2.D(P))$ à partir de BM ou TC, où $D(P) = \max\{|Succ_P(x)| \mid x \in X\}$.

Chapitre 4

Discussion sur les méthodes de codage d'ordres

Récapitulatif

Toutes les méthodes de codages présentées dans la section 3.2 tentent de résoudre le problème de la comparaison au mieux. L'objectif qu'elles partagent est de représenter un ordre avec le moins de mémoire possible, tout en ayant des algorithmes qui répondent très rapidement aux questions de comparaison à partir des informations qui ont été stockées.

Toutes les méthodes ont été présentées dans un cadre "statique" : l'ordre à coder est fixé et les besoins de compression en espace et de vitesse des tests passent avant la question des ressources (temps, espace) nécessaires pour générer le codage de l'ordre. On va revenir ensuite sur ces problèmes de génération du codage, sur les problèmes d'optimisation associés et les problèmes de traitement dynamique du codage si l'ordre est modifié.

On s'est a priori placé dans un modèle séquentiel, le modèle Word-RAM, à la fois pour l'utilisation des codages et pour la génération de ces codages. On peut quand même remarquer que pratiquement toutes les méthodes présentées ne font usage que des informations locales de x et de y pour déterminer si $x \leq_P y$? (seule la méthode des ensembles de successeurs immédiats qui utilise un parcours de graphe pour résoudre les tests utilise les informations d'autres éléments). Ces méthodes distribuent bien l'information.

On peut remarquer que ces méthodes ne font pas appel à des structures de données très sophistiquées. Ceci peut s'expliquer par le fait que les auteurs cherchent à se rapprocher le plus possible de tests en temps constant dans le pire des cas (sachant qu'on suppose ne rien connaître de l'enchaînement des requêtes, ce qui empêche de raisonner en amorti ou en moyenne). Et même si le temps constant n'est pas atteint, l'objectif reste de donner des algorithmes de traitement des opérations les plus simples possibles, avec le moins d'instructions possibles (toutes les méthodes qui ont été présentées fournissent des codages uniformes, mis à par la méthode de la *Dimension Booléenne*).

Le lecteur est renvoyé aux études de Vitek, Horspool et Krall [VHK97] et de Zibin et Gil [ZG01] sur un jeu de tests, ainsi qu'aux expérimentations de Agrawal *et al* [ABJ89a, ABJ89b] ou de Capelle [Cap94] pour vérifier l'important potentiel de compression de bon nombre de méthodes qui ont été proposées (au moins pour certaines classes d'ordres, les jeux de tests ou les simulations ayant en général des distributions très particulières).

Comparaison des méthodes

Quant à la comparaison des méthodes présentées dans la section 3.2, elles exploitent chacunes des structures régulières (souvent des classes d'ordres faciles à coder) qui sont différentes dans chaque cas. Elles ont ainsi chacune un terrain de prédilection et aucune n'est universellement meilleure que toutes les autres. On peut pousser cette remarque un peu plus loin. Soit $\mathcal{C}$ une classe d'ordres admettant une méthode polynomialement optimale, alors il est possible de construire une méthode de codage pour tous les ordres, qui privilégie les ordres de $\mathcal{C}$ (c.à.d. avec des performances optimales pour ces ordres) : il suffit de considérer une représentation par intersection ou par union basée sur $\mathcal{C}$ (pour peu que $\mathcal{C}$ remplisse une des conditions permettant de définir ces représentations présentées dans la section 2.6). Noter que si l'on cherche une méthode générale ***en temps constant*** privilégiant les ordres de $\mathcal{C}$, le problème se complique et la question de l'existence d'une telle méthode générale se pose. On donnera un élément de réponse dans les paragraphes sur l'élaboration de méthodes.

Malgré tout, certaines méthodes ont clairement de meilleurs résultats que d'autres (en espace de stockage et en temps de l'opération de comparaison) sur certains jeux de tests [Cap94, VHK97, ZG01]. On peut y voir deux explications. Il peut s'agir d'une simple illustration de ce qu'on vient de dire précédemment : si par exemple les ordres des jeux de tests se ressemblent beaucoup, si une des méthodes est définie à partir d'une classe à laquelle ils appartiennent presque tous, ou s'ils sont tous de paramètre borné pour l'une des méthodes. Sinon il existe peut être une notion de meilleurs résultats "en moyenne" à définir : par exemple en fixant une distribution intéressante sur l'ensemble des ordres, et en étudiant non seulement l'espérance de l'espace de codage mais aussi sa variance ...

D'autre part, si on compare les méthodes deux à deux (et non plus par rapport à toutes les autres), l'une est parfois une généralisation de l'autre et conserve, voire améliore, ses performances. Il est donc parfois possible de comparer objectivement deux méthodes en annonçant que l'une est meilleure que l'autre.

Ces remarques, les différents atouts et inconvénients exposés dans les fiches, ainsi que les problèmes d'adéquation des méthodes aux applications pratiques montrent qu'il est difficile de construire une unique échelle de comparaison des méthodes.

Par contre, si l'on se fixe des objectifs précis de performances, on peut se demander quels ingrédients permettent de les atteindre ? Si on veut une méthode en temps constant, il faut apparemment une méthode qui morcelle vraiment les informations nécessaires à l'opération, en conservant un accès en temps constant à ces informations. L'idéal est que l'information nécessaire tienne toujours sur un mot mémoire de w bits, et l'utilisation de la structure de tableau permet par exemple des accès en temps constant. Ces remarques ainsi que les exemples de la section 3.2 montrent que les méthodes à base de partitionnement sont de très bons candidats pour obtenir des méthodes en temps constant pour la comparaison.

Quel est alors l'intérêt d'utiliser des méthodes en temps non constant (comme la *e-dimension*) ou non uniformes (comme la *dimension booléenne*) ? Plusieurs réponses sont envisageables : meilleures compressions en espace avec des comparaisons en temps presque constants, possibilité de traiter d'autres opérations que la comparaison, classes d'ordres codées efficacement plus grandes (on note l'intérêt de savoir fixer les bornes sur un paramètre telles que la reconnaissance des ordres entre ces bornes reste polynomiale), adéquation avec certaines applications pratiques, adéquation avec certaines architectures de machines (par exemple dans un pipeline, meilleur parallélisme d'instructions pour les opérations), algorithmes plus rapides de génération de codages, mises à jour plus rapides dans un contexte dynamique ? Pour l'instant, nous ne trancherons aucune de ces questions concernant la comparaison des méthodes en temps constant par rapport aux autres méthodes. Nous manquons de résultats théoriques et de données précises sur ces questions pour les méthodes de la section 3.2. Remarquer que ces questions peuvent aussi permettre de départager des méthodes en temps constant.

Elaboration de méthodes

A partir d'une classe d'ordres pour laquelle on connaît une bonne méthode de codage (par exemple une méthode polynomialement optimale), on peut très souvent élaborer une ou plusieurs méthodes pour les ordres en général en considérant la technique de plongement d'ordre ou les représentations par intersection ou par union (voir section 2.6). Si les méthodes contruites ainsi peuvent très bien compresser en espace, elles sont par contre rarement en temps constant.

On revient maintenant sur la technique de partitionnement, prometteuse puisque les meilleures méthodes en temps constant y font appel (voir la section 2.6 pour les notations). Trois approches sont maintenant présentées pour élaborer des méthodes de codage à base de partitionnement et si possible proposer des algorithmes de génération de ces codages : partionnement et compression d'ensembles, partionnement et représentation par union, composition de partitionnements. Il existe probablement de nombreuses autres manières d'utiliser le partitionnement que ces trois approches qui se recouvrent parfois.

▷***Partitionnement et compression d'ensembles* :**

On a vu que le partitionnement se marie très bien avec les techniques de compression d'ensembles (schémas A et B de la section 2.6). Ainsi dès que l'on a une idée pour décrire efficacement (et donc compresser) des sous-ensembles d'un ordre, on peut élaborer de nouvelles méthodes de codage avec les schémas A ou B. Considérons par exemple les *PQ-encodings* d'un ordre $P = (X, \leq_P)$, cette méthode s'incrit dans le schéma B de partitionnement :

Schéma B : Trouver une partition $\{T_j\}_{1\leq j\leq l}$ telle que pour tout j, il existe une numérotation de X telle que $\forall x \in T_j$, $\downarrow_P x$ est un intervalle $[g_x, d_x]$.

L'idée de compression consiste simplement à exprimer un ensemble sous forme d'un intervalle d'entiers pour peu qu'on ait bien numéroté les éléments. On peut par conséquent donner le pendant de cette méthode selon le schéma A :

Schéma A : Trouver une partition $\{S_i\}_{1\leq i\leq k}$ telle que pour tout i, il existe une numérotation de S_i telle que $\forall x \in X$, $S_i \cap \downarrow_P x$ est un intervalle $[g_x(i), d_x(i)]$ (éventuellement vide).

A notre connaissance, cette méthode ne correspond à aucune méthode connue. C'est une généralisation de la méthode de *partition en antichaines* :

celle-ci impose que les intervalles $[g_x(i), d_x(i)]$ soient réduits à un singleton ou vides. C'est aussi une généralisation de la méthode de *partition en chaines* : les chaines C_i sont numérotées dans l'ordre croissant et par conséquent les numéros des ensembles $C_i \cap_P \downarrow_P x$ forment bien des intervalles.

Remarquons que pour une idée de compression, on n'a pas toujours les deux versions (schéma A et schéma B) possibles, en particulier si on veut passer du schéma A au schéma B. Par exemple, la méthode de *partition en chaines* appartient au schéma A : on cherche une partition $\{S_i\}_{1 \leq i \leq k}$ de X telle que pour tout i, $\forall x \in X$, $S_i \cap \downarrow_P x$ est une chaine de P. Mais elle n'a pas directement de pendant dans le schéma B : on ne peut pas partitionner X en $\{T_j\}_{1 \leq j \leq l}$ tel que $\forall x \in T_j$, $\downarrow_P x$ est une chaine de P, à moins que P ne soit lui-même une chaine.

Pour proposer de nouvelles méthodes dans le schéma A ou B, il faut se rappeler que nous pouvons considérer les parties S_i (ou T_j) de la partition soit comme des ensembles sans aucune relation, soit munies de l'ordre induit par P (c'est par exemple le cas dans la méthode de *partition en chaines* où on impose que les C_i soient des chaines). Si on se place dans le schéma A, quelles combinaisons de contraintes faut-il imposer aux S_i et $S_i \cap \downarrow_P x$ pour avoir des méthodes efficaces ? (c'est-à-dire être capable, pour i et $x \in X$ fixés, de traiter rapidement la question "$y \in S_i \cap \downarrow_P x$?" pour tout $y \in S_i$). Nous pouvons imaginer beaucoup d'exemples de combinaisons :

- S_i quelconque et $S_i \cap \downarrow_P x$ est une chaine de S_i,
- S_i quelconque et $S_i \cap \downarrow_P x$ est une antichaine de S_i,
- S_i quelconque et $S_i \cap \downarrow_P x$ est un intervalle d'entiers pour une bonne numérotation de S_i (c'est la méthode qui a été proposée précédemment),
- S_i en tant qu'ordre admet un codage en temps constant et $S_i \cap \downarrow_P x$ est un idéal principal de S_i.

Mais a priori toutes ne fournissent pas des méthodes efficaces. Ces contraintes en entrainent d'autres (par exemple la première combinaison implique que les ordres induits sur les S_i sont des arbres) et ils n'est pas toujours possible qu'un codage puisse les respecter, ou bien que l'on puisse trouver ensuite des algorithmes de génération efficaces. La question de la génération de tels codages est bien sûr primordiale.

$\triangleright$ ***Partitionnement et représentations par union :***

Revenons aussi sur la technique de partitionnement vue comme représentation par union. La stratégie suivante permet d'élaborer des méthodes de codage d'ordres :

1. Identifier une classe d'ordres $\mathcal{C}$ ayant une méthode de codage performante $\mathcal{M}$ (par exemple $\mathcal{M}$ étant polynomialement optimale pour $\mathcal{C}$).
2. Pour tout ordre $P = (X, \leq_P)$, construire deux partitions $\{S_i\}_{1\leq i\leq k}$ et $\{T_j\}_{1\leq j\leq l}$ de X telles que tous les ordres $P_{i,j}$ sont dans $\mathcal{C}$.

Si $\mathcal{M}$ est polynomialement optimale pour $\mathcal{C}$ (donc en temps constant), alors on peut ainsi décrire une méthode en temps constant à partir de $\{S_i\}_{1\leq i\leq k}$ et $\{T_j\}_{1\leq j\leq l}$: on génére et on stocke un codage optimal pour chacun des ordres $P_{i,j}$, puis on associe à chaque élément $x \in X$ l'indice s_x de la partie de $\{S_i\}_{1\leq i\leq k}$ à laquelle x appartient et l'indice s'_x de la partie de $\{T_j\}_{1\leq j\leq l}$ à laquelle x appartient. Un test de comparaison est alors résolu par $x \leq_P y$ si et seulement si $x \leq_{P_{s_x,s'_y}} y$, ce dernier test s'exécutant en temps constant avec le codage de P_{s_x,s'_y}.

Si de plus $\mathcal{M}$ est une représentation implicite, alors on peut décrire une méthode en temps constant et implicite : il suffit de reprendre la construction précédente et ensuite de répartir les informations locales des codages des $P_{i,j}$ entre les différents éléments de P, cela correspond exactement à l'implantation proposée dans la section 2.6 pour les méthodes à base de partitionnement (où chaque $x \in X$ reçoit des entiers s_x, s'_x, des identifiants id_x, id'_x qui sont souvent des entiers et les informations locales $(t_x[1], \dots, t_x[k])$ et $(t'_x[1], \dots, t'_x[l])$ souvent sous forme de tableaux pour y accéder en temps constant).

Nous sommes maintenant confrontés au problème de la génération de tels codages. Supposons la classe $\mathcal{C}$ fixée ainsi que $\mathcal{M}$ une méthode polynomialement optimale pour $\mathcal{C}$, cette méthode contient un algorithme polynomial $\mathcal{A}$ de reconnaissance des ordres de $\mathcal{C}$ et de génération d'un codage optimal pour tout ordre de $\mathcal{C}$. Pour suivre la méthode proposée au-dessus, trouver deux partitions $\{S_i\}_{1\leq i\leq k}$ et $\{T_j\}_{1\leq j\leq l}$ telles que $\forall i, j$, $P_{i,j} \in \mathcal{C}$ impose beaucoup de contraintes. Sans avoir plus de connaissances sur $\mathcal{C}$, caractériser les couples de partitions qui fonctionnent semble très difficile. Par contre, on peut se restreindre à des cas particuliers de partitions, comme les types 2 et 3 de partionnement exposés dans la section 2.6.

Sans perte de généralité pour traiter les comparaisons et pour plus de clarté, on se place sur l'ordre biparti $B(P)$ (on pourrait aussi prendre $B^0(P)$). Retranscrite pour ce biparti, la méthode revient à trouver $\{S_i\}_{1\leq i\leq k}$ et $\{T_j\}_{1\leq j\leq l}$ telles que $\forall i, j$, le biparti $(S_i, T_j, \leq_P)$ est dans $\mathcal{C}$. On choisit de s'intéresser au type 2 de partionnement, c'est à dire $\{S_i\}_{1\leq i\leq k}$ est quelconque mais $\{T_j\}_{1\leq j\leq l}$ est réduit à $\{X\}$ (les constructions seront similaires pour le cas 3 de partitionnement). On cherche donc à générer une partition $\{S_i\}_{1\leq i\leq k}$ de X telle que pour tout i, le biparti $(S_i, X, \leq_P)$ est dans $\mathcal{C}$ et à générer le codage de ces bipartis. Pour cela, il existe un algorithme glouton de génération très simple, il est décrit par l'algorithme *CodageGlouton* ci-dessous. Il nécessite que $\mathcal{C}$ contienne les anti-arbres de hauteur 1 (c.à.d. un élément maximal avec ses prédécesseurs dans le biparti).

Algorithme 1: [***CodageGlouton***] Génère un codage pour $B(P)$ à partir de la classe $\mathcal{C}$.

Entrée : Le biparti $B(P)$ et un algorithme $\mathcal{A}$ de reconnaissance et de génération de codage optimal pour $\mathcal{C}$.

Sortie : Une partition $\{S_i\}_{1\leq i\leq k}$ de X et des codages optimaux des bipartis $(S_i, X, \leq_P)$ qui sont tous dans $\mathcal{C}$.

```
début
  Y ← X ; i ← 1 ;
  tant que Y ≠ ∅ faire
    Choisir un élément y de Y ; S_i ← {y} ;
    Y ← Y − {y} ; Z ← Y :
    tant que Z ≠ ∅ faire
      Choisir un élément z de Z ;
      Reconnaitre avec A si (S_i ∪ {z}, X, ≤_P) est dans C ;
      si OUI alors S_i ← S_i ∪ {z} ;
      Z ← Z − {z} ;
    Générer avec A un codage optimal de (S_i, X, ≤_P) ;
    i ← i + 1 ;
fin
```

Du point de vue de la complexité en temps de cette génération, l'idéal est d'avoir un algorithme de reconnaissance $\mathcal{A}$ qui soit incrémental, ce qui évite de reprendre l'étape de reconnaissance à zéro après chaque ajout d'élément, et qui calcule un codage optimal en même temps que la reconnaissance. Avec cet algorithme glouton de génération, on n'a pas forcément le meilleur codage en espace pour la méthode générale de partitionnement associée à $\mathcal{C}$ (que l'on se soit astreint de rester dans les types 2 ou 3 de partitionnement ou pas), mais pour tout ordre on peut génèrer rapidement un codage correspondant à cette méthode.

Nous allons maintenant donner quelques exemples illustrant ces méthodes de partionnement basées sur une classe d'ordres $\mathcal{C}$:

1. Si on prend $\mathcal{C}$ la classe des ordres dont le dual est un biparti convexe (un ordres biparti $B = (U, V, <_B)$ est convexe s'il existe une numérotation de U vérifiant $\forall x \in V$, $ImPred_B(v)$ est un intervalle d'entiers) et si on se restreint au cas 3 de partionnement, on retrouve exactement la méthode des *PQ-Encodings*. Il existe une méthode polynomialement optimale pour $\mathcal{C}$ avec un algorithme incrémental de reconnaissance basé sur les PQ-trees [BL76] qui génère simultanement un codage optimal. La version de *CodageGlouton* utilisant cet algorithme s'avère être exactement l'heuristique de génération des *PQ-Encodings* proposée dans [ZG01].
2. Si on prend $\mathcal{C}$ la classe des ordres d'intervalles et un algorithme de reconnaissance incrémental, on a une restriction de la *représentation par union d'ordres d'intervalles* sur les bipartis qui est en temps constant. En fait en quittant le biparti $B(P)$ et en se replaçant sur P, on peut prouver que l'on construit ainsi une partition en chaines de P, on est donc en quelque sorte en train de considérer un cas particulier de la méthode de *partition en chaines*.
3. Si on prend $\mathcal{C}$ la classe des ordres de dimension égale à 2, est-ce que le codage obtenu correspond à une méthode connue ?

Pour ce genre de méthode, on travaille donc directement sur le biparti $B(P)$ (ou bien $B^0(P)$). Afin d'élaborer des méthodes efficaces en suivant cette stratégie, quels facteurs interviennent ? Comment choisir la classe $\mathcal{C}$ et sa méthode de codage $\mathcal{M}$? Il semble clair que plus la classe $\mathcal{C}$ est proche des ordres codés en pratique, plus les performances des codages seront bonnes en pratique.

$\triangleright$ ***Composition de partitionnement* :**

L'idée consiste à enchainer plusieurs étapes de partitionnement et non pas une seule. Dans quels cas, est-ce que cela peut améliorer les performances des méthodes (par exemple la compression en espace) ?

Questions de complexité

Le première remarque concerne les problèmes d'optimisation liés à la génération des codages. Ces problèmes d'optimisation sont importants puisqu'ils permettent de minimiser les espaces de stockage et aussi d'accélérer les opérations sur les ordres. Cependant pour l'opération de comparaison, on peut constater sur plusieurs des méthodes présentées (et conjecturer pour les autres) que les méthodes efficaces sont souvent associés à des problèmes d'optimisation $\mathcal{NP}$-complets. Les résultats de non-approximabilité montrent qu'il s'agit de problèmes très difficiles. La similitude entre les ratios de non-approximabilité (souvent liés à ceux de la coloration de graphe) nous amène à nous demander si ces problèmes d'optimisation ont un comportement similaire tout simplement parce que leurs formulations, leurs contraintes, sont du même genre, ou plutôt parce qu'il existerait un problème général sous-jacent à toute méthode de codage efficace des ordres, dont la difficulté se répercuterait sur les problèmes d'optimisation associés ?

Une autre question concerne les bornes de complexité sur les méthodes de codage : suivant les contraintes imposées sur le temps, quelles sont les bornes (inférieures ou supérieures) sur l'espace, et inversement ? L'opération qui nous intéresse est toujours la comparaison entre éléments. Si on reprend les trois méthodes classiques de codage des ordres, on a les complexités suivantes dans notre modèle Word-RAM :

1. **Matrice d'incidence** : espace total de stockage en $\mathcal{O}(n^2)$ bits, comparaison en $\mathcal{O}(1)$.
2. **Ensembles de successeurs** : espace total en $\mathcal{O}(| <_P |.\lceil \log_2(n) \rceil)$ bits, comparaison en $\mathcal{O}(\log_2(n))$.
3. **Ensembles de successeurs immédiats** : espace total de stockage en $\mathcal{O}(| \prec_P |.\lceil \log_2(n) \rceil)$ bits, comparaison en $\mathcal{O}(n)$.

Parmi les méthodes existantes, beaucoup d'auteurs se sont intéressés aux méthodes en temps constant (dans le modèle Word-RAM). C'est pourquoi on peut se demander quelle est la frontière des méthodes de codage en temps constant (pour la comparaison entre éléments). Les questions suivantes concernent cette frontière (voir s'il est plus approprié de laisser ou pas le facteur $\lceil \log_2(n) \rceil$ dans les bornes de complexité ? Par exemple, mettre $| <_P |.\lceil \log_2(n) \rceil$ bits ou simplement $| <_P |$ bits comme borne dans les questions ?) :

1. Est-ce qu'il existe une méthode de codage en temps constant et en espace $\mathcal{O}(| <_P |.\lceil \log_2(n) \rceil)$?
2. Quelles classes d'ordres admettent une méthode de codage en temps constant et en espace $\mathcal{O}(| <_P |.\lceil \log_2(n) \rceil)$? Idem avec un espace en $\mathcal{O}(| \prec_P |.\lceil \log_2(n) \rceil)$ (encore plus restrictif) ? Est-ce qu'il en existe avec $\Omega(2^{n^2})$ ordres à n éléments (question de Zibin et Gil [ZG01]) ?

3. Est-ce que, pour les méthodes en temps constant, $\mid \prec_P \mid . \lceil \log_2(n) \rceil$ ou $\mid <_P \mid . \lceil \log_2(n) \rceil$ est une borne inférieure en espace pour tout ordre ? pour beaucoup d'ordres ?
4. Plus théorique : soit $EspaceMin(P)$ le plus petit espace nécessaire pour coder un ordre P fixé avec des tests en temps constant dans le modèle Word-RAM (l'espace étant la somme des informations globales et locales, plus l'espace pris par les algorithmes), est-ce qu'il existe des bornes intéressantes sur ce paramètre ? Des bornes faisant appel à des paramètres classiques des ordres ? Quelle est la complexité pour calculer $EspaceMin(P)$ et générer un codage en temps constant correspondant ?

Pour ces questions, il s'agit d'abord de questions d'existence, ensuite en cas d'existence, quelles sont les complexités de génération du codage pour un ordre fixé ? Polynômial, $\mathcal{NP}$-complet ou encore plus dur ?

Apparemment Talamo *et al* [NTV99] répondent OUI à la question 1 avec un algorithme de génération polynomial ... Nous n'avons pas encore pu étudier le détail de la méthode proposée. Si c'est bien le cas, alors la question 2 perd un peu de son sens. Cependant on pourra lui en redonner en se focalisant alors sur la complexité de génération des codages car il sera toujours intéressant d'avoir des classes d'ordres avec des méthodes respectant les bornes en espace et temps de la question 1 et pour lesquelles il existe un algorithme de génération très rapide.

Les précédentes questions font presque toutes apparaitre $\mid <_P \mid$ dans leurs bornes. Mais est-ce que $\mid <_P \mid$ est le bon paramètre à regarder ? Est-ce que c'est par rapport à ce paramètre que la frontière des méthodes en temps constant se situe ? On pourrait imaginer d'autres paramètres traçant cette frontière : par exemple $\mid \leq_P \mid / h(P)$ en borne inférieure ou bien un paramètre connu qui ressurgirait, comme $dim(P)$. Sur les méthodes présentées qui s'appliquent à tous les ordres, il y en a qui semblent bien compresser par rapport à $h(P)$ (par exemple la *partition en chaines*), d'autres plutôt par rapport à $w(P)$ (par exemple la *2-dimension*), mais il semble toujours exister pour n éléments et $\mid <_P \mid$ comparabilités un exemple d'ordre P pour lequel la méthode donne des codages en espace toujours supérieur à $\mid <_P \mid . \lceil \log_2(n) \rceil$ bits (cela reste quand même à vérifier).

Rappelons qu'il existe une classe importante à regarder pour tous ces problèmes théoriques qui concernent les méthodes de codage : les ordres bipartis. On les rencontre très souvent : soit directement pour représenter les ordres (comme les bipartis $B(P)$, $\overline{B}(P)$, $B^0(P)$ ou $\overline{B^0}(P)$ associés à un ordre P), soit en tant que sous-classe des ordres que l'on étudie. En particulier, la complexité d'un problème sur les ordres bipartis est souvent révélatrice de la complexité du problème pour l'ensemble de tous les ordres.

Rappelons aussi que pour trouver des classes avec des méthodes polynomialement optimales, on peut s'intéresser aux ordres de paramètres bornés pour une méthode fixée (ce qui implique souvent qu'il existe cette méthode est en temps constant pour ces ordres et que leur espace de stockage est en $\mathcal{O}(n \log n)$). Il s'agit donc d'étudier pour une méthode fixée les limites sur les paramètres tel que la classe d'ordres associée soit reconnaissable en temps polynomial. Les classes héréditaires (si un ordre appartient à la classe, tous ses sous-ordres aussi) sont aussi de bons candidats (voir le travail de Spinrad sur les classes héréditaires de graphes non orientés [Spi97]).

Analyses en moyenne

On notera la quasi-absence d'études théoriques sur le comportement en moyenne des méthodes de codage d'ordres. Ceci peut s'expliquer par le fait qu'il faut d'abord se fixer une distribution intéressante sur les ordres et que l'analyse de complexité en moyenne est très difficile. D'autre part les différents auteurs ont plutôt utilisé la simulation et des jeux de tests pour évaluer les performances de leurs méthodes.

Aspects dynamiques

De même, du point de vue dynamique, les questions de mise à jour du codage quand l'ordre évolue, même si elles sont souvent mentionnées dans la littérature, posent de nombreux problèmes encore ouverts.

Le problème de maintenir dynamiquement la fermeture transitive est très souvent sous-jacent (par exemple à partir du moment où un nouvel élément est donné avec juste ses prédécesseurs ou successeurs immédiats, ou si on ajoute une comparaison entre deux éléments). Or c'est un problème difficile qui a donné lieux à de nombreux travaux. Sans plus entrer dans le détail des algorithmes et structures de données utilisées, nous pouvons citer les travaux suivants [DI00, Hes01, HK95, IK83, Ita86, Ita88, KMW96, Kin99, KS99, PvL88, Yel93].

Pour les différentes méthode de codage d'ordre, on est en fin de compte confronté à deux problèmes : maintenir la fermeture transitive et maintenir les performances du codage (notamment la compression en espace).

Pour une méthode de codage fixée, si un ordre P évolue peu dans le temps (par rapport à la fréquence des requètes de comparaisons) et s'il existe un algorithme rapide de génération d'un codage, alors on peut parfois se permettre de recalculer un nouveau codage de P après chaque modification en reprenant la génération à zéro. Sinon il faut élaborer des algorithmes spécifiques de mises à jour. Ces questions de mise à jour des codages d'ordres demandent encore à être explorées, d'autant plus que les comportements dynamiques sont fréquents en pratique.

Utilisation pratique

L'utilisation pratique des codages d'ordres est un point important. Les besoins pratiques peuvent privilégier certaines méthodes au nom d'autres critères que l'espace total de stockage et le temps d'exécution de l'opération de comparaison, comme par exemple le contexte d'exécution des opérations ou la répartition des informations en mémoire. La structure des ordres rencontrés ou la manière dont évoluent les ordres dans le temps peuvent aussi favoriser certaines méthodes par rapport aux autres.

On peut remarquer que presque tous les codages présentés dans cette synthèse sont tels que pour tout couple d'éléments x,y d'un ordre codé, le coût de la requête "$x \leq_P y$?" est le même. Ceci peut être vu comme une contrainte. En effet peut être qu'au cours de l'utilisation, on répètera presque toujours les mêmes requêtes. Dans ce cas, on peut imaginer une méthode qui stockerait juste la réduction transitive pour économiser l'espace (comme la *représentation par ensembles des successeurs immédiats* utilisant les listes chainées), plus pour chaque élément un petit cache conservant les réponses aux 2 ou 3 dernières requêtes de comparaison sur cet élément. Une réponse non prévue serait calculée par un brutal parcours en profondeur, en espérant que cela arrive le moins souvent possible. Noter qu'une telle méthode se prête bien aux aspects dynamiques, les mises à jour du codage pour l'ajout d'un élément ou d'une comparabilité étant très simples. Pour l'ajout d'une comparabilité entre deux éléments x et y d'un ordre P, on peut commencer par vérifier si, avant cet ajout, on avait $x \leq_P y$. Si c'est le cas, on n'ajoute pas la comparabilité, sinon on ajoute y aux successeurs immédiats de x. Plus généralement, on pourrait imaginer une méthode qui, pour un ordre P, stockerait un graphe orienté G dont la fermeture transitive est P, suffisamment petit pour gagner de l'espace par rapport à la représentation par ensembles de successeurs et suffisamment grand pour traiter efficacement les comparaisons le plus souvent possible. Ce graphe G serait modifié au fur et à mesure des requêtes de comparaisons pour se calquer sur les requêtes les plus fréquentes.

Des informations sur l'enchainement des requètes de comparaisons sont peut être parfois exploitables pour élaborer des méthodes de codage plus efficaces.

Conclusion

De nombreuses questions ont été soulevées au cours de notre travail. L'effort de synthèse a permis de mettre en avant certains principes généraux pour les méthodes de codage d'ordres. Cela a abouti à de nouvelles propositions de méthodes. Elles méritent sûrement une analyse plus approfondie, tout comme certaines des méthodes présentées dans cette synthèse pour lesquelles un bon nombre de problèmes restent ouverts. Il serait aussi intéressant d'essayer d'inclure les autres méthodes juste mentionnées en fin de section 3.2 et de continuer à enrichir cette synthèse.

Pour guider le choix des méthodes et les comparer, il serait bon pour chaque méthode d'avoir des bornes sur les performances en fonction de paramètres classiques (comme par exemple $| \leq_P |$, $| \prec_P |$, $w(P)$, $h(P)$ ou d'autres) ou mieux d'avoir des analyses en moyenne pour de bonnes distributions.

Le contexte d'utilisation des codages joue bien sûr un rôle déterminant sur ce choix, à cause des contraintes supplémentaires qu'il peut imposer, du comportement des utilisateurs et donc des mesures de complexités les plus appropriées. C'est pourquoi des études théoriques de ce genre seront d'autant plus intéressantes que nous saurons à quels problèmes pratiques elles peuvent s'adresser.

Nous nous sommes surtout focalisé sur l'opération de comparaison (fondamentale car caractérisant totalement l'ordre), il serait aussi intéressant d'étudier d'autres opérations sur les ordres comme celles présentées dans la section 2.1. Par exemple pour générer en temps $\mathcal{O}(|Succ_P(x)|)$ l'ensemble des successeurs d'un élément x, les représentations fortement implicites pour l'opération de comparaison ne sont pas forcément très adaptées.

Le traitement des aspects dynamiques des codages reste aussi l'un des points les plus intéressants à explorer.

Deuxième partie

Une synthèse sur la 2-dimension des ordres

Chapitre 5

La 2-dimension des ordres

Cette étude effectuée avec Michel Habib, Lhouari Nourine et Olivier Raynaud poursuit l'important travail de synthèse effectué par Michel Habib et Lhouari Nourine sur le sujet, ayant abouti en particulier à identifier et regrouper plusieurs problèmes connus, équivalents au calcul de la 2-dimension d'un ordre, et à la description de plusieurs heuristiques de calcul. Ce travail a déjà fait l'objet d'un rapport de recherche [RT00] rédigé avec Olivier Raynaud et d'un chapitre dans la thèse d'Olivier Raynaud [Ray00]. Il faut ajouter la participation de Fabien de Montgolfier dans notre projet d'élaboration et d'implantation d'une nouvelle heuristique générale de codage.

Ce chapitre traite des codages des ordres dits ***codages bit-vecteur*** et associés au paramètre appelé la ***2-dimension***. Ces codages sont connus en particulier pour la résolution des problèmes suivants propres aux structures ordonnées.

Problème 1 (Comparaison). *Etant donné un ordre* $P = (X, \leq_P)$ *et* $x, y \in X$*, est-ce que* $x \leq_P y$ *?*

Problème 2 (Borne supérieure). *Etant donné un ordre* $P = (X, \leq_P)$ *et* $x, y \in X$*, calculer la borne supérieure* $x \vee_P y$ *de* x *et* y *si elle existe.*

Problème 3 (Borne inférieure). *Etant donné un ordre* $P = (X, \leq_P)$ *et* $x, y \in X$*, calculer la borne inférieure* $x \wedge_P y$ *de* x *et* y *si elle existe.*

Ces trois problèmes correspondent à trois opérations de base quand on manipule les structures ordonnées. D'un point de vue algorithmique, pour un ordre P fixé, la problématique courante est l'élaboration d'un codage de l'ordre P qui permette de répondre efficacement aux questions ci-dessus pour tout couple $x, y \in X$. A cette fin, de nombreux techniques de codages (structures de données et algorithmes correspondants) ont été proposées, c'est l'objet du chapitre précédent de la thèse. Nous nous intéressons essentiellement au problème 1 dans ce chapitre.

La section 5.1 définit le type de codages d'ordres que nous allons étudier : les ***codages par vecteur de bits*** ou ***codages bit-vecteur*** pour utiliser un anglicisme. Ce sont des codages à base de plongement des ordres dans des treillis booléens, autrement dit on cherche à représenter les éléments par des parties d'un ensemble fixé S et à se ramener ainsi à l'ordre d'inclusion sur ces parties. La taille du codage est alors définie comme le cardinal de S. Le paramètre de la ***2-dimension*** des ordres est alors défini, il correspond à la taille minimale des codages bit-vecteur d'un ordre. C'est ce paramètre qui est le nœud de cette étude puisque notre principal problème va consister à construire des codages de taille minimale.

La section 5.2 présente les critères informatiques communément considérés pour juger l'intérêt d'un codage d'ordres. Elle précise le fonctionnement des codages bit-vecteur d'un point de vue algorithmique. Les premiers arguments en faveur et en défaveur de ces codages sont avancés et un état de l'art est présenté.

La section 5.3 donne les premières propriétés concernant la 2-dimension des ordres et les codages bit-vecteur associés.

La section 5.4 montre que le problème du calcul de la 2-dimension est équivalent à plusieurs autres problèmes d'optimisation classiques. Des preuves constructives des résultats sont données.

La section 5.5 exploite ces liens entre problèmes d'optimisation pour positionner le problème de calcul de la 2-dimension d'un point de la complexité. Il était connu qu'il s'agit d'un problème $\mathcal{NP}$-complet, nous y ajoutons un résultat de non-approximabilité.

Malgré tout dans certains cas, le calcul de la 2-dimension et d'un codage bit-vecteur optimal est possible de manière efficace, ou du moins on connait des ordres de grandeur. Une liste de cas particuliers est ainsi étudiée dans la section 5.6. Face à la difficulté intrinsèque du problème, une première heuristique connue est rappelée en section 5.7 : le simple coloriage.

La section 5.8 résume le travail effectué sur la classe des arbres qui a abouti à l'élaboration d'une heuristique pour ce cas particulier : le ***codage dichotomique***. L'analyse montre les performances intéressantes de cet algorithme pour le cas des arbres.

Cet algorithme permet aussi de proposer des heuristiques pour le cas général, utilisant la décomposition modulaire et décrites en section 5.9. Dans cette section, les résultats des différentes heuristiques connues sont présentés pour un jeu de test.

Pour finir, la section 5.10 est consacrée aux applications et perspectives de ce travail. On y discute des aspects dynamiques de tels codages, brièvement car on connait peu de choses.

On trouvera dans ce chapitre à la fois des théorèmes et des algorithmes qui ont été implantés et testés sur des benchmarks.

5.1 Plongements et codages bit-vecteur

Avec les définitions des plongements données dans le chapitre 1, nous pouvons introduire la notion de ***codage bit-vecteur*** d'un ordre et le paramètre appelé ***2-dimension*** d'un ordre. Ils sont définis à base de plongements dans les treillis booléens.

Définition 5.1. *Soit $P = (X, \leq_P)$ un ordre et S un ensemble fini. Un plongement d'ordre ϕ de P dans le treillis booléen $\underline{2}^S$ est appelé un* ***codage bit-vecteur*** *de P. La* ***taille*** *de ce codage est notée taille(ϕ) et vaut $|S|$.*

La ***2-dimension*** *de P, notée $dim_2(P)$ est la taille minimum d'un codage bit-vecteur de P. Un codage bit-vecteur de P de taille $dim_2(P)$ est dit* ***optimal****.*

Les éléments de l'ensemble S sont appelés ***couleurs*** *ou* ***éléments de codage*** *ou* ***gènes****.*

Avec des notations mathématiques, une application $\phi : X \to \underline{2}^S$ est un codage bit-vecteur si et seulement si

$$\forall x, y \in X,\ x \leq_P y \iff \phi(x) \subseteq \phi(y)$$

.

Il est clair que deux ordres isomorphes admettent les mêmes codages bit-vecteur. Par contre étant donné un ordre, tous ses codages bit-vecteur optimaux ne sont pas "isomorphes" (au sens "identiques à une permutation près de S"). D'ailleurs on peut se poser la question de savoir pour quels ordres il n'existe qu'un unique codage bit-vecteur optimal.

Exemple 5.2. *Soit S ensemble fini de cardinal $n \geq 0$. Pour le treillis booléen $\underline{2}^S$, l'identité est un codage bit-vecteur optimal et $dim_2(\underline{2}^S) = n$. Quand $n = 0$, le treillis $\underline{2}^S$ est réduit à un singleton (ordre qui est parfois noté $\underline{1}$) qui peut être codé avec $\emptyset$ et on a bien $dim_2(\underline{1}) = 0$.*

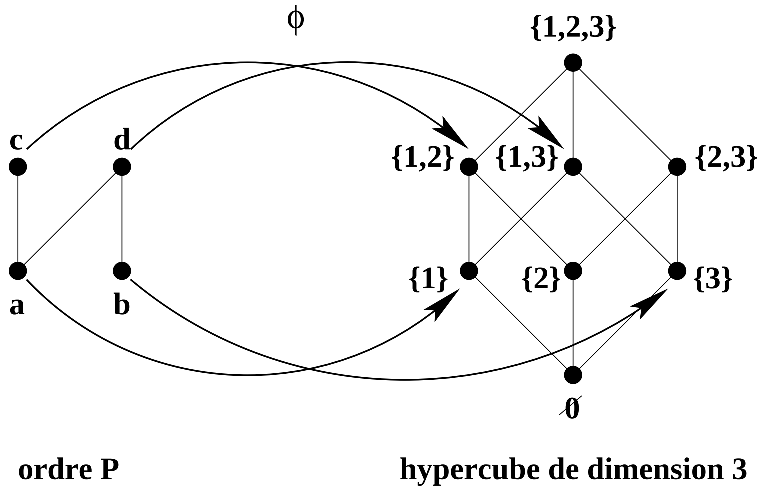


FIG. 5.1 – Un plongement de l'ordre P dans un hypercube de dimension 3

Définition 5.3. *Soit $P = (X, \leq_P)$ un ordre, S un ensemble fini et $\phi : X \to \underline{2}^S$ un codage bit-vecteur. Alors le* ***codage réduit*** *ϕ_r associé à ϕ est l'application $\phi_r : X \to \underline{2}^S$ définie pour tout $x \in X$ par : $\phi_r(x) = \phi(x) \setminus \bigcup_{y <_P x} \phi(y)$. Le codage initial ou* ***codage total*** *ϕ se réobtient alors en propageant les codes réduits des éléments inférieurs ou égaux, autrement dit $\phi(x) = \bigcup_{y \leq_P x} \phi_r(y)$.*

Remarque 5.4. *A partir d'un codage réduit quelconque, c'est à dire d'une application quelconque f de X dans $\underline{2}^S$, on peut toujours construire une application croissante de X dans $\underline{2}^S$ par $\phi(x) = \bigcup_{y \leq_P x} f(y)$ mais on n'a pas toujours un plongement car l'implication $\phi(x) \subseteq \phi(y) \Rightarrow x \leq_P y$ n'est pas toujours vérifiée.*

Par la suite, on préférera utiliser le terme "couleur" pour désigner les éléments de codage pour éviter toute confusion avec les éléments de l'ordre (même s'ils peuvent être parfois identiques comme dans le cas du codage bit-vecteur canonique). Le terme "couleur" vient d'une équivalence du problème de la 2-dimension avec le problème de coloration des graphes [Sko92]. Le terme "gène" vient de la notion de codage réduit et d'héritage : le code d'un élément hérite de tous les codes réduits de ses prédécesseurs.

La figure 5.2 montre trois différentes représentations pour un même codage bit-vecteur de taille $|S| = 4$ pour un ordre P. La première associe à chaque élément x de P son code $\phi(x)$ qui est une partie de $S = \{1, 2, 3, 4\}$. La deuxième montre le code réduit $\phi_r(x)$ de chaque élément x. La dernière

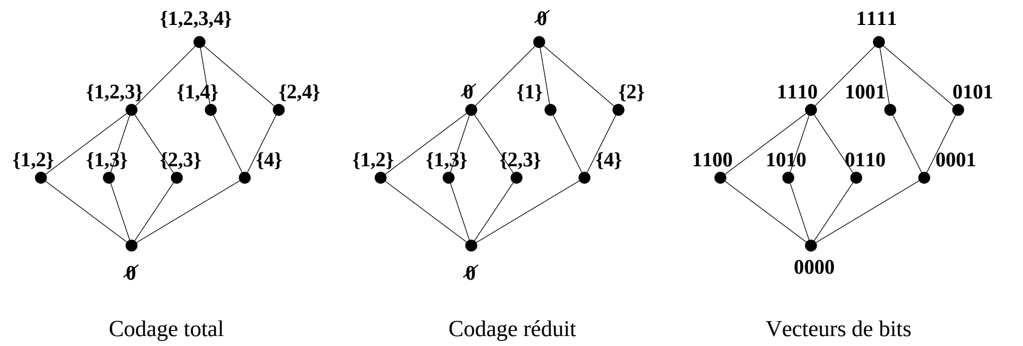


FIG. 5.2 – Les trois différentes représentations d'un codage bit-vecteur

associe à chaque élément x un vecteur de $|S|$ bits où le bit i vaut 1 si $i \in \phi(x)$ et 0 sinon. On omettra parfois de mettre $\emptyset$ sur les dessins pour le code vide.

Pour finir cette présentation, nous donnons les codages bit-vecteur les plus classiques.

Exemple 5.5. *Soit $P = (X, \leq_P)$ un ordre. L'application $\phi : X \rightarrow \underline{2}^X$ définie par $\phi(x) = \downarrow_P x$ est un codage bit-vecteur de taille $|P|$. On l'appellera **codage bit-vecteur canonique**.*

Soit $T = (X, \leq_T)$ un treillis. L'application $\phi_\wedge : X \rightarrow 2^{J(T)}$ (resp. $\phi_\vee : X \rightarrow 2^{M(T)}$) définie par $\phi_\wedge(x) = J_x$ (resp. $\phi_\vee(x) = M(T) \backslash M_x$) est un $\wedge$-plongement (resp. un $\vee$-plongement). Par conséquent, ces deux applications $\phi_\wedge$ et $\phi_\vee$ sont des codages bit-vecteur de T de tailles respectives $|J(T)|$ et $|M(T)|$.

La preuve que les deux derniers codages constituent respectivement un $\wedge$-plongement et un $\vee$-plongement utilise le lemme 1.1 et le fait que $J_{x \wedge y} = J_x \cap J_y$ et $M_{x \vee y} = M_x \cup M_y$ d'après la définitions des bornes inférieures et supérieures.

Bien entendu le problème va consister à générer des codages bit-vecteur de taille aussi petite que possible, l'idéal étant d'atteindre la 2-dimension. Par exemple, le codage canonique (qui a l'intérêt de prouver qu'il existe bien des codages bit-vecteur pour tout ordre) a une taille de n bits pour un ordre à n éléments, ce qui est identique à l'utilisation de la matrice binaire. Mais on peut espérer faire mieux dans de nombreux cas et rivaliser avec les autres méthodes utilisées pour coder les ordres.

La section suivante fait un bref état de l'art sur les codages bit-vecteur et la 2-dimension.

5.2 Un état de l'art

Du point de vue informatique, les méthodes de résolution des problèmes 1, 2 ou 3 sont souvent jugées selon une liste de critères qui ont été exposés dans le Chapitre 2. Une des fiches du Chapitre 2 présente la méthode de codage par vecteur de bits avec sa description et son implantation. Rappelons que les codages bit-vecteur se prètent très bien à un traitement informatique. Comme on l'a vu, soient $P = (X, \leq_P)$ et ϕ un plongement d'ordre de P dans $\underline{2}^k$, le code $\phi(x)$ de chaque élément $x \in X$ peut se représenter par un vecteur de k bits et $x \leq_P y$ si et seulement si pour tout i, $1 \leq i \leq k$, le i-ème bit de $\phi(x)$ est inférieur au i-ème bit de $\phi(y)$. Une manière d'implanter cette comparaison consiste à utiliser l'opérateur `OU` sur les vecteurs de bits, une comparaison revient alors au test $\phi(x)$ `OU` $\phi(y) = \phi(y)$. Un tel test s'effectue en temps $\mathcal{O}(k/w)$ dans le modèle Word-RAM avec des mots de w bits. Ce test peut s'écrire avec un nombre constant d'instructions indépendantes de l'ordre codé (avec une boucle), on a donc affaire à une méthode de codage uniforme. L'espace total de stockage est en $\mathcal{O}(kn)$ bits, où n est le nombre d'èléments de P. Nous pouvons aussi donner la complexité du passage du codage réduit d'un ordre à son codage total lorsqu'on utilise cette implantation par vecteurs de bits. Si l'ordre P est donné par sa matrice de fermeture transitive, alors le passage d'un codage réduit ϕ_r au codage total ϕ peut s'effectuer en $\mathcal{O}(n^2)$ opérations `OU`, en utilisant la formule $\phi(x) = \bigcup_{y \leq_P x} \phi_r(y)$. Si l'ordre P est donné par ses ensembles de successeurs ou de prédécesseurs, le passage peut s'effectuer en $\mathcal{O}(| \leq_P |)$ opérations `OU` à l'aide de cette même formule, avec $| \leq_P |$ qui désigne le nombre de comparabilités dans P. Si l'ordre P est donné par sa réduction transitive (soit ses ensembles de successeurs immédiats, soit ses ensembles de prédécesseurs immédiats), alors le passage d'un codage réduit ϕ_r au codage total ϕ peut s'effectuer en $\mathcal{O}(| \prec_P |)$ opérations `OU` à partir de la formule $\phi(x) = \bigcup_{y \in ImPred_P(x)} \phi_r(y)$, avec $| \prec_P |$ qui désigne le nombre d'arcs de la réduction transitive. Nous revenons maintenant sur l'historique des travaux concernant les codages bit-vecteur et la 2-dimension.

La 2-dimension est un paramètre qui a d'abord été étudié du point de vue mathématique. En 1941, Dushnik et Miller introduisent la notion de ***dimension*** d'un ordre comme le nombre minimum d'extensions linéaires dont l'intersection redonne l'ordre initial. Puis Ore caractérise la dimension d'un ordre P comme le nombre minimum de chaines (de hauteurs quelconques) tel qu'il existe un plongement de P dans le produit direct de ces chaines [Ore62]. Novak étend alors ce concept en définissant la ***k-dimension*** d'un ordre P [Nov63] comme le nombre minmum de chaines toutes de hauteur $k-1$ telles qu'il existe un plongement de P dans le produit direct de ces chaines. La 2-dimension en est un cas particulier. Parmi les premiers articles à étudier les propriétés de la 2-dimension, on peut citer le travail de Trotter [Tro75]. On peut aussi trouver quelques résultats concernant la k-dimension des ordres dans [Tro91].

Ces plongements dans un produit de chaines de hauteur 1 sont équivalents à représenter les éléments d'un ordre par des ensembles et les munir de la relation d'inclusion. Les travaux de Markowski [Mar80] traitent de cette représentation des ordres et des treillis par des ensembles. Plus généralement, il donne plusieurs théorèmes qui précisent les conditions d'existence de plongements, $\wedge$-plongements et $\vee$-plongements pour les ordres et les treillis.

Les travaux de Bouchet [Bou71, Bou84] s'intéressent aussi à ces notions de codage d'ordres et plus généralement de relations binaires. Il y étudie les plongements dans des produits de chaines et se penche en particulier sur le cas des treillis, des $\vee$-plongements et des $\wedge$-plongements.

Parmi les premières utilisations des codages bit-vecteur du point de vue pratique, on peut citer les travaux de Aït-Kaci et al [AKBLN89] qui s'intéressent au traitement informatique des opérations sur les treillis et qui utilisent les applications $\phi_\vee$ et $\phi_\wedge$ définies dans l'exemple 5.5.

Il existe d'ailleurs un résultat d'optimalité pour le calcul des bornes supérieure (et inférieures) présenté par Bouchet [Bou71] et qui peut aussi être vu comme un corollaire d'un théorème plus général de Markowski (théorème 2.3 conséquence du théorème 2.4 dans [Mar80]). Ce résultat est le suivant, il résout le problème 2 et de manière duale le problème 3 quand on utilise des codages bit-vecteur, on cherche ici un $\vee$-plongement (resp. un $\wedge$-plongement) du treillis initial dans un treillis booléen du taille minimum.

Théorème 5.6. *Soit $T=(X,\leq_T)$ un treillis. Tout $\vee$-plongement (resp. $\wedge$-plongement) de T dans un treillis booléen $\underline{2}^S$ est de taille $|S| \geq |M(T)|$ (resp. $|S| \geq |J(T)|$). Le codage $\phi_\vee$ (resp. $\phi_\wedge$) défini dans l'exemple 5.5 est un $\vee$-plongement (resp. Inf-plongement) de taille $|M(T)|$ (resp. $J(T)|$) donc optimal. C'est l'unique au sens où s'il existe un $\vee$-plongement ϕ_1 (resp. $\wedge$-plongement) optimal de T dans $\underline{2}^{S_1}$ et un autre optimal ϕ_2 de T dans $\underline{2}^{S_2}$, alors il existe une bijection π de S_1 sur S_2 telle que pour tout $x \in X$, $\phi_2(x) =$*

$\{\pi(s)|s \in \phi_1(x)\}$.

Des synthèses sur les codages bit-vecteur ont été réalisées par Habib et Nourine dans [HN94] et [HN95], ainsi que par Fall [Fal98]. En particulier, il a été mis en évidence que l'optimisation de la taille d'un codage bit-vecteur se ramène à d'autres problèmes d'optimisation combinatoire classiques (on y reviendra dans les sections 5.4 et 5.5). Il a ainsi été démontré que le calcul de la 2-dimension d'un ordre est associé à un problème de décision $\mathcal{NP}$-complet [SW84].

Plusieurs heuristiques de génération de codages bit-vecteur ont été proposées. La plus connue est la simple coloration proposée par Caseau [Cas93] dans le cadre de la programmation orientée objets. Revue par Krall, Vitek et Horspool [KVH97], cette heuristique a été étudiée en détail par Habib, Nourine et Raynaud dans [HNR97] et [CHNR99], articles qui synthétisent caractérisations, résultats de complexité et corrections des travaux antérieurs pour cette heuristique.

Ces codages sont particulièrement appropriés pour les treillis, notamment en ce qui concerne la compression en espace.

5.3 Premières propriétés de la 2-dimension

Cette section énumère quelques propriétés concernant la 2-dimension. Ce sont des propriétés connues qui s'apparentent à celles qui traitent de la ***dimension*** des ordres, si ce n'est que les propriétés énoncées ici sont bien plus simples à montrer que celles issues des travaux d'Hiragushi pour la dimension. Hiragushi a étudié dans les années 50 la dimension des ordres [Hir55] en proposant plusieurs égalités et inégalités (on peut les retrouver dans le livre de Trotter [Tro91] qui est dédié à l'étude de la dimension des ordres).

Proposition 5.7. *Soit* $P = (X, \leq_P)$ *un ordre. Alors son dual* P^d *vérifie :* $dim_2(P^d) = dim_2(P)$.

Preuve. Etant donné un plongement d'ordre ϕ de P dans $\underline{2}^S$, on vérifie facilement que ϕ' définie par $\phi'(x) = S \backslash \phi(x)$ est un plongement d'ordre de P^d dans $\underline{2}^S$. On passe de même de P^d à P puisque $(P^d)^d = P$. ■

Par conséquent le choix de l'orientation des dessins ("du bas vers le haut" ou "du haut vers le bas") n'a pas d'importance a priori dans l'étude de la 2-dimension et des codages bit-vecteur. Malgré tout, parfois, on devine plus facilement des codages bit-vecteur d'un ordre, pour une des deux orientations. Ceci peut s'expliquer par le fait que, intuitivement, on cherche plutôt un codage réduit de l'ordre. Or le codage réduit ne se comporte pas pareil dans les deux sens (contrairement au codage total, où il suffit de passer au complémentaire pour coder le dual). On pourra le vérifier dans le cas des peignes (étudié dans la section 5.6.5).

Par défaut, on conserve l'orientation classique "du bas vers le haut" pour le dessin des ordres.

Le proposition suivante sur la monotonie de la 2-dimension est évidente.

Proposition 5.8. *Soit $P = (X, \leq_P)$ un ordre et Q un sous-ordre de P. Alors $dim_2(Q) \leq dim_2(P)$.*

De même, la proposition suivante est évidente par composition de plongements d'ordres.

Proposition 5.9. *Soit $Q = (Y, \leq_Q)$ et $P = (X, \leq_P)$ deux ordres tels qu'il existe un plongement d'ordre de Q dans P. Alors $dim_2(Q) \leq dim_2(P)$.*

Comme cela a été mentionné dans l'état de l'art, les codages bit-vecteur sont très appropriés pour le codage des treillis. C'est vrai si l'on cherche des codages préservant les bornes supérieures ou les bornes inférieures, mais aussi pour simplement répondre à la question de comparabilité. Des liens très forts unissent les codages bit-vecteur des ordres et ceux des treillis. C'est le fond de la proposition suivante.

Pour tout ordre P, il existe un treillis appelé ***treillis de Dedekind-MacNeille*** de P [Mac37] et noté $DM(P)$ vérifiant :

1. il existe un plongement d'ordre ϕ de P dans $DM(P)$,
2. pour tout treillis T tel qu'il existe un plongement de P dans T, alors il existe un plongement de $DM(P)$ dans T.

La condition 2 assure l'unicité d'un tel treillis, à un isomorphisme près. Par conséquent, $DM(P)$ est le plus petit treillis (en terme de nombre de sommets) dans lequel se plonge P. Pour une preuve de l'existence d'un tel treillis, consulter par exemple le livre de Davey et Priestley [DP91].

Proposition 5.10. *Soit $P = (X, \leq_P)$ un ordre et $DM(P)$ son complété de Dedekind-MacNeille. Alors $dim_2(DM(P)) = dim_2(P)$.*

Preuve. C'est une conséquence de la propriété 2 de la définition de $DM(P)$. Une preuve constructive sera aussi donnée dans le théorème 5.24 de la section 5.4 : on montrera comment construire un plongement de $DM(P)$ dans $\underline{2}^S$ à partir d'un plongement de P dans $\underline{2}^S$.

On notera que ce résultat est aussi présent dans [Bou71]. ■

Proposition 5.11. *Soit* $P = (X, \leq_P)$ *un ordre. Alors*

$$\lceil \log_2(|P|) \rceil \leq dim_2(P) \leq |P|.$$

Preuve. Soit ϕ un plongement d'ordre de P dans $\underline{2}^S$. Comme ϕ est injective et que le cardinal de $\underline{2}^S$ vaut $2^{|S|}$, on a $|P| \leq 2^{|S|}$ et donc la borne inférieure sur la 2-dimension.

D'autre part, le plongement canonique ϕ défini par $\phi(x) = \downarrow_P x$ est un codage bit-vecteur de taille $|P|$, ce qui donne la borne supérieure sur la 2-dimension. ■

Les cas d'égalité $dim_2(P) = |P|$ seront présentés dans la section 5.6.2.

Proposition 5.12. *Soit* $\mathcal{C}_n$ *un ordre total à* n *éléments. Alors*

$$dim_2(\mathcal{C}_n) = n - 1.$$

Preuve. Soient $x_0 < x_1 < \ldots < x_{n-1}$ les n éléments de $\mathcal{C}_n$ et ϕ un codage bit-vecteur optimal de $\mathcal{C}_n$ à k couleurs. Alors $\phi(x_0) = \emptyset$ (sinon en considérant les codes privés de $\phi(x_0)$ on aurait un nouveau codage de taille strictement inférieure) et $\phi(x_0) \subsetneq \phi(x_1) \subsetneq \ldots \subsetneq \phi(x_{n-1})$. Par conséquent $0 = |\phi(x_0)| < |\phi(x_1)| < \ldots < |\phi(x_{n-1})|$ et la taille du codage qui vaut $|\phi(x_{n-1})|$ est nécessairement supérieure ou égale à $n - 1$.

Or il existe bien un codage de taille $n - 1$: associer à x_i le code $\phi(x_i) = \{1, \ldots, i\}$ si $i \geq 1$ et $\emptyset$ si $i = 0$. La figure 5.3 le représente pour $n = 5$. ■

La proposition suivante donne la 2-dimension d'une antichaine. Elle utilise la fonction $sp(n) = \min\{k \mid C_k^{\lfloor k/2 \rfloor} \geq n\}$ pour $n \geq 0$ qui est en fait la largeur du treillis booléen $\mathcal{B}_n$. Ce résultat obtenu par Sperner en 1928 est à l'origine de ce qu'on appelle la théorie de Sperner qui étudie les structures ordonnées du point de vue de l'analyse combinatoire (dénombrement de configurations particulières, problèmes extrémaux, inégalités remarquables). Pour des preuves et plus d'informations, on peut consulter les ouvrages de Engel et Gronau [EG85] (5 preuves différentes du théorème de Sperner), Engel [Eng97] ou Anderson [And87].

Proposition 5.13 (Sperner (Spe1928)). *Soit* $\mathcal{A}_n$ *une antichaine à* n *éléments. Alors* $dim_2(\mathcal{A}_n) = sp(n)$*. Un codage bit-vecteur optimal de* $\mathcal{A}_n$ *consiste à associer à chaque élément un sous-ensemble de* $\{1, \ldots, k\}$ *de cardinal* $\lfloor k/2 \rfloor$*, où* $k = sp(n)$*.*

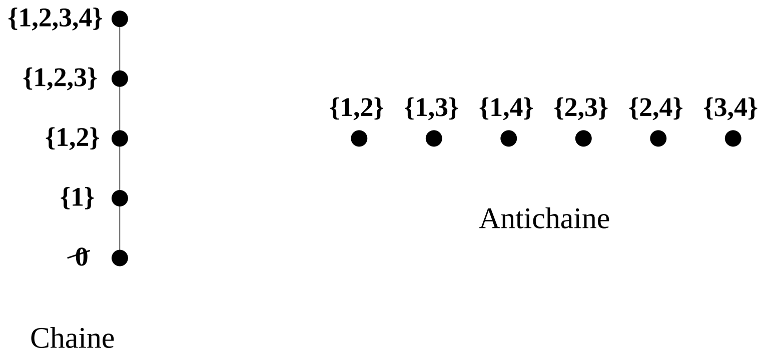


FIG. 5.3 – Codage optimaux pour une chaine à 5 éléments et une antichaine à 6 éléments

La figure 5.3 représente un codage optimal d'une antichaine à 6 éléments, il utilise $sp(6) = 4$ couleurs.

La fonction $sp(n)$ se comporte à peu près comme $\log_2(n)$. La proposition suivante précise cette évaluation et donne le comportement asymptotique de cette fonction.

Proposition 5.14. *Soit $n \geq 1$. Alors $\lceil \log_2(n) \rceil \leq sp(n) \leq 2\lceil \log_2(n) \rceil$. Quand $n \to \infty$, la fonction $sp(n)$ est équivalente à $\log_2(n)$. Plus précisément, $sp(n) = \log_2(n) + 1/2 \log_2(\log_2(n)) + \mathcal{O}(1)$.*

Preuve. La borne inférieure est celle de la proposition 5.11. Pour montrer que $sp(n) \leq 2\lceil \log_2(n) \rceil$, on peut soit faire une preuve d'analyse en montrant que $C_{2\lceil \log_2(n) \rceil}^{\lceil \log_2(n) \rceil} \geq n$, soit proposer un codage bit-vecteur de taille $2\lceil \log_2(n) \rceil$. On peut construire récursivement un tel codage :

- Si $n = 1$, on donne le code vide à cette antichaine réduite à un singleton.
- Sinon on divise l'antichaine $\mathcal{A}_n$ en deux antichaines $\mathcal{A}_{n_1}$ et $\mathcal{A}_{n_2}$ de largeurs respectives $n_1 = \lceil n/2 \rceil$ et $n_2 = \lfloor n/2 \rfloor$, on code indépendamment $\mathcal{A}_1$ et $\mathcal{A}_2$ c'est à dire on construit (récursif) un codage de $\mathcal{A}_1$ avec $2\lceil \log_2(n_1) \rceil$ couleurs et un codage de $\mathcal{A}_2$ avec $2\lceil \log_2(n_2) \rceil$ couleurs prises parmi les couleurs du codage de $\mathcal{A}_1$. Pour finir on prend deux nouvelles couleurs, on ajoute la première à tous les codes de $\mathcal{A}_1$ et l'autre à tous les codes de $\mathcal{A}_2$.

Le codage construit est de taille $2\lceil \log_2(n) \rceil$.

Pour obtenir un développement asymptotique de $sp(n)$, passer au continu : $t \in \mathbb{R}$ et $t! = \Gamma(t+1)$ où Γ est la fonction Gamma. Résoudre

$C_t^{t/2} = n$ en essayant de donner une valeur asymptotique pour t en fonction de n : utiliser la formule de Stirling pour avoir un équivalent de $C_t^{t/2}$ et écrire t comme $t = \log_2(n) + 1/2\log_2(\log_2(n)) + u$, on arrive ainsi à $u = o(1)$. En repassant au discret cela donne le $\mathcal{O}(1)$. ■

Les questions du calcul exact de $sp(n)$ et de la complexité de la génération complète d'un codage bit-vecteur optimal de $\mathcal{A}_n$ sont intéressantes.

Question 1. *Nous conjecturons qu'une meilleure analyse fournit pour tout n l'encadrement suivant :* $\log_2(n) + 1/2\log_2(\log_2(n)) \leq sp(n) \leq \log_2(n) + 1/2\log_2(\log_2(n)) + 2$. *Peut-on préciser cet encadrement ? En particulier, donner une formule close ou bien un algorithme efficace pour calculer $sp(n)$. Quelle est ensuite la complexité de la génération complète d'un codage de $\mathcal{A}_n$?*

Du fait de la monotonie de la 2-dimension (proposition 5.8) et comme tout ordre P contient une chaine de hauteur $h(P)$ (c'est à dire à $h(P)+1$ éléments) et une antichaine de largeur $w(P)$, les propositions 5.12 et 5.13 fournissent la borne inférieure suivante.

Corollaire 5.15. *Soit $P = (X, \leq_P)$ un ordre de hauteur $h(P)$ et largeur $w(P)$. Alors $dim_2(P) \geq \max(h(P), sp(w(P)))$.*

Dans le cas des treillis, on peut produire une borne supérieure meilleure que celle de la proposition 5.11, en fonction du nombre d'éléments irréductibles.

Proposition 5.16. *Soit $T = (X, \leq_T)$ un treillis. Alors*

$$dim_2(T) \leq \min(|J(T)|, |M(T)|)$$

Preuve. Comme on l'a vu dans l'exemple 5.5, l'application $\phi_\vee$ (resp. $\phi_\wedge$) est un codage bit-vecteur de taille $|M(T)|$ (resp. $|J(T)|$). ■

Proposition 5.17. *Soit $P = (X, \leq_P)$ un ordre, soit $x \in X$ et $P - x$ l'ordre induit par P sur $X \backslash \{x\}$. Alors $dim_2(P) \leq dim_2(P-x)+2$. Si x est maximal (resp. minimal) dans P et si $P-x$ n'a pas de maximum (resp. de minimum), alors $dim_2(P) \leq dim_2(P-x)+1$.*

Preuve. Soit ϕ un plongement de $P-x$ dans $\underline{2}^S$. Soit $x \in X$ quelconque. On peut construire un codage Φ de P à partir de ϕ en ajoutant deux couleurs c et c' à S, de la manière suivante. On fait les modifications sur le codage réduit. Pour tout y comparable à x, prendre $\Phi_r(y) = \phi_r(y)$. Pour tout y incomparable

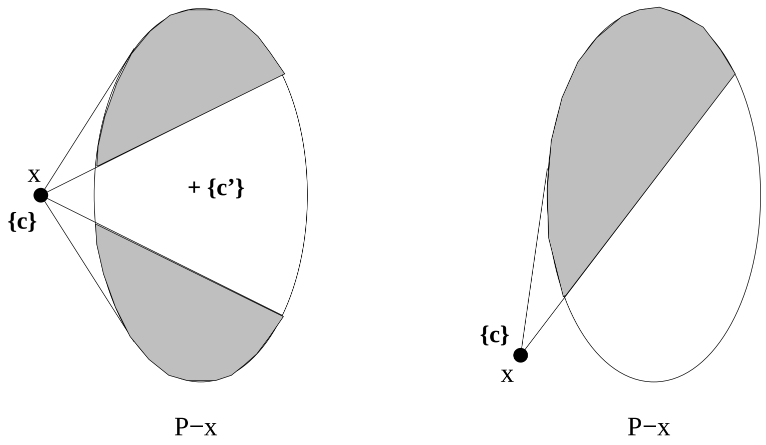


FIG. 5.4 – Construction d'un codage réduit de P à partir d'un codage réduit de $P - x$

à x, prendre $\Phi_r(y) = \phi_r(y) \cup \{c'\}$. Pour finir, prendre $\Phi_r(x) = \{c\}$. Ces ajouts aux codes réduits sont schématisés sur la figure 5.4. On vérifie facilement que Φ_r est le codage réduit d'un codage bit-vecteur de taille $|S| + 2$.

Si $x \in X$ est minimal et P n'a pas de minimum , n'utiliser qu'une seule nouvelle couleur c. Pour tout $y \neq x$, prendre $\Phi_r(y) = \phi_r(y)$ et prendre $\Phi_r(x) = \{c\}$. Aucun des $\phi(y)$ pour $y \neq x$ n'est égal à $\emptyset$, sinon $P - x$ aurait un minimum, cela permet de vérifier que Φ_r fournit bien un codage bit-vecteur de P.

Si $x \in X$ est maximal et P sans maximum, passer par le dual de l'ordre.

Le cas où x est maximal ou minimal est traité par Trotter dans [Tro75]. La figure 5.5 montre un exemple d'ordre P et d'élément x tels que $dim_2(P) = dim_2(P - x) + 2$ (seuls les codes réduits sont indiqués). ■

Corollaire 5.18. *Soit $P = (X, \leq_P)$ un ordre et $Min(P)$ l'ensemble des éléments minimaux de P. Alors $dim_2(P) \leq_P |Min(P)| + dim_2(P \backslash Min(P))$.*

Soit $P = (X, \leq_P)$ un ordre et $Y \subseteq X$, on note $P|Y$ l'ordre induit par P sur Y et $P \backslash Y$ l'ordre induit par P sur $X \backslash Y$. On aimerait pouvoir exprimer ou au moins borner $dim_2(P)$ en fonction de $dim_2(P|Y)$ et $dim_2(P \backslash Y)$. Mais a priori il est difficile de relier ces valeurs dans le cas général, à moins que Y ait une structure particulière dans P. On va voir que c'est parfois possible si Y a une structure de module (la définition des modules sera donnée dans la section 5.9). C'est ce que formulent les propositions suivantes. On en redonne

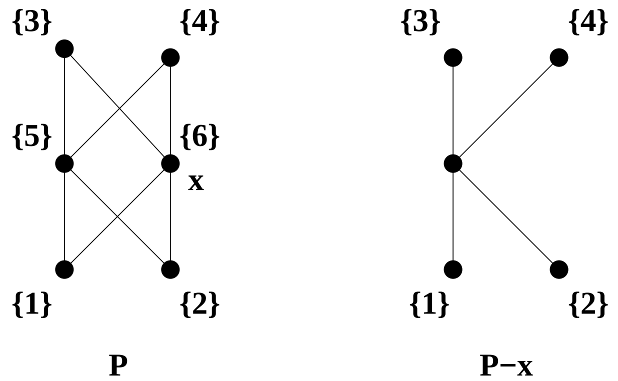


FIG. 5.5 – Un exemple où l'égalité $dim_2(P) = dim_2(P - x) + 2$ est atteinte

les preuves, car elles contiennent les constructions permettant de générer des codages bit-vecteur d'un ordre à partir de codages de certains de ses sous-ordres.

Proposition 5.19 (Trotter [Tro75]). *Soient P et Q deux ordres. Alors $dim_2(P \oplus Q) = dim_2(P) + dim_2(Q)$ à moins que P possède un élément maximum $\top_P$ et que Q possède un élément minimum $\bot_Q$. Dans ce cas, $dim_2(P \oplus Q) = dim_2(P) + dim_2(Q) + 1$.*

Preuve. Soit Φ un plongement de $P \oplus Q$ dans $\underline{2}^S$. On note A (resp. B) l'ensemble des couleurs des codes réduits des éléments de P (resp. Q). On vérifie facilement que A et B forment une partition de S (la définition de $P \oplus Q$ implique pour tout x dans Q que $A \subseteq \Phi(x)$ et donc $\Phi_r(x) \cap A = \emptyset$). Du plus on vérifie que Φ_r restreint aux éléments de P forme un codage réduit de P à $|A|$ couleurs et restreint à Q forme un codage réduit de Q à $|B|$ couleurs. Par conséquent $dim_2(P \oplus Q) \geq dim_2(P) + dim_2(Q)$.

Supposons que P possède un maximum $\top_P$ et Q possède un minimum $\bot_Q$. On peut préciser encore cette borne car nécessairement $\Phi(\top_P) = A$, donc $A \subsetneq \Phi(\bot_Q)$ et $\Phi_r(\bot_Q) \neq \emptyset$. Alors l'application ϕ_r qui associe à tout x dans Q, $\phi_r(x) = \Phi_r(x) \backslash \Phi_r(\bot_Q)$ est un plongement de taille $|B| - |\Phi_r(\bot_Q)| \leq |B| - 1$. Donc $|B| - 1 \geq dim_2(Q)$ et par conséquent $dim_2(P \oplus Q) \geq dim_2(P) + dim_2(Q) + 1$. Réciproquement, soit ϕ (resp. ϕ') un plongement de P (resp. Q) dans $\underline{2}^S$ (resp. $\underline{2}^{S'}$) avec S et S' disjoints. Introduisons une nouvelle couleur c et définissons l'application Φ de $P \oplus Q$ dans $\underline{2}^{S \cup S' \cup \{c\}}$ par $\Phi(x) = \phi(x)$ si x est dans P et $\Phi(x) = S \cup \{c\} \cup \phi'(x)$ si x est dans Q. On vérifie qu'il s'agit bien d'un plongement, de taille $|S| + |S'| + 1$, et donc $dim_2(P \oplus Q) \leq$

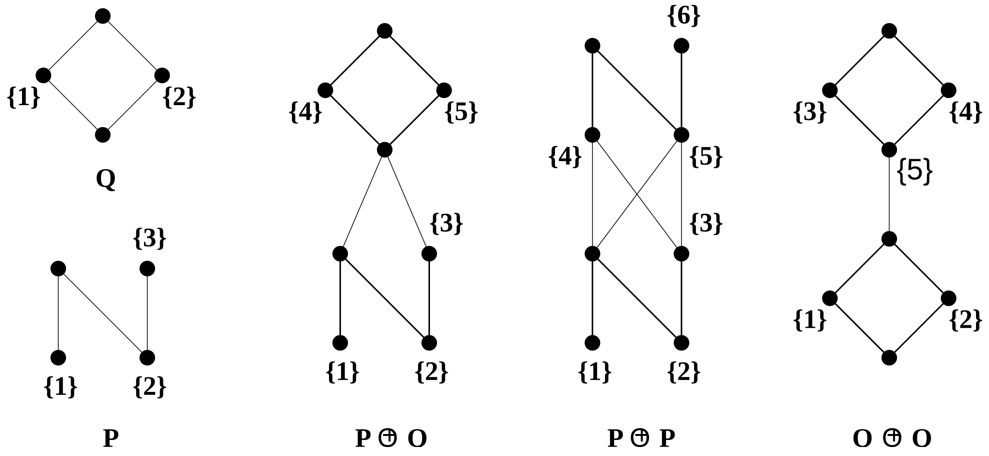


FIG. 5.6 – Construction d'un codage réduit de $P \oplus Q$ avec des codages de P et Q

$dim_2(P) + dim_2(Q) + 1$.

Supposons maintenant que Q ne possède pas de minimum. Soit ϕ (resp. ϕ') un plongement de P (resp. Q) dans $\underline{2}^S$ (resp. $\underline{2}^{S'}$) avec S et S' disjoints. Définissons l'application Φ de $P \oplus Q$ dans $\underline{2}^{S \cup S'}$ par $\Phi(x) = \phi(x)$ si x est dans P et $\Phi(x) = S \cup \phi'(x)$ si x est dans Q. On vérifie clairement que Φ restreint à P est un plongement ainsi que Φ restreint à Q. De plus, comme Q ne possède pas d'élément minimum, pour tout x dans Q, $\phi'(x) \neq \emptyset$. On en déduit que pour x dans P et y dans Q, $\Phi(x) \subsetneq \Phi(y)$. Donc Φ est bien un plongement d'ordre de taille $|S| + |S'|$ et $dim_2(P \oplus Q) \leq dim_2(P) + dim_2(Q)$.

Supposons que P ne possède pas de maximum. On fait la même construction à partir de plongements de P et de Q. Le dernier argument est que comme P ne possède pas d'élement maximum, pour tout x dans P, $\phi(x) \subsetneq S$. Par conséquent, on a des inclusions strictes entre les codes de P et ceux de Q, donc un plongement de $P \oplus Q$ et $dim_2(P \oplus Q) \leq dim_2(P) + dim_2(Q)$.

La figure 5.6 résume ces constructions (on a juste noté les codes réduits). ■

Un ordre P est ***premier pour l'opération série*** s'il existe pas deux ordres P' et P'' tels que $|P'| \geq 1$, $|P''| \geq 1$ et $P = P' \oplus P''$.

Corollaire 5.20 (Trotter [Tro75]). *Soient P_1,P_2,...,P_n n ordres, tous premiers pour l'opération série, $n \geq 2$. Alors $dim_2(P_1 \oplus P_2 \oplus \ldots \oplus P_n) = s + \sum_{1 \leq i \leq n} dim_2(P_i)$, où s désigne le nombre d'indices $i \leq n-1$ tels que*

$|P_i| = |P_{i+1}| = 1$.

L'étude de l'opération série permet de traiter le cas du produit direct.

Proposition 5.21 (Trotter [Tro75]). *Soient P et Q deux ordres. Alors $dim_2(P \times Q) \leq dim_2(P) + dim_2(Q)$. Si de plus P a un élément maximum et Q a un élément minimum (ou l'inverse), alors $dim_2(P \times Q) = dim_2(P) + dim_2(Q)$.*

Preuve. Soit ϕ_1 un plongement de P dans $\underline{2}^{S_1}$ et ϕ_2 un plongement de Q dans $\underline{2}^{S_2}$. Prendre S_1 et S_2 disjoints et construire l'application Φ de $P \times Q$ dans $\underline{2}^{S_1 \cup S_2}$ telle que pour $(x, y) \in P \times Q$, $\Phi((x, y)) = \phi_1(x) \cup \phi_2(y)$. On vérifie facilement qu'il s'agit bien d'un plongement d'ordre. On en déduit l'inégalité.

Supposons que $P = (X, \leq_P)$ a un maximum $\top_P$ et $Q = (Y, \leq_Q)$ un minimum $\bot_Q$. Alors on vérifie facilement que $P \times \{\bot_Q\}$ est un sous-ordre de $P \times Q$ qui est isomorphe à P, de même $\{\top_P\} \times Q$ est un sous-ordre isomrphe à Q. Ces deux sous-ordres ont un unique élément en commun qui est $(\top_P, \bot_Q)$ et pour tout $x \in X$ et $y \in Y$, $(x, \bot_Q) \leq_{P \times Q} (\top_P, y)$. Par conséquent en notant $\underline{1}$ l'ordre réduit à un singleton, on constate que $P \times Q$ contient un sous-ordre isomorphe à $(P - \top_P) \oplus \underline{1} \oplus (Q - \bot_Q)$. Du point de vue de la 2-dimension, on a :

$$\begin{aligned} dim_2((P - \top_P) \oplus \underline{1} \oplus (Q - \bot_Q)) &= dim_2((P - \top_P) \oplus \underline{1} \oplus \underline{1} \oplus (Q - \bot_Q)) - 1 \\ &= dim_2(P \oplus Q) - 1 \\ &= dim_2(P) + dim_2(Q) + 1 - 1 \end{aligned}$$

La première égalité est une conséquence du corollaire 5.20 et la troisième égalité une conséquence de la proposition 5.19. ■

La proposition suivante donne une borne sur le comportement de la 2-dimension par rapport à l'opération de substitution.

Proposition 5.22. *Soit $P = (X, \leq_P)$ un ordre à n éléments et n ordres $Q_1 = (X_1, \leq_{Q_1})$, ... , $Q_n = (X_n, \leq_{Q_n})$. Alors $dim_2(P[Q_1, \ldots, Q_n]) \leq dim_2(P) + \sum_{1 \leq i \leq n} dim_2(Q_i)$.*

Preuve. Considérer des plongements ϕi de $Q_i = (X_i, \leq_{Q_i})$ dans $\underline{2}^{S_i}$ où $|S| = dim_2(Q_i)$. Soit $P = (X, \leq_P)$ avec $X = \{x_1, \ldots, x_n\}$ et ϕ un plongement de P dans $\underline{2}^S$ où $|S| = dim_2(P)$. Alors quitte à prendre S et les S_i disjoints, on peut construire l'application Φ de $P[Q_1, \ldots, Q_n]$ dans $\underline{2}^{S \cup S_1 \cup \ldots \cup S_n}$ définie par son codage réduit Φ_r tel que $\Phi_r(x) = \phi_r(x_i) \cup \overline{\phi i}_r(x)$ pour $x \in X_i$. On vérifie bien qu'on construit alors un plongement de $P[Q_1, \ldots, Q_n]$ de taille $dim_2(P) + \sum_{1 \leq i \leq n} dim_2(Q_i)$. ■

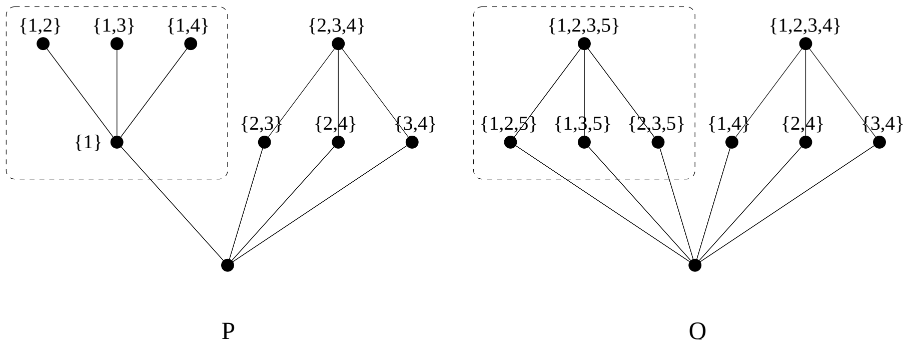


FIG. 5.7 – Deux ordres P et Q avec des graphes de comparabilité isomorphes, mais $dim_2(P) = 4$ et $dim_2(P) = 5$

Cette borne supérieure n'est pas très satisfaisante, et il est probablement possible de l'améliorer quitte à introduire d'autres paramètres.

Remarque 5.23. *Le produit lexicographique de deux ordres est un cas particulier de la substitution ($P \times_{lexico} Q$ est isomorphe à $P[Q, \ldots, Q]$)*

Les propositions 5.19 et 5.22 permettent de construire des codages bit-vecteur en utilisant la décomposition modulaire. Cette idée sera développée dans la section 5.9.

Pour finir, notons que la 2-dimension n'est pas un ***invariant de comparabilité*** (contrairement à la dimension des ordres). En effet, comme la figure 5.7 le montre, ils existent des ordres P et Q tels que $G_{comp}(P)$ est isomorphe à $G_{comp}(Q)$, mais $dim_2(P) \neq dim_2(Q)$.

5.4 Théorèmes d'équivalence

Cette section énonce des théorèmes qui font le lien entre plusieurs problèmes classiques d'optimisation combinatoire : plongement dans un hypercube, recouvrement d'un graphe biparti par des bipartis complets, coloriage de graphe, notion de base d'une famille d'ensembles. Ce sont des théorèmes dont les preuves constructives offrent plusieurs méthodes pour aborder le calcul de la 2-dimension d'un ordre.

Nous allons d'abord présenter ces problèmes auxquels se rattachent le calcul de la 2-dimension des ordres et des treillis (problème noté 2DIM dans le cas général des ordres).

Recouvrement d'un biparti par des bipartis complets [BIPCOVER] :
Donnée : un graphe biparti B.
Résultat : le nombre minimum $rec(B)$ de bipartis complets nécessaires pour recouvrir B. Un graphe biparti B est *recouvert* par une famille de bipartis complets $B_1, B_2, \ldots, B_k$ si et seulement si chacun des B_i est un sous-graphe induit de B et que toute arête de B appartient à au moins un des B_i.

Coloration de graphe [COLOR] :
Donnée : un graphe G.
Résultat : le nombre minimum $\chi(G)$ de couleurs nécessaires pour colorer les sommets de G tels que deux sommets adjacents aient toujours deux couleurs différentes.

Base d'une famille d'ensembles [SETBASIS] :
Données : un ensemble fini X et une famille $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ de n sous-ensembles de X.
Résultat : le nombre minimum $base(\mathcal{F})$ de sous-ensembles de X formant une base de $\mathcal{F}$. Une ***base*** (sous-entendu pour l'union) de $\mathcal{F}$ est une famille $\{B_1, \ldots, B_m\}$ de sous-ensembles de X tels que tout élément F_i de $\mathcal{F}$ est l'union de certains des B_j.

Les problèmes COLOR, BIPCOVER et SETBASIS sont associés à des problèmes de décision $\mathcal{NP}$-complets et respectivement classifiés dans [GJ79] par les sigles GT4, GT18 et SP7.

Le nombre minimum de bipartis complets recouvrant un biparti a été par exemple étudié par Fishburn et Hammer [FH93], par Müller [Mül96]. Ce problème a aussi été étudié par Wille [Wil82] et par Stahl et Wille [SW84] dans le cadre des treillis de concepts, où il est désigné comme la *s-dimension* du biparti. Le problème de la base minimum d'une famille d'ensembles, qui lui est lié, a par exemple été étudié par Lubiw [Lub90].

Bien entendu la liste de problèmes mentionnés dans les théorèmes suivants n'est pas exhaustive. Parmi les problèmes proches du calcul de la 2-dimension et repertoriés par Habib et Nourine, on pourra par exemple aussi citer le calcul de la 2-dimension intervallaire [FH93], le calcul du rang non-ambigu d'une matrice booléenne [Fro95] ou pour une réaction aux antigènes d'un leukocyte humain (donnée par une matrice), trouver le nombre minimum de spécificités qui expliquent cette réaction [NMWA78].

Théorème d'équivalence pour les treillis

Les définitions suivantes présentent les constructions qui sont utilisées pour se ramener aux différents problèmes.

On rappelle qu'on peut associer à un treillis T son graphe biparti $Bip(T) = (J(T), M(T), \not\leq_T)$. De plus, étant donné un treillis T, le ***graphe de conflit*** $G(T)$ est le graphe dont les sommets sont les couples jm où $j \in J(T)$, $m \in M(T)$ et $j \not\leq_P m$ et tel que jm et $j'm'$ sont adjacents si et seulement si $j \leq_T m'$ ou $j' \leq_T m$. Le théorème d'équivalence est le suivant (il est ici énoncé pour un treillis, mais quitte à reformuler les objets considérés, on pourra en donner une version s'appliquant directement à un ordre).

Théorème 5.24. *Etant donné un treillis $T = (X, \leq_T)$ et un entier k, les propositions suivantes sont équivalentes :*

(1) Il existe un plongement d'ordre de T dans l'hypercube $\underline{2}^k$.

(2) Il existe un plongement d'ordre de $(J(T) \cup M(T), \leq_T)$ dans l'hypercube $\underline{2}^k$.

(3) Il existe un recouvrement de $Bip(T) = (J(T), M(T), \not\leq_T)$ par k bi-partis complets.

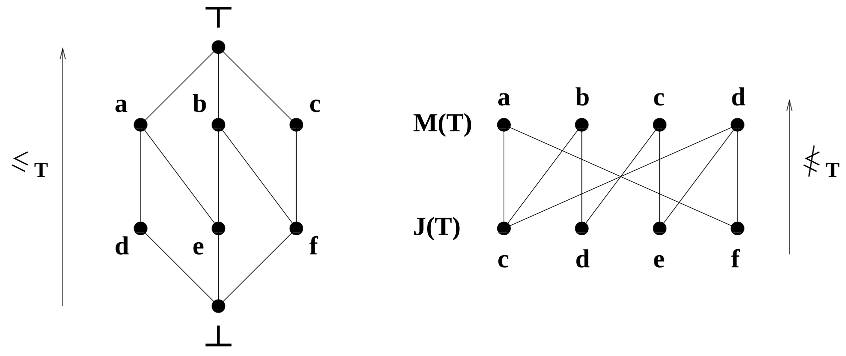
(4) Le graphe de conflit $G(T)$ peut être coloré avec k couleurs.

(5) Il existe une base de k sous-ensembles de $M(T)$ pour la famille $\mathcal{F}_T = \{Succ_{Bip(T)}(j) \mid j \in J(T)\}$.

Preuve. L'ordre dans lequel sont démontrées les implications est résumé par la figure 5.8. Un exemple est traité tout au long de la preuve pour illustrer les différentes constructions utilisées. Il s'agit du treillis T représenté ci-dessous, dont le biparti $Bip(T)$ est dessiné à coté. Les éléments de 2^k sont représentés par les parties de $\{1, 2, \ldots, k\}$ et un plongement dans 2^k consiste à associer à chaque sommet une partie de $\{1, 2, \ldots, k\}$ tel que l'ordre soit respecté.
(1) $\Rightarrow$ (2) : par restriction.

(2) $\Rightarrow$ (1) : La preuve la plus courte consiste à utiliser la propriété connue que T est isomorphe au complété de Dedekind-MacNeille $DM(J(T) \cup M(T))$ de l'ordre induit sur $J(T) \cup M(T)$ (voir [DP91] page 168). Par conséquent, si $(J(T) \cup M(T), \leq_T)$ se plonge dans $\underline{2}^k$, alors $DM(J(T) \cup M(T))$ qui est isomorphe à T se plonge aussi dans $\underline{2}^k$.

Nous présentons quand même un moyen de construire un plongement pour T à partir d'un plongement pour $J(T) \cup M(T)$. Soit φ un plongement de $(J(T) \cup M(T), \leq)$ dans 2^k, on étend φ à T par morphisme : soit $x \in T$, on sait que $x = \bigvee_{j \in J_x} j$, on choisit donc de poser $\phi(x) = \bigcup_{j \in J_x} \varphi(j)$. Il reste



Le treillis T **Son biparti Bip(T)**

à prouver que ϕ est un plongement, c'est à dire que pour tout $x, y \in T$, $x \leq y \iff \phi(x) \subseteq \phi(y)$.

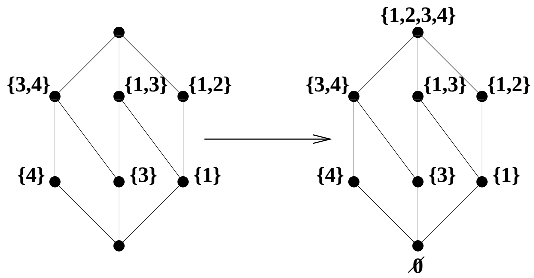
Le sens $\Rightarrow$ est clair. Pour la réciproque $\Leftarrow$, supposons que $\phi(x) \subseteq \phi(y)$.

Montrons d'abord que $\phi(x) \subseteq \bigcap_{m \in M_x} \varphi(m)$. En effet, on a l'implication suivante : $\forall j \in J(T), m \in M(T), j \leq x$ et $x \leq m \Longrightarrow \varphi(j) \subseteq \varphi(m)$, et donc $\bigcup_{j \in J_x} \varphi(j) \subseteq \bigcap_{m \in M_x} \varphi(m)$. Par conséquent, on a les équivalences suivantes qui permettent de conclure :

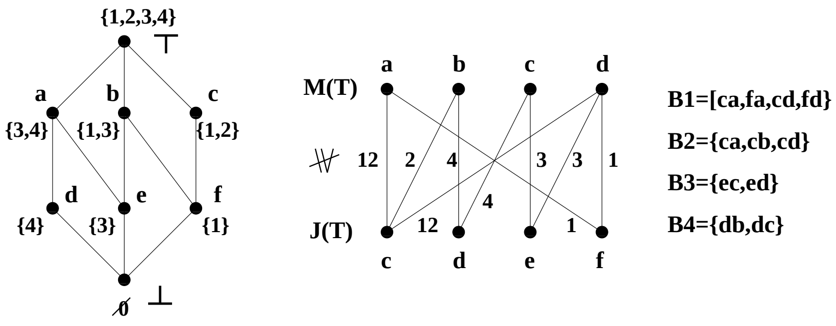
$$\begin{aligned}
\phi(x) \subseteq \phi(y) &\iff \bigcup_{j \in J_x} \varphi(j) \subseteq \bigcap_{m \in M_y} \varphi(m) \\
&\iff \forall j \in J(T),\ m \in M(T),\ j \leq x,\ y \leq m,\ \varphi(j) \subseteq \varphi(m) \\
&\iff \forall j \in J(T),\ m \in M(T),\ j \leq x,\ y \leq m,\ j \leq m \\
&\iff \bigvee_{j \in J_x} j \leq \bigwedge_{m \in M_y} m \\
&\iff x \leq y
\end{aligned}$$

$(2) \Rightarrow (3)$: étant donné ϕ un plongement de $J(T) \cup M(T)$ dans 2^k, prendre la famille des bipartis B_i, $1 \leq i \leq k$, où chaque B_i est composé des arêtes jm telles que $i \in \phi(j) \backslash \phi(m)$. On vérifie que chacun des B_i est un biparti complet et que les B_i recouvrent toutes les arêtes de $Bip(T)$.

$(3) \Rightarrow (4)$: supposons que $Bip(T)$ est recouvert par la famille de bipartis complets $B_1, \ldots, B_k$. Pour chaque sommet jm du graphe de conflit $G(T)$, choisir un des indices (n'importe lequel) parmi les indices i tels que jm appartient à B_i dans $Bip(T)$ et colorer le sommet jm avec cet indice. Par



Plongement de J(T)UM(T) **Plongement de T**

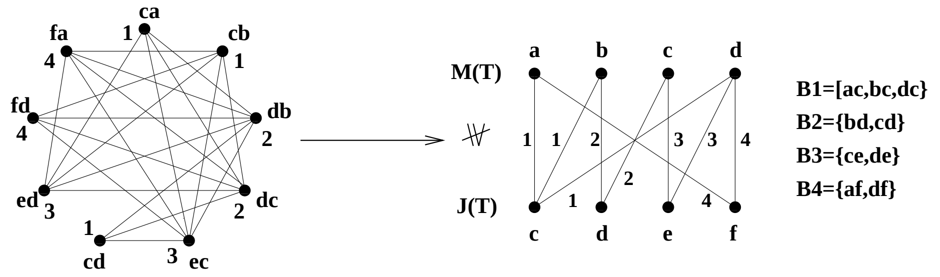


Treillis T **Biparti Bip(T)**

définition de la relation d'adjacence dans $G(T)$, on vérifie que cela constitue une coloration valide de $G(T)$ avec k couleurs.

(4) $\Rightarrow$ (3) : pour chaque couleur i, $1 \leq i \leq k$, considérer dans $Bip(T)$ l'ensemble des sommets qui sont les extremités des arêtes jm telles que jm a pour couleur i dans $G(T)$. Soit B_i le biparti induit par ces sommets dans $Bip(T)$. On vérifie que B_i est bien un biparti complet : si jm et $j'm'$ ont toutes les deux la couleur i dans $G(T)$, soit $j = j'$, soit $m = m'$, soit jm et $j'm'$ sont deux arêtes disjointes dans $Bip(T)$, mais dans ce cas comme elles portent la même couleur dans $G(T)$, on a $j \not\leq m'$ et $j' \not\leq m$ et par conséquent jm' et $j'm$ sont deux arêtes de $Bip(T)$.
(3) $\Rightarrow$ (5) : étant donné un recouvrement de $Bip(T)$ par les bipartis complets B_i, $1 \leq i \leq k$, considérer la famille $\{M_1, \ldots, M_k\}$ de k sous-ensembles de $M(T)$ telle que M_i est l'ensemble des extremités dans $M(T)$ des arêtes de B_i.

La famille $\{M_1, \ldots, M_k\}$ est une base pour l'union de la famille $\mathcal{F}_T = \{Succ_{Bip(T)}(j) \mid j \in J(T)\}$. Soit $j \in J(T)$, alors $Succ_{Bip(T)}(j)$ est l'union



Le graphe de conflit G(T) et une 4−coloration

Biparti Bip(T)

des M_i tels que B_i contient j parmi les extrémités de ses arêtes. En effet, $Succ_{Bip(T)}(j)$ est inclus dans cette union car pour tout $m \in Succ_{Bip(T)}(j)$, jm est une arête de $Bip(T)$ donc appartient à un B_i et donc $m \in M_i$. L'autre inclusion se montre en prouvant que si B_i contient j parmi les extremités de ces arêtes, alors $M_i \subseteq Succ_{Bip(T)}(j)$: soit $m \in M_i$, comme B_i est complet, jm est une arête de $Bip(T)$ et $m \in Succ_{Bip(T)}(j)$.

(5) $\Rightarrow$ (3) : Soit $M_1, \ldots, M_k$ une famille de k sous-ensembles de $M(T)$, qui forme une base de $\mathcal{F}_T = \{Succ_{Bip(T)}(j) \mid j \in J(T)\}$. Alors pour tout i, $1 \leq i \leq k$, le biparti B_i induit sur $Bip(T)$ par les éléments de M_i et les éléments $j \in J(T)$ tels que $M_i \subseteq Succ_{Bip(T)}(j)$ est un biparti complet. De plus, la famille des B_i, $1 \leq i \leq k$, recouvre $Bip(T)$.

(5) $\Rightarrow$ (2) : étant donnée une base $\{M_1, \ldots, M_k\}$ de $\mathcal{F}_T = \{Succ_{Bip(T)}(j) \mid j \in J(T)\}$, on va construire un plongement ϕ de $(J(T) \cup M(T), \leq)$ dans l'hypercube 2^k.

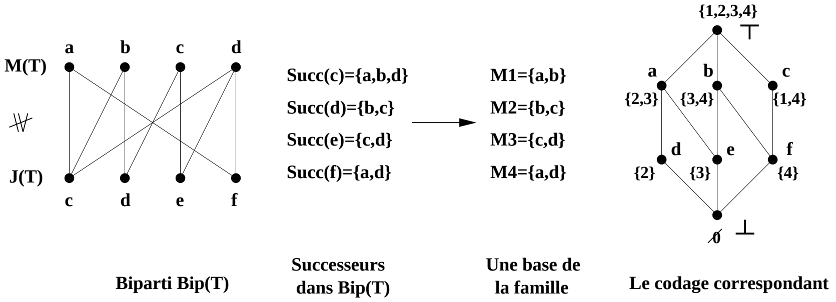
Dans ce qui suit, les variables $j, j', j'', \ldots$ désignerons toujours des éléments de $J(T)$ et les variables $m, m', m'', \ldots$ des éléments de $M(T)$. De plus, on omet l'indice $Bip(P)$ pour $Succ$ et $Pred$ qui seront utilisés juste pour ce biparti.

On construit d'abord un codage pour les sup-irréductibles. Soit $j \in J(T)$, on définit le code de j par :

$$\phi(j) = \{i \mid M_i \subseteq Succ(j)\}$$

A partir du codage des sup-irréductibles, on définit le codage des inf-irréductibles. Soit $m \in M(T)$, on définit le code de m par :

$$\phi(m) = \bigcup_{j \leq_T m} \phi(j)$$



Pour montrer que ce codage constitue d'abord un plongement de $(J(T),\leq)$ dans l'hypercube 2^k, il faut prouver que : $j \leq_T j' \iff \phi(j) \subseteq \phi(j')$. La preuve utilise les lemmes 1.1 et 1.2 qui montrent que $(J(T),\leq_T)$ est isomorphe à $(\{Succ(j)\ ,\ j \in J(L)\},\subseteq)$ avec $j \leq j' \iff Succ(j) \subseteq Succ(j')$. Or, comme la famille $\{M_1,\ldots,M_k\}$ est une base de $\mathcal{F}_T$, on a $Succ(j) \subseteq Succ(j')$ si et seulement si $\phi(j) \subseteq \phi(j')$. En effet, le sens $\Rightarrow$ est clair et pour le sens $\Leftarrow$, si $\phi(j) \subseteq \phi(j')$, alors $\forall i,\ 1 \leq i \leq k,\ M_i \subseteq Succ(j) \Rightarrow M_i \subseteq Succ(j')$ et on conclut grâce à l'égalité $Succ(j) = \bigcup_{M_i \subseteq Succ(j)} M_i$.

Il reste à montrer que l'application ϕ constitue bien un plongement de $(J(T) \cup M(T),\leq)$ dans 2^k. Il est clair que : $j \leq m \Longrightarrow \phi(j) \subseteq \phi(m)$. De même que : $m \leq m' \Longrightarrow code(m) \subseteq code(m')$. Les équivalences suivantes traitent le cas $m \leq j$:

$$\begin{aligned} m \leq_T j &\iff \forall j' \leq_T m,\ j' \leq_T j \\ &\iff \forall j' \leq_T m,\ \phi(j') \subseteq \phi(j) \\ &\iff \phi(m) \subseteq \phi(j) \end{aligned}$$

Il reste à vérifier que réciproquement si $\phi(j) \subseteq \phi(m)$ alors $j \leq_T m$. Supposons, par l'absurde, que $\phi(j) \subseteq \phi(m)$ mais $j \not\leq_T m$. Comme $m \in Succ(j)$, on peut supposer que $m \in M_1$ (car les M_i forment une base des $Succ(j),\ j \in J(T)$). Alors $1 \in \phi(j)$ et donc $1 \in \phi(m)$. Par définition de $\phi(m)$, $1 \in \phi(j')$ avec $j' \leq_T m$. Cela signifie que $M_1 \subseteq Succ(j')$ avec toujours $j' \leq_T m$. Or $m \in M_1$, donc $m \in Succ(j')$, c'est à dire $j' \not\leq_T m$ par définition de $Bip(T)$. Contradiction avec $j' \leq_T m$.

Ceci permet en plus de traiter la dernière implication à vérifier :

$$\begin{aligned} \phi(m) \subseteq \phi(m') &\Longrightarrow \forall j \leq m,\ \phi(j) \subseteq \phi(m') \\ &\Longrightarrow \forall j \leq m,\ j \leq m' \\ &\Longrightarrow m \leq m' \end{aligned}$$

■

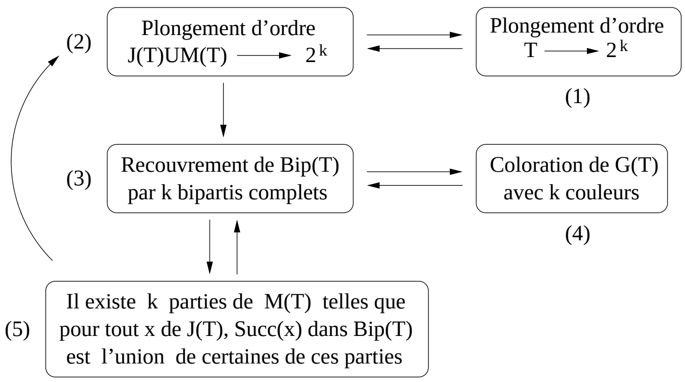


FIG. 5.8 – Ordre de démonstration des implications

Corollaire 5.25. *Soit T un treillis. Alors*

$$dim_2(T) = rec(Bip(T)) = \chi(G(P)) = base(\mathcal{F}_T)$$

Remarque 5.26. *On aurait pu remplacer la condition (5) par la variante (5') suivante. Au lieu de considérer $\mathcal{F}_T = \{F_1, \ldots, F_n\}$, on peut introduire la fermeture par union de $\mathcal{F}_T$ définie par $\mathcal{UF}_T = \{\bigcup_{i \in I} F_i \mid I \subseteq [1, n]\}$. La condition (5') s'exprime alors par : il existe une base de k sous-ensembles de $M(T)$ pour la famille $\mathcal{UF}_T$. Il est clair que $(5) \Rightarrow (5')$. De plus on peut vérifier que $\mathcal{UF}_T$ munie de l'inclusion est un ordre isomorphe à $AM(Bip(T))$ lui-même isomorphe à T (voir le théorème 1.3). Par conséquent, on a une preuve rapide que $(5') \Rightarrow (1)$. On a conservé l'implication $(5) \Rightarrow (2)$ car elle fournit par exemple avec $(4) \Rightarrow (5) \Rightarrow (2)$ une construction polynomiale d'un codage bit-vecteur de $(J(T) \cup M(T), \leq_T)$ à partir d'un recouvrement de $Bip(T)$.*

Quelques liens entre ordres, treillis et bipartis

Nous rappelons ici un processus de réduction des bipartis et son lien avec le treillis des antichaines maximales d'un ordre.

Définition 5.27. *Soit $B = (X, Y, <_B)$ un ordre biparti. on dit que B est* ***réduit*** *si les conditions suivantes sont vérifées :*

- *Pour tout $x \in X$, $Succ(x) \neq \bigcup_{x' \neq x,\ Succ(x') \subseteq Succ(x)} Succ(x')$.*
- *Pour tout $y \in Y$, $Pred(y) \neq \bigcup_{y' \neq y,\ Pred(y') \subseteq Pred(y)} Pred(y')$.*

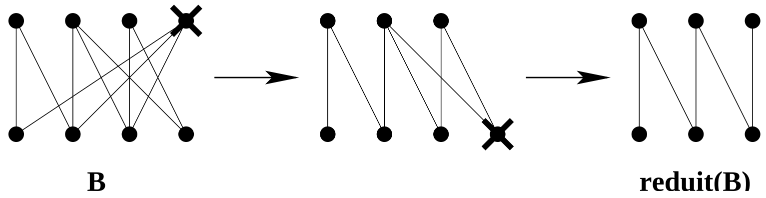


FIG. 5.9 – Biparti B et son biparti réduit $reduit(B)$

Tout biparti B peut être réduit en supprimant au fur et à mesure les éléments x et y ne vérifiant pas les conditions 1 et 2. Quelle que soit la séquence de suppression, on finit par obtenir un biparti réduit unique (à un isomorphisme près), noté $Reduit(B)$. La figure 5.9 représente une telle séquence de suppressions.

Théorème 5.28 (Morvan, Nourine [Nou93, MN93, MN96]). *Soit T un treillis. Alors $Bip(T)$ est un biparti réduit et T est isomorphe à $AM(Bip(T))$. De plus, soit B un biparti et $T = AM(B)$, alors $Reduit(B)$ est isomorphe à $Bip(T)$.*

Concernant le comportement de cette réduction par rapport à BIPCOVER, on a la proposition suivante.

Proposition 5.29. *Soit B un biparti. Alors $rec(Reduit(B)) = rec(B)$.*

Preuve. $Reduit(B)$ correspond à un sous-graphe induit de B et il est clair que tout recouvrement de B avec k bipartis complets induit sur $Reduit(B)$ un recouvrement à k bipartis complets.

Dans l'autre sens, B s'obtient à partir de $Reduit(B)$ en ajoutant successivement des sommets ne vérifiant pas la condition 1 et 2.

Montrons que s'il existe un recouvrement d'un biparti $B = (X, Y, <_B)$ avec k bipartis complets $B_1, \ldots, B_k$ et si on ajoute à X un nouveau sommet x tel que $Succ(x) = \bigcup_{x' \in A} Succ(x')$ avec $A \subseteq X$, alors il existe un recouvrement du nouveau biparti avec toujours k biparti complets. En effet, pour chaque $y \in Succ(x)$, il existe $x' \in A$ tel que $x'y$ est une arête de B et donc il existe un B_i contenant cette arête et il suffit alors d'ajouter x à B_i (ce qui donne un nouveau biparti complet). De manière duale, on traite le cas où on ajoute un sommet à Y dont les prédecesseurs sont l'union des prédecesseurs d'autres sommets.

■

Théorème d'équivalence pour les ordres

Nous allons maintenant présenter une version de ce théorème pour le cas général des ordres. Son énoncé et sa preuve sont similaires au théorème d'équivalence pour les treillis. Pour un ordre $P = (X, \leq_P)$, on rappelle qu'on peut définir le biparti $\overline{B}(P) = (X, X, \not\leq_P)$. On définit un graphe de conflit $G(P)$ dont les sommets sont les couples xy où $x, y \in X$ avec $x \not\leq_P y$, et tel que deux sommets xy et st sont adjacents si et seulement si $x \leq_P t$ ou $s \leq_P y$.

Théorème 5.30. *Etant donné un ordre $P = (X, \leq_P)$ et un entier k, les propositions suivantes sont équivalentes :*

(1) Il existe un plongement d'ordre de P dans l'hypercube $\underline{2}^k$.

(2) Il existe un recouvrement de $\overline{B}(P) = (X, X, \not\leq_P)$ par k bipartis complets.

(3) Le graphe de conflit $G(P)$ peut être coloré avec k couleurs.

(4) Il existe une base de k sous-ensembles de X pour la famille $\mathcal{F}_P = \{Succ_{\overline{B}(P)}(x) \mid x \in X\}$.

Preuve. Nous donnons à nouveau des preuves constructives. Les constructions sont du même type que celles employées pour le théorème d'équivalence sur les treillis, et les preuves sont pratiquement identiques. Nous redonnons ces constructions avec les arguments remplaçant ceux des treillis quand c'est nécessaire.

(1) $\Rightarrow$ (2) : Soit ϕ un plongement d'ordre de P dans $\underline{2}^k$, considérer les bipartis B_i constitués des arêtes xy de $\overline{B}(P)$ telles que $x \not\leq_P y$ avec $i \in \phi(x) \backslash \phi(y)$, pour $1 \leq i \leq k$. Ce sont des bipartis complets recouvrant $\overline{B}(P)$.

(2) $\Rightarrow$ (3) : Soit $B_1, \ldots, B_k$ k bipartis complets recouvrant $\overline{B}(P)$, colorer chaque sommet xy de $G(P)$ avec un i quelconque tel que xy est une arête de B_i.

(3) $\Rightarrow$ (2) : Soit une coloration de $G(P)$ avec k couleurs, pour chaque i, $1 \leq i \leq k$, considérer dans $\overline{B}(P)$ l'ensemble des sommets qui sont les extremités des arêtes xy telles que xy a pour couleur i dans $G(P)$. Le biparti B_i induit par ces sommets dans $\overline{B}(P)$ est un biparti complet et la famille des B_i recouvre $\overline{B}(P)$.

(2) $\Rightarrow$ (4) : Etant donné un recouvrement de $\overline{B}(P)$ par les bipartis complets B_i, $1 \leq i \leq k$, considérer la famille $\{M_1, \ldots, M_k\}$ de k sous-ensembles de X telle que M_i est l'ensemble des extremités dans $Max(\overline{B}(P))$ des arêtes de B_i. C'est une base de $\mathcal{F}_P$.

(4) $\Rightarrow$ (1) : Etant donnée une base $\{M_1, \ldots, M_k\}$ de $\mathcal{F}_P$, on définit pour

chaque $x \in X$ son codage par $\phi(x) = \{i \mid M_i \subseteq Succ_{\overline{B}(P)}(x)\}$. La démonstration qu'il s'agit bien d'un plongement de P dans $\underline{2}^k$ utilise les mêmes arguments que celle pour les treillis, sauf qu'il faut utiliser l'équivalence $x \leq_P x' \Leftrightarrow Succ_{\overline{B}(P)}(x) \subseteq Succ_{\overline{B}(P)}(x')$ et que l'on a seulement un cas à traiter (cas correspondant aux sup-irréductibles seuls).

■

Corollaire 5.31. *Soit P un ordre. Alors*

$$dim_2(P) = rec(\overline{B}(P)) = \chi(G(P)) = base(\mathcal{F}_P)$$

Preuve. Il suffit d'appliquer le théorème d'équivalence pour les ordres. Mais on pourrait aussi montrer le première égalité en passant par les treillis avec le théorème 5.24 d'équivalence sur les treillis, le théorème 5.28, les propositions 5.10 et 5.29 et un théoréme présenté dans [Nou93] qui dit que $MN(P)$ est isomorphe à $AM(\overline{B}(P))$:

$$\begin{aligned} dim_2(P) &= dim_2(MN(P)) \\ &= dim_2(AM(\overline{B}(P)) \\ &= rec(Bip(AM(\overline{B}(P)))) \\ &= rec(Reduit(\overline{B}(P))) \\ &= rec(\overline{B}(P)) \end{aligned}$$

■

Pour ce qui est de la réduction vers le problème de coloration de graphe, Skorsky a montré que l'on peut se restreindre à colorer un sous-graphe du graphe de conflit pour obtenir la 2-dimension de l'ordre initial. Ce sous-graphe ne comprend que les sommets xy tels que x et y sont en relation flèche. Dans un ordre $P = (X, \leq_P)$, deux éléments x et y sont en ***relation flèche***, noté $x \swarrow\!\!\nearrow y$, si $x \not\leq_P y$ et $\forall a \in X$, $a <_P x \implies a \leq_P y$ et $a >_P y \implies a \geq_P x$.

Cette relation très utile pour décrire certaines propriétés des treillis a été introduite par Wille [Wil83]. L'ensemble des couples xy tels que $x, y \in X$ et $x \swarrow\!\!\nearrow y$ est noté $Cr(P)$ (l'ensemble des paires critiques $Crit(P)$ défini dans [RR79] est un sous-ensemble de $Cr(P)$). Pour un ordre $P = (X, \leq_P)$, le ***graphe de conflit critique*** $G_{cr}(P)$ est le graphe dont les sommets sont les couples xy avec $x, y \in X$ et $x \swarrow\!\!\nearrow y$, et tel que xy et st sont adjacents si et seulement si $x \leq_P t$ ou $s \leq_P y$. C'est clairement un sous-graphe de $G(P)$ que P soit un ordre ou un treillis. Noter que dans un treillis T, si $x \swarrow\!\!\nearrow y$, cela implique que $x \in J(T)$ et $y \in M(T)$ (mais la réciproque n'est pas vraie).

Théorème 5.32 (Skorsky [Sko92]). *Soit* $P = (X, \leq_P)$ *un ordre et* $G_{cr}(P)$ *son graphe de conflit critique. Alors* $dim_2(P) = \chi(G_{cr}(P))$.

Preuve. Nous ne donnons pas la preuve complète ici, mais du point de vue des constructions, il suffit de savoir que dans les théorèmes d'équivalence, on a vu comment colorer $G(P)$ à partir d'un plongement de P dans $\underline{2}^k$ (implication (1) $\Rightarrow$ (2) $\Rightarrow$ (3) du théorème 5.30) et donc comment colorer $G_{cr}(P)$ qui est un sous-graphe de $G(P)$.

Nous présentons maintenant la construction réciproque qui permet d'obtenir un plongement d'ordre de P dans $\underline{2}^k$ à partir d'une coloration χ de $G_{crit}(P)$ avec k couleurs. En notant $\chi(st)$ la couleur du couple st, on définit l'application ϕ de X dans $\underline{2}^k$ par :

$$\phi(x) = \bigcup_{st \in Cr(P),\ s \leq_P x} \chi(st) \text{ pour } x \in X$$

La preuve qu'il s'agit bien d'un plongement d'ordre utilise la propriété que pour tout $x, y \in X$ tels que $x \not\leq_P y$, il existe un couple $st \in Cr(P)$ tel que $s \leq_P x$ et $y \leq_P t$ (raisonner par l'absurde pour montrer ensuite que $\phi(x) \subseteq \phi(y)$ implique $x \leq_P y$). ■

5.5 $\mathcal{NP}$-complétude et non-approximabilité

Cette section positionne le problème du calcul de la 2-dimension du point de vue de sa complexité théorique. Pour les définitions employées pour décrire les problèmes $\mathcal{NP}$-complets, les problèmes d'approximation et les transformations pour passer d'un problème à un autre (aussi appelées réductions), le lecteur est renvoyé au Chapitre 1.

$\mathcal{NP}$-complétude de la 2-dimension des ordres

Depuis [SW84], on sait que le calcul de la 2-dimension des ordres est associé à un problème de décision $\mathcal{NP}$-complet. Nous allons quand même redonner ici une démonstration de cette $\mathcal{NP}$-complétude en donnant une réduction à partir du problème de couverture des arêtes d'un graphe biparti par des bipartis complets (BIPCOVER). Cette réduction est présente dans un rapport de recherche de Habib et Nourine, elle fait appel au théorème d'équivalence 5.24 et à d'autres résultats sur les treillis, on en trouvera une légère variante dans [RT00]. On pourra vérifier qu'en fait il s'agit d'une réduction continue et que $BIPCOVER \xrightarrow{1} 2DIM$.

Théorème 5.33. *Il existe une réduction continue de BIPCOVER vers 2DIM telle que BIPCOVER* $\xrightarrow{1}$ *2DIM. Par conséquent, le problème de décision associé à 2DIM est $\mathcal{NP}$-complet.*

Preuve. Soit B un biparti. La réduction est la suivante.
▷ Calculer le biparti réduit $Reduit(B)$ (c'est polynomial en temps), on le note $B' = (J, M, <)$.

On se place maintenant dans le contexte du théorème d'équivalence 5.24 en considérant le treillis $T = AM(B')$. Comme B' est réduit et d'après le théorème 5.28, B' est isomorphe à $Bip(T)$. Par conséquent on peut assimiler J à $J(T)$, M à $M(T)$ et $<$ à $\not\leq_T$.
▷ Avec le théorème d'équivalence 5.24 et la proposition 5.29, on a $dim_2(J(T) \cup M(T)) = rec(B') = rec(B)$ et à partir d'un codage bit-vecteur de $(J(T) \cup M(T), \leq_T)$ de taille k, on sait calculer un recouvrement de B' puis de B avec k bipartis complets, en temps polynomial.

Il reste donc à reconstruire l'ordre $(J(T) \cup M(T), \leq_T)$ à partir de B'.
▷ A partir de $B' = (J, M, <)$, on construit le biparti $B_1 = (J \cup M, M, <_1)$ avec $j <_1 m$ si et seulement si $j < m$ et $m <_1 m'$ si et seulement si $Pred_{B'}(m) \not\supseteq Pred_{B'}(m')$. Puis on construit l'ordre P sur $J \cup M$ en ordonnant les éléments par inclusion des successeurs. Par conséquent, si plusieurs éléments ont les mêmes successeurs dans $(J \cup M, M, <_1)$, on les contracte en un unique élément. L'ordre P est isomorphe à $(J(T) \cup M(T), \leq_T)$.

Pour prouver ce dernier point, il suffit de remarquer que pour $j \in J$, $Succ_{B_1}(j) = M \backslash M_T(j)$. On sait que $m \leq_T m'$ si et seulement si $Pred_{B'}(m) \supseteq Pred_{B'}(m')$ (comme B' est isomorphe à $Bip(T)$, et avec le lemme 1.2 pour les sup-irréductibles). Donc pour $m \in M$, $Succ_{B_1}(m) = M \backslash M_T(m)$. On conclut grâce au lemme 1.2 pour les inf-irréductibles qui dit que, pour tout x et y éléments de T (et donc a fortiori de $J \cup M$), $x \leq_T y$ si et seulement si $M_T(x) \supseteq M_T(y)$, c'est à dire pour $x, y \in J \cup M$, si et seulement si $Succ_{B_1}(x) \subseteq Succ_{B_1}(y)$.

La construction de P est aussi polynomiale en temps et $|P| \leq |B|$.

Maintenant que les propriétés de la réduction sont prouvées, comme le problème de décision associé à BIPCOVER est $\mathcal{NP}$-complet d'après un résultat de Orlin [Orl76], alors 2DIM est aussi associé à un problème de décision $\mathcal{NP}$-complet. ■

Remarque 5.34. *On peut effectuer la même construction à partir du biparti initial et non pas du biparti réduit (c'est à dire directement à partir de l'étape 3). Cette transformation fonctionne aussi, mais la preuve qu'il faut* *développer est alors très formelle (laissée au lecteur), on sait moins bien quels objets intermédiaires on manipule, contrairement à la transformation avec le réduit du biparti où on utilise des objets et des propositions d'usage courant en théorie de l'ordre.*

Non-approximabilité de la 2-dimension des ordres

Simon a montré dans [Sim90] que $COLOR \xrightarrow{1+} BIPCOVER$ et plus précisément, en analysant sa réduction, que si BIPCOVER est approximable à $r(n)$ près où n est le nombre de sommets du biparti, alors COLOR est approximable à $2r(n^3)$ près où n est le nombre de sommets du graphe. La coloration des graphes a fait l'objet de son coté de plusieurs études et les théorèmes suivants présentent deux des derniers résultats publiés à ce jour.

Théorème 5.35 (Bellare,Goldreich,Sudan [BGS98]). *Pour tout $\epsilon > 0$, COLOR n'est pas approximable à $n^{1/7-\epsilon}$ près, où n est le nombre de sommets du graphe, à moins que $\mathcal{P} = \mathcal{NP}$.*

Théorème 5.36 (Feige,Kilian [FK98]). *Pour tout $\epsilon > 0$, COLOR n'est pas approximable à $n^{1-\epsilon}$ près, où n est le nombre de sommets du graphe, à moins que $\mathcal{ZPP} = \mathcal{NP}$.*

A l'aide du théorème 5.33 et du résultat de Simon, on en déduit le théorème de non-approximabilité suivant.

Théorème 5.37. *Pour tout $\epsilon > 0$, soit n le nombre de sommets de l'ordre, 2DIM n'est pas approximable à $n^{1/21-\epsilon}$ près à moins que $\mathcal{P} = \mathcal{NP}$ et n'est pas approximable à $n^{1/3-\epsilon}$ près à moins que $\mathcal{ZPP} = \mathcal{NP}$.*

Par conséquent, le calcul de la 2-dimension des ordres est bien un problèmes très difficile dans le cas général. Un corollaire de ce théorème est qu'on ne peut pas l'approximer à un facteur constant près, à moins bien sûr que $\mathcal{P} = \mathcal{NP}$. Il est vraisemblable qu'on puisse élargir les bornes de non-approximabilité de 2DIM à celles de COLOR.

Question 2. *Est-ce qu'il existe une réduction plus "économique" de COLOR vers BIPCOVER qui permettrait de démontrer que si BIPCOVER est approximable à $r(n)$ près où n est le nombre de sommets du biparti, alors COLOR est approximable à $r(n)$ près où n est le nombre de sommets du graphe ? Dans ce cas, BIPCOVER et 2DIM hériteraient des mêmes bornes de non-approximabilité que COLOR.*

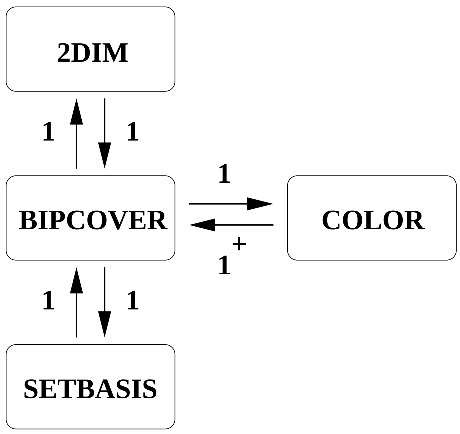


FIG. 5.10 – Réductions continues entre 2DIM, BIPCOVER, COLOR et SETBASIS

Liens entre les différents problèmes : 2DIM, BIPCOVER, SETBASIS, COLOR.

On remarque finalement que le calcul de la 2-dimension peut se réinterpréter simplement en plusieurs autres problèmes combinatoires. Ce qui n'est pas le cas d'autres plongements dans des produits de chaines comme la dimension des ordres dont on sait peu de choses d'un point de vue algorithmique et complexité (la démonstration de $\mathcal{NP}$-complétude étant par exemple très difficile [Yan82], voir [Spi94] pour une liste de problèmes encore ouverts sur la dimension).

La figure 5.10 résume différentes réductions qui existent entre les problèmes d'optimisation 2DIM, BIPCOVER, COLOR et SETBASIS. Les flèches correspondent à des réductions continues avec le facteur indiqué. Nous donnons maintenant certaines de ces réductions en précisant à chaque fois la taille des instances construites à partir des instances initiales.

$2DIM \xrightarrow{1} BIPCOVER$: A partir d'un ordre $P = (X, \leq_P)$, construire le biparti $\overline{B}(P) = (X, X, \leq_P)$. Alors il existe un plongement de P dans l'hypercube $\underline{2}^k$ si et seulement si il existe un recouvrement de $\overline{B}(P)$ avec k bipartis complets. La preuve figure dans le théorème d'équivalence 5.30. Toutes ces constructions sont polynomiales, le facteur de la réduction continue est 1 et $|\overline{B}(P)| = 2|P|$ (si on veut réduire un peu plus la taille du biparti construit, on peut rajouter une étape dans laquelle on calcule le réduit $Reduit(\overline{B}(P))$).

$BIPCOVER \xrightarrow{1} 2DIM$: Prendre la réduction du théorème 5.33.

$BIPCOVER \xrightarrow{1} COLOR$ **et** $COLOR \xrightarrow{1^+} BIPCOVER$: Ce sont les réductions présentées par Simon dans [Sim90]. A partir d'un biparti B d'arêtes E, il construit le graphe G dont l'ensemble des sommets est E et tel que $e, e' \in E$ sont adjacents si et seulement si les extremités de e et e' ne forment pas un biparti complet de B. La preuve qu'il existe k bipartis complets recouvrant B si et seulement si il existe un coloration de G avec k couleurs, est similaire à celle donnée pour (3) $\Leftrightarrow$ (4) dans le théorème d'équivalence 5.24.

Ensuite, à partir d'un graphe $G = (V, E)$ et pour un rationnel $\alpha > 1$, il donne la construction d'un biparti B avec $|B| \leq 4/(\alpha - 1)|V|^3$ et tel que pour une coloration de G à ϵ près de l'optimum, on peut construire un recouvrement de B à $\alpha\epsilon$ près de l'optimum.

$BIPCOVER \xrightarrow{1} SETBASIS$: Soit $B = (X, Y, <)$ un biparti, considérer la famille $\mathcal{F}$ sur Y définie par $\mathcal{F} = \{Succ(x) \mid x \in X\}$. B admet un recouvrement avec k bipartis complets si et seulement si $\mathcal{F}$ admet une base avec k ensembles. La preuve et la construction pour passer d'une solution de SETBASIS à une solution de BIPCOVER sont données dans l'équivalence (3) $\Leftrightarrow$ (5) du théorème 5.24 (où on n'utilise aucune propriété d'ordre ou de treillis).

$SETBASIS \xrightarrow{1} BIPCOVER$: Soit $\mathcal{F}$ une famille de sous-ensembles de X, considérer le biparti $B = (\mathcal{F}, X, <)$ où pour $F \in \mathcal{F}$, $Succ(F) = F \subseteq X$. B admet un recouvrement avec k bipartis complets si et seulement si $\mathcal{F}$ admet une base avec k ensembles. La preuve et la construction pour passer d'une solution de BIPCOVER à une solution de SETBASIS sont données dans l'équivalence (3) $\Leftrightarrow$ (5) du théorème 5.24.

Remarquer que les réductions de $2DIM \xrightarrow{1} COLOR$ et $2DIM \xrightarrow{1} SETBASIS$ présentes dans le théorème 5.30 correspondent bien à $2DIM \xrightarrow{1} BIPCOVER \xrightarrow{1} COLOR$ et $2DIM \xrightarrow{1} BIPCOVER \xrightarrow{1} SETBASIS$.

Remarque 5.38. *La réduction directe de SETBASIS vers 2DIM qui consiste, à partir d'une famille d'ensembles, à considérer l'ordre d'inclusion sur ces ensembles ne fonctionne pas.*

Prendre par exemple la famille $\mathcal{F} = \{\{1\}, \{2\}, \{1, 2, 3\}\}$ *sur* $X = \{1, 2, 3\}$. *L'ordre d'inclusion sur ces ensembles a une 2-dimension qui vaut 2 alors qu'une base de* $\mathcal{F}$ *contient nécessairement 3 ensembles. Cette réduction ne* *fonctionne que si la famille d'ensembles est fermée par union (voir la remarque 5.26). Cependant la méthode qui consisterait alors à fermer par union une famille d'ensembles pour ensuite considérer son ordre d'inclusion fonctionne mais n'est pas polynomiale car la fermeture peut avoir une taille exponentielle par rapport à la famille initiale.*

5.6 Quelques cas particuliers

Les sections suivantes énumèrent quelques cas particuliers pour lesquels on sait dire quelque chose de la 2-dimension.

5.6.1 Deux ou trois chaines

Proposition 5.39. *Soient trois chaines $\mathcal{C}_{n_1}$, $\mathcal{C}_{n_2}$ et $\mathcal{C}_{n_3}$ avec respectivement n_1, n_2 et n_3 éléments, tel que $n_1 \geq n_2 \geq n_3$. Alors :*

1. $dim_2(\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2}) = \max(dim_2(\mathcal{C}_{n_1}), dim_2(\mathcal{C}_{n_2})) + 2 = n_1 + 1$.
2. $dim_2(\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}) = \max(n_1 + 1, n_2 + 2)$.

Preuve. Soit $\mathcal{C}_{n_1}$ et $\mathcal{C}_{n_2}$ deux chaines avec respectivement n_1 et n_2 éléments, $n_1 \geq n_2$. On note $\perp_1$ le minimum de $\mathcal{C}_{n_1}$ et $\perp_2$ le minimum de $\mathcal{C}_{n_2}$. Soit ϕ un plongement de $\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2}$ dans $\underline{2}^S$. Alors nécessairement $\exists a \in S$ tel que $a \in \phi(\perp_1) \backslash \phi(\perp_2)$, sinon on aurait $\phi(\perp_1) \subseteq \phi(\perp_2)$ et donc $\perp_1 \leq \perp_2$ dans l'ordre initial, ce qui n'est pas le cas. De même $\exists b \in \phi(\perp_2) \backslash \phi(\perp_1)$, dont on déduit que pour tout élément x de $\mathcal{C}_{n_1}$, $b \notin \phi(x)$. L'application ϕ' de $\mathcal{C}_{n_1}$ dans $\underline{2}^{S \backslash \{a,b\}}$ définie par $\phi'(x) = \phi(x) \backslash \{a\}$ est clairement un plongement, de taille $|S \backslash \{a, b\}| \geq dim_2(\mathcal{C}_{n_1}) = n_1 - 1$. Donc $|S| \geq n_1 + 1$.

Maintenant soit ϕ_1 un codage optimal de $\mathcal{C}_{n_1}$ dans $\underline{2}^{S_1}$, $|S_1| = n_1 - 1$ et ϕ_2 un codage optimal de $\mathcal{C}_{n_2}$ dans $\underline{2}^{S_2}$, $|S_2| = n_2 - 1$, qui s'étend naturellement en un plongement dans $\underline{2}^{S_1}$ car $n_1 \geq n_2$. On introduit alors deux nouvelles couleurs a et b permettant de définir un plongement ϕ de $\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2}$ dans $\underline{2}^{S \cup \{a,b\}}$ par $\phi(x) = \phi_1(x) \cup \{a\}$ si x est dans $\mathcal{C}_{n_1}$ et $\phi(x) = \phi_2(x) \cup \{b\}$ si x est dans $\mathcal{C}_{n_2}$. La taille de ce plongement vaut $n_1 - 1 + 2 = n_1 + 1$. La première égalité de la proposition est prouvée.

Si on introduit en parallèle une troisième chaine $\mathcal{C}_{n_3}$ à n_3 éléments, avec $n_1 \geq n_2 \geq n_3$. Supposons que $n_1 + 1 \geq n_2 + 2$. Comme $\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2}$ est un sous-ordre de $\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}$, on a $dim_2(\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}) \geq dim_2(\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2}) = n_1 + 1$. De même, on sait qu'il existe un codage de $\mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}$ avec $n_2 + 1$ couleurs et qu'il existe un codage de $\mathcal{C}_{n_1}$ avec $n_1 - 1$ couleurs. On code donc la chaine $\mathcal{C}_{n_1}$ avec $n_1 - 1$ couleurs, on code $\mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}$ en utilisant les mêmes couleurs, c'est faisable car $n_2 + 1 \leq n_1 - 1$. On introduit deux nouvelles couleurs a et b, et on ajoute a à tous les codes de $\mathcal{C}_{n_1}$ et b à tous les codes de $\mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}$. On obtient ainsi un codage à $n_1 + 1$ couleurs de $\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}$.

Supposons que $n_2+2 \geq n_1+1$, ce qui signifie nécessairement que $n_2 = n_1$. On note $\perp_1$, $\perp_2$ et $\perp_3$ les minimum respectifs de $\mathcal{C}_{n_1}$, $\mathcal{C}_{n_2}$ et $\mathcal{C}_{n_3}$. Supposons qu'il existe un plongement ϕ de $\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}$ dans $\underline{2}^S$ avec $|S| \leq n_1 + 1$. Alors on sait que nécessairement $\exists a \in S$ tel que $a \in \phi(\perp_1) \backslash \phi(\perp_2)$ et $\exists b \in S$ tel que $a \in \phi(\perp_2) \backslash \phi(\perp_1)$. De plus ϕ_1 de $\mathcal{C}_{n_1}$ dans $\underline{2}^{S \backslash \{a,b\}}$ définie par $\phi_1(x) = \phi(x) \backslash \phi(\perp_1)$ est un plongement. Le nombre de couleurs utilisées par ϕ_1 vaut $|S| - |\phi(\perp_1)| - 1$ et est nécessairement $\geq n_1 - 1$. Comme $|S| \leq n_1 + 1$, ceci implique que $|\phi(\perp_1)| \leq 1$ et donc $\phi(\perp_1) = \{a\}$ et aussi $\phi(\top_1) = S \backslash \{b\}$ où $\top_1$ désigne le maximum de $\mathcal{C}_{n_1}$. De même, on montre que $\phi(\perp_2) = \{b\}$ et $\phi(\top_2) = S \backslash \{a\}$ où $\top_2$ désigne le maximum de $\mathcal{C}_{n_2}$. Maintenant si on regarde $\phi(\perp_3)$, si $\phi(\perp_3)$ ne contient pas à la fois a et b, il est inclus dans $\phi(\top_1)$ ou $\phi(\top_2)$, ce qui est impossible. Sinon $\phi(\perp_3)$ contient à la fois a et b, mais dans ce cas il contient $\phi(\perp_1)$ et $\phi(\perp_2)$, ce qui est aussi impossible. Donc l'hypothèse initiale était fausse, on a un plongement avec $|S| \geq n_1+2$. Il existe un codage de $\mathcal{C}_{n_1}$ avec $n_1 - 1$ couleurs, on réutilise ces couleurs pour coder $\mathcal{C}_{n_2}$ et pour coder $\mathcal{C}_{n_3}$ (ce qui ne nécessite que $n_3 - 1$ couleurs, or $n_3 - 1 \leq n_1 - 1$). Puis on introduit trois nouvelles couleurs a, b et c. On ajoute a à tous les codes de $\mathcal{C}_{n_1}$, b à tous les codes de $\mathcal{C}_{n_2}$ et c à tous les codes de $\mathcal{C}_{n_3}$. On obtient ainsi un codage de $\mathcal{C}_{n_1} \uplus \mathcal{C}_{n_2} \uplus \mathcal{C}_{n_3}$ à $n_1 + 2 = n_2 + 2$ couleurs. ■

Si on passe à quatre chaines en parallèle, on peut à nouveau essayer de construire une formule pour la 2-dimension à partir d'une étude de cas. Cependant en multipliant le nombre de chaines en parallèle, on multiplie rapidement le nombre de cas à considérer, ce qui devient très vite inextricable. Nous ne connaissons pour l'instant ni formule, ni algorithme polynomial donnant la 2-dimension d'un nombre quelconque de chaines en parallèle (on reviendra sur cette question dans l'étude des arbres en section 5.8).

5.6.2 Ordres MQ-critiques

Dans [Tro75], Trotter a étudié le cas des ordres P vérifiant l'égalité $dim_2(P) = |P|$. De tels ordres sont appelés ***MQ-critiques*** pour reprendre sa terminologie. Il est parvenu à complètement caractériser ces ordres.

Théorème 5.40 (Trotter [Tro75]). *Soit P un ordre, cet ordre peut s'écrire $P = P_1 \oplus \ldots \oplus P_k$ où $P_1, \ldots, P_k$ sont k ordres premiers pour l'opération série. Alors P est MQ-critique si et seulement si les ordres P_i, $1 \leq i \leq k$, sont tous des ordres MQ-critiques premiers pour l'opération série tels que $|P_i| \geq 2$. La liste des ordres Q qui sont MQ-critiques et premiers pour l'opération série est donnée ci-dessous :*

1. *si $|Q| = 2$, Q est $\mathcal{A}_2$.*
2. *si $|Q| = 3$, Q est soit $\mathcal{A}_3$, soit $\mathcal{C}_2 \uplus \mathcal{C}_1$.*
3. *si $|Q| = 4$, Q est soit $\mathcal{A}_4$, soit $\mathcal{C}_3 \uplus \mathcal{C}_1$, soit $\mathcal{C}_2 \uplus \mathcal{A}_2$.*
4. *si $|Q| \geq 5$, Q est $\mathcal{C}_{|Q|-1} \uplus \mathcal{C}_1$.*

5.6.3 Ordres dont la 2-dimension est la hauteur

De la même manière qu'à la section précédente on a identifié les cas d'égalité avec la borne supérieure de la 2-dimension, on peut s'intéresser aux cas d'égalité avec la borne inférieure avec la hauteur et chercher à caractériser les ordres P tels que $dim_2(P) = h(P)$. Contrairement aux ordres MQ-critiques, la reconnaissance de tels ordres est difficile comme l'indique le théorème suivant.

Théorème 5.41. *Décider si un ordre P vérifie $dim_2(P) = h(P)$ est un problème $\mathcal{NP}$-complet.*

Preuve. Sans perte de généralité, on ne considère que les ordres ayant un minimum et un maximum. En effet, soit P un ordre, si P n'a pas de minimum on lui en ajoute un et si P n'a pas de maximum on lui en ajoute un, et ces transformations ne modifient pas sa 2-dimension.

Le fait que P admette un minimum et un maximum permet de reprendre les raisonnements de la proposition 5.39 et d'obtenir de la même manière le résultat suivant : soit P un ordre avec un minimum et un maximum et $\mathcal{C}_{k+1}$ une chaine de hauteur k, alors en mettant P et $\mathcal{C}_{k+1}$ en parallèle, on a $dim_2(P \uplus \mathcal{C}_{k+1}) = \max(dim_2(P), k) + 2$.

Maintenant soit P un ordre avec un maximum et un minimum, et k un entier positif tel que $k \geq h(P)$ (sinon la question $dim_2(P) \leq k$? est immédiatement résolue par la négative) et tel que $k \leq |P|$ (sinon la question $dim_2(P) \leq k$? est immédiatement résolue par la positive). Alors $dim_2(P) \leq k$ si et seulement si $dim_2(\underline{1} \oplus (P \uplus \mathcal{C}_{k+1}) \oplus \underline{1}) = h(\underline{1} \oplus (P \uplus \mathcal{C}_{k+1}) \oplus \underline{1})$. En effet $h(\underline{1} \oplus (P \uplus \mathcal{C}_{k+1}) \oplus \underline{1}) = \max(h(P), k) + 2 = k + 2$ et $\max(dim_2(P), k) +$

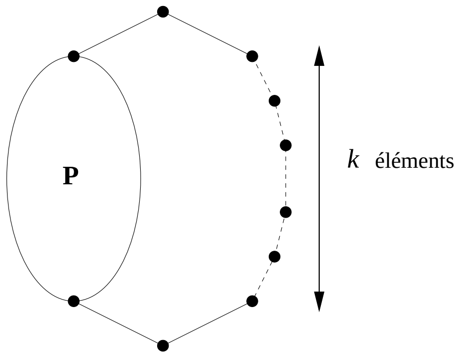


FIG. 5.11 – L'ordre $\underline{1} \oplus (P \uplus \mathcal{C}_{k+1}) \oplus \underline{1}$ utilisé dans le théorème 5.41.

$2 = k + 2$ si et seulement si $dim_2(P) \leq k$. La figure 5.11 représente l'ordre $\underline{1} \oplus (P \uplus \mathcal{C}_{k+1}) \oplus \underline{1}$ construit à partir de P.

Comme le problème de décision associé à la 2-dimension est $\mathcal{NP}$-complet et que la réduction est polynomiale (car $k \leq |P|$), la reconnaissance des ordres tels que $dim_2(P) = h(P)$ est aussi un problème $\mathcal{NP}$-complet. ■

5.6.4 Treillis extrémaux et distributifs

On connait la 2-dimension de certains treillis. Certains en particulier vérifient l'égalité $dim_2(T) = \min(|J(T)|, M(T)|)$.

Proposition 5.42 (Nourine, Habib [HN94]). *Soit $T = (X, \leq_T)$ un treillis. Si T est $\vee$-extrémal (resp. $\wedge$-extrémal), alors $dim_2(T) = |J(T)| = h(T)$ (resp. $dim_2(T) = |M(T)| = h(T)$).*

Preuve. Avec la proposition 5.16 et le corollaire 5.15, on sait que $h(T) \leq dim_2(T) \leq \min(|J(T)|, |M(T)|)$. Si T est $\vee$-extrémal (resp. $\wedge$-extrémal), alors par définition $h(T) = |J(T)|$ (resp. $h(T) = |M(T)|$) et on a donc la 2-dimension. Dans ce cas l'application $\phi_\vee$ (resp. $\phi_\wedge$) définie dans l'exemple 5.5 fournit un codage optimal. ■

Les treillis distributifs étant $\vee$-extrémaux et $\wedge$-extrémaux, on a le corollaire immédiat.

Proposition 5.43. *Soit $T = (X, \leq_T)$ un treillis distributif. Alors $dim_2(T) = h(T) = |J(T)| = |M(T)|$.*

Le résultat sur les extrémaux est sous-jacent dans le travail de Markowski [Mar80]. Le cas des treillis distributifs est présent dans [Tro75] et [HN94]. A l'inverse, on ne connait pas de caractérisation des treillis T tels que $dim_2(T) = \min(|J(T)|, |M(T)|)$.

Question 3. *Existe-t-il une caractérisation intéressante des treillis T tels que $dim_2(T) = |J(T)|$ (resp. $dim_2(T) = |M(T)|$) ?*

Plus généralement, on sait d'après la proposition 5.10 que pour un treillis T, si on connait un ordre P tel que $DM(P)$ est isomorphe à T (par exemple $(J(T) \cup M(T), \leq_T)$) et si on connait $dim_2(P)$ et un codage optimal de P, alors on connait $dim_2(T) = dim_2(P)$ et on peut en déduire un codage optimal de T.

Concernant la complexité du problème pour les treillis dans le cas général, la réduction utilisée dans la section 5.5 ne permet pas de conclure à la $\mathcal{NP}$-complétude. Cette réduction construit un ordre qui a priori est loin d'être un treillis et passer au complété de Dedekind-MacNeille pour obtenir un treillis avec la même 2-dimension risque de faire exploser le nombre de sommets (exponentiel par rapport au nombre de sommets de l'ordre initial dans certains cas), la réduction n'étant plus polynomiale dans ce cas. Le problème pour les treillis est encore ouvert.

Question 4. *Quelle est la complexité du calcul de la 2-dimension des treillis ? Est-ce que le calcul de la 2-dimension des treillis est approximable en temps polynomial ? Avec quels ratios ?*

Il est vraisemblable que le calcul de la 2-dimension des treillis corresponde à un problème de décision $\mathcal{NP}$-complet. Et il y a des chances pour que les résultats de non-approximabilité pour les ordres en général s'étendent aux treillis.

5.6.5 Peigne

Soit deux entiers $d \geq 1$ et $h \geq 0$. Un ***peigne*** de degré d et de hauteur h est un ordre à $d.h+1$ éléments, possédant un minimum $\perp$, où pour tout élément $x \neq \perp$, $|ImPred(x)| = 1$, et avec une chaine $x_0 < x_1 < \ldots < x_h$ de hauteur h telle que pour tout élément x_i, $i < h$, de cette chaine, $|ImSucc(x_i)| = d$. La figure 5.12 représente un peigne de degré 4 et de hauteur 4

Proposition 5.44. *La 2-dimension d'un peigne de degré d et de hauteur h vaut $h + sp(d) - 1$.*

Preuve. Remarquer d'abord que tout peigne P de degré d et de hauteur h admet $\mathcal{C}_h \oplus \mathcal{A}_d$ comme sous-ordre. D'après la proposition 5.19, on a $dim_2(\mathcal{C}_h \oplus$

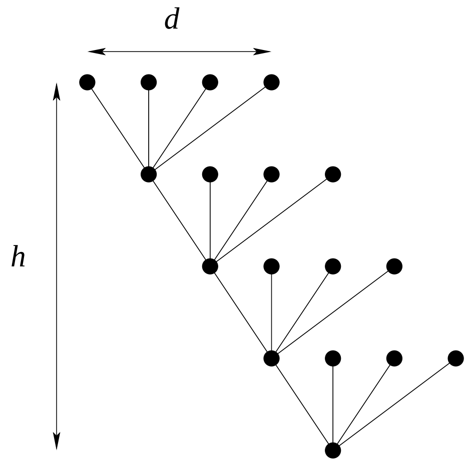


FIG. 5.12 – Peigne de degré $d = 4$ et de hauteur $h = 4$

$\mathcal{A}_d) = h - 1 + sp(d)$. Par conséquent la 2-dimension de P est supérieure ou égale à $h + sp(d) - 1$. Nous allons maintenant exhiber un codage bit-vecteur de taille $h + sp(d) - 1$ pour P. Le codage est en fait décrit pour le dual du peigne car le codage réduit s'exprime mieux sur ce dual P^d.

Soit $x_h > \ldots > x_1$ la chaine de P^d telle que pour $1 \leq i \leq h$, $|ImPred(x_i)| = d$. On note $A_0 = ImPred(x_1)$, puis $A_1 = Impred(x_2)\backslash\{x_1\}$, ..., $A_{h-1} = Impred(x_h)\backslash\{x_{h-1}\}$. Le codage réduit est le suivant :

- Les éléments de A_0 recoivent le codage optimal de l'antichaine de largeur d, utilisant les couleurs 1, ..., $sp(d)$.
- Pour $1 \leq i \leq h-1$, les $d-1$ éléments de A_i recoivent le codage optimal de l'antichaine de largeur d plus la couleur $sp(d) + i$ dans chacun de leur code, et x_i reçoit $\emptyset$ comme code réduit.

On vérifie que ce codage réduit à $sp(d)+h-1$ couleurs fournit bien un codage bit-vecteur du dual du peigne P. La figure 5.13 illustre ce codage. ∎

5.6.6 Arbre complet

Soit deux entiers $d \geq 1$ et $h \geq 0$. Un ***arbre complet*** de degré d et de hauteur h est un ordre à $(d^{h+1} - 1)/(d-1)$ éléments, possédant un minimum $\perp$, où pour tout élément $x \neq \perp$, $|ImPred(x)| = 1$, tel que pour tout élément non maximal x, $|ImSucc(x)| = d$ et où toutes les feuilles ont pour hauteur h. La figure 5.14 représente un peigne de degré 3 et de hauteur 3

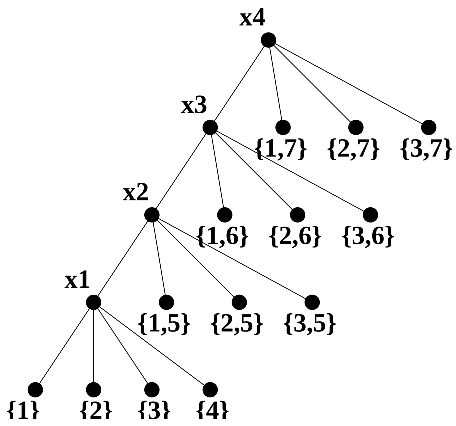


FIG. 5.13 – Codage réduit optimal du dual d'un peigne

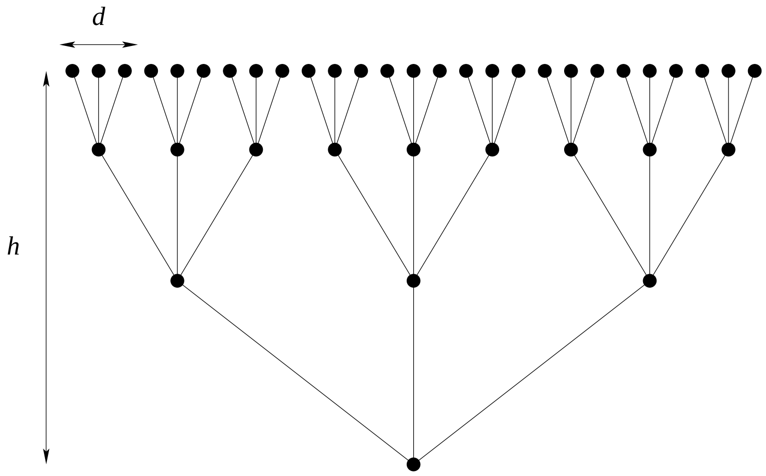


FIG. 5.14 – Arbre complet de degré $d = 3$ et de hauteur $h = 3$.

Proposition 5.45. *La 2-dimension d'un arbre complet de degré d et de hauteur h est inférieure ou égale à $h.sp(d)$.*

Preuve. Nous allons décrire un codage réduit à $h.sp(d)$ couleurs. Comme pour le peigne, on utilise des codages optimaux de l'antichaine de largeur d utilisant $sp(d)$ couleurs. Les couleurs sont des entiers de 1 à $h.sp(d)$.

Le minimum $\perp$ reçoit le code vide. Puis pour tout élément x, les d fils de x reçoivent le codage optimal de l'antichaine de largeur d, en utilisant les couleurs de $h(x) + 1$ à $h(x) + sp(d)$ (où $h(x)$ désigne la hauteur de x dans

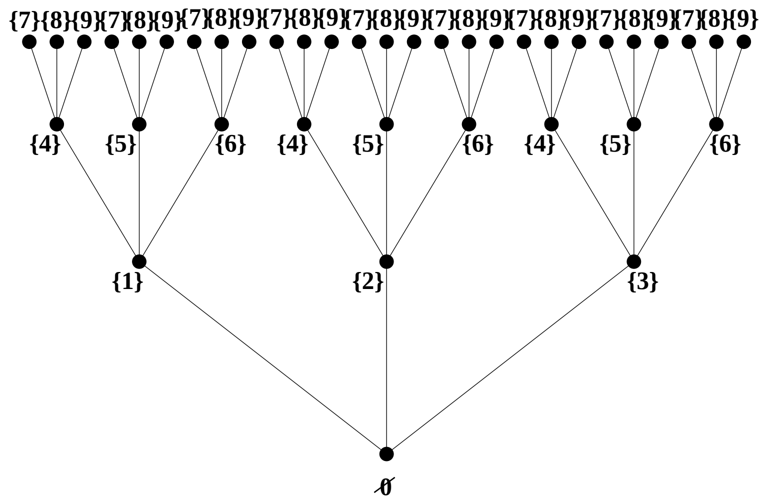


FIG. 5.15 – Codage réduit optimal (?) d'un arbre complet

l'arbre). ■

Apparemment le codage présenté est optimal, mais la preuve n'en est pas encore faite.

Conjecture 1. *La 2-dimension d'un arbre complet de degré d et de hauteur h vaut exactement h.sp(d).*

5.6.7 Couronne

Une ***couronne*** S_n^k, où $n \geq 3$ et $2 \leq k \leq n-1$, est un ordre biparti $(X_n, Y_n, <)$ défini sur $X_n = \{x_1, \ldots, x_n\}$ et $Y_n = \{y_1, \ldots, y_n\}$ par $Succ(x_i) = \{y_i, y_{i+1}, \ldots, y_{i+k-1}\}$ (sachant que les indices sont lus de manière cyclique, par exemple y_{n+2} correspond à y_2).

La 2-dimension des couronnes a été étudiée par Trotter (le 2-dimension exacte annoncée dans [Tro75] s'est avérée inexacte) et surtout Stahl [Sta84] qui ont prouvé quelques bornes. Certaines de ces bornes ont été améliorées par Schumacher. Nous nous bornons ici à donner les résultats de Stahl.

Théorème 5.46 (Stahl [Sta84]). *1. Soient deux entiers r et s tels que $r \geq 2$, $s \geq 2$, et $2 \leq k \leq \lfloor r/2 \rfloor + 2$, $n = r^{s-1}(r-k+2)$. Alors $dim_2(S_n^k \leq s.r$.*

2. Il existe un entier d_0 tel que pour tout entier $d > d_0$, on ait :

$$\max\{n \mid dim_2(S_n^2) \leq d\} \geq \frac{2^d}{2,51.t^{5,98}}.$$

5.6.8 Idéaux de treillis booléen

Comme la dimension, la 2-dimension des ordres est un paramètre difficile à évaluer et des bornes plus fines que celles de la section 5.3 sont très difficiles à calculer. Trotter et Kierstead ont par exemple étudié la dimension et la 2-dimension de certains idéaux des treillis booléens. Les bornes obtenues nécessitent une compréhension très pointue de la structure de ces ensembles.

Soit $\mathcal{B}_n$ le treillis booléen sur X avec $|X| = n$. Chaque élément de $\mathcal{B}_n$ est un sous-ensemble de X. On note $\mathcal{B}_n(1, k)$ le sous-ordre induit sur les sous-ensembles de cardinal 1 et de cardinal k. Cet ordre a la même 2-dimension que l'idéal de $\mathcal{B}_n$ composé des sous-ensembles de X de cardinal $\leq k$ (c'est son complété de Dedekind-MacNeille, privé de $\top$). On note alors $dim_2(1, k; n)$ la 2-dimension de $\mathcal{B}_n(1, k)$.

Nous présentons maintenant une borne obtenue par Trotter dans [Tro75].

Théorème 5.47 (Trotter [Tro75]). *Soit $k \geq 3$. Alors il existe une constante positive c telle que pour tout $n \geq k$, $dim_2(1, k; n) \leq c \log_2(n)$.*

Les deux propositions suivantes sont des corollaires issus d'un théorème de Kierstead [Kie96] qui généralise et affine cette borne de Trotter.

Proposition 5.48 (Kierstead [Kie96]). *Soit $1 \leq k \leq n$. Alors $dim_2(1, k; n) \leq e^2 k^2 \ln(n)$. En particulier, $dim_2(1, \ln(n); n) \leq e^3 (\ln(n))^3$.*

En prenant $k = 1$, on trouve $dim_2(1, 1; n) = sp(n) \leq e^2 \ln(2) \log_2(n)$, une inégalité un peu moins précise que celle de la proposition 5.14 puisque $e^2 \ln(2) \simeq 5.1$.

Proposition 5.49 (Kierstead [Kie96]). *Soit $1 \leq k \leq n$. Alors $dim_2(1, k; n) \leq 2k^2 (\log_2(n))^2 / (\log_2(k))^2$. En particulier, $dim_2(1, n^\epsilon; n) \leq 2\epsilon^{-2} n^{2\epsilon}$.*

Peu de classes d'ordres ont été étudiées du point de vue de la 2-dimension. Par conséquent, les bornes présentées ici, ainsi que dans la section 5.6.7, sont autant d'exemples ou de contre-exemples pour l'étude de la 2-dimension. Mais le principal intérêt de ces résultats réside dans les techniques déployées pour les prouver, comme par exemple l'utilisation d'arguments probabilistes.

5.7 Heuristiques : la simple coloration

Le problème du calcul de la 2-dimension et a fortiori la génération de codages bit-vecteur optimaux sont donc des problèmes intrinsèquement difficiles.

Du fait de cette difficulté, une orientation naturelle est d'élaborer des heuristiques de calcul pour se rapprocher le plus près et le plus souvent possible de l'optimal (on sait maintenant grâce au théorème 5.37 qu'on ne peut pas garantir dans le cas général des ratios d'approximation intéressants dans le cas général, à moins que $\mathcal{P} = \mathcal{NP}$).

L'une d'elle, l'***heuristique de simple coloration***, a la propriété de formaliser un type particulier de codages bit-vecteur et ensuite de proposer une stratégie pour générer de tels codages.

Nous allons juste présenter les objets manipulés et les théorèmes utilisés. Pour des détails et des preuves, le lecteur est renvoyé aux articles mentionnés dans le texte et à la thèse d'Olivier Raynaud [Ray00].

Simple coloration des ordres

Définition 5.50. *Soit $P = (X, \leq_P)$ un ordre. Un codage bit-vecteur ϕ de P est une **simple coloration** de P si son codage réduit vérifie $\forall x \in X$, $|\phi_r(x)| \leq 1$.*

Introduite par Caseau dans [Cas93], puis étudiée notamment par Krall, Vitek et Horspool [KVH97] puis Caseau, Habib, Nourine et Raynaud [CHNR99], ce type de codage bit-vecteur englobe les codages utilisés par plusieurs autres auteurs comme [AKBLN89, HHN95, HNR97]. Il s'agit bien sûr à nouveau d'essayer de minimiser la taille d'une simple coloration. L'espoir d'une telle approche était que la simplification du problème de codage initial permette de résoudre ce nouveau problème d'optimisation en temps polynomial.

Les simples colorations d'un ordre $P = (X, \leq_P)$ sont liés aux colorations d'un graphe de conflit $G_{simple}(P)$. Ce graphe a juste besoin pour sommets des éléments de $J(P)$ qui sont les sup-irréductibles de son complété de Dedekind-MacNeille $DM(P)$ et que l'on peut caractériser sans passer par la construction explicite de $DM(P)$. Un élément $j \in X$ est dans $J(P)$ s'il existe un élément $x \in X$ tel que $j \not\leq_P x$ et $Pred(j) \subseteq Pred(x) \cup \{x\}$. De manière duale on peut définir $M(P)$ à partir des inf-irréductibles de $DM(P)$. Ensuite dans $G_{simple}(P)$, deux sommets $j, j' \in J(P)$ sont adjacents si et seulement si il existe $m \in X$ tel que ($jm \in Cr(P)$ et $j' \leq_P m$) ou ($j'm \in Cr(P)$ et $j \leq_P m$).

Théorème 5.51 (Caseau *et al* [CHNR99]). *Soit $P = (X, \leq_P)$ un ordre. La plus petite simple coloration de P est de taille $\chi(G_{simple}(P))$.*

Krall, Vitek et Horspool ont proposé une autre réduction vers la coloration de graphes. Mais leur graphe de conflit qui a le même ensemble de sommets contient les arêtes de $G_{simple}(P)$ plus d'autres arêtes qui en fait ne sont pas nécessaires, elles détectent de "faux" conflits. Par conséquent leur graphe de conflit a un nombre chromatique supérieur à $\chi(G_{simple}(P))$ (voir [CHNR99]). Il faut aussi noter que Krall, Vitek et Horspool proposent plusieurs types de précalculs pour modifier un peu la structure de l'ordre initial et ensuite calculer de meilleurs simples colorations. Même s'ils ne quantifient pas le gain de cette phase de précalcul (paramétrable), c'est ainsi qu'ils obtiennent sur des benchmarks leurs meilleurs résultats.

Le lien avec la coloration de graphes est très fort mais il faut remarquer que tout graphe n'est pas graphe de conflit $G_{simple}(P)$ d'un ordre P. Par conséquent, on pouvait espérer tirer partie de structures intéressantes des graphes de conflit. En fait trouver la plus petite simple coloration d'un ordre est associé à un problème $\mathcal{NP}$-complet comme cela a été démontré dans [CHNR99] (sauf pour le cas des arbres qui sera présenté dans la section 5.8).

Théorème 5.52 (Caseau, Nourine, Habib, Raynaud [CHNR99]). *Le calcul de la plus petite simple coloration d'un ordre est associé à un problème de décision $\mathcal{NP}$-complet.*

Noter que la réduction présentée dans [CHNR99] qui part de la coloration de graphe, ne permet pas de transférer les résultats de non-approximabilité de la coloration de graphe au calcul de la plus petite simple coloration (ce n'est pas une réduction continue). La question de l'approximation de la plus petite simple coloration d'un ordre reste donc un problème ouvert.

Les auteurs de [CHNR99] proposent des algorithmes pour calculer $J(P)$, $Cr(P)$ puis $G_{simple}(P)$, ainsi que pour calculer un codage bit-vecteur à partir d'une coloration de $G_{simple}(P)$. Nous en donnons les complexités. Soit P un ordre à n éléments, donné par sa matrice de fermeture transitive (donc avec un espace en $\mathcal{O}(n^2)$). Les ensembles $J(P)$ et $Cr(P)$ peuvent être construits en temps $\mathcal{O}(n^3)$. Le graphe $G_{simple}(P)$ peut être construit en temps $\mathcal{O}(n^3)$. On peut stocker ce graphe par exemple sous forme de sa matrice d'adjacence ou

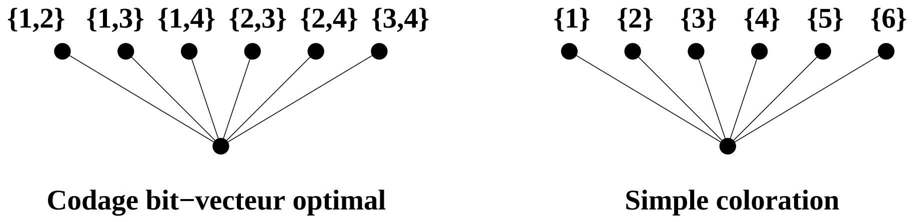


FIG. 5.16 – Codage optimal et simple coloration de l'antichaine $\mathcal{A}_6$

sous forme de ses listes d'adjacence (plus approprié pour certains algorithmes gloutons de coloration). A partir d'une coloration de $G_{simple}(P)$, on peut alors construire un codage bit-vecteur de P en temps $\mathcal{O}(n^3)$.

L'algorithme de construction du graphe de conflit de [KVH97] et la construction d'un codage bit-vecteur après coloration sont aussi en temps $\mathcal{O}(n^3)$ et en espace $\mathcal{O}(n^2)$.

Par comparaison, si on veut avoir une chance d'obtenir un codage bit-vecteur optimal de l'ordre P, il faut construire le graphe de conflit critique $G_{cr}(P)$ présenté dans le théorème 5.32. Comme on l'a vu, le calcul de $Cr(P)$ peut se faire en temps $\mathcal{O}(n^3)$. D'après la définition de $G_{cr}(P)$, le test d'adjacence entre deux éléments de $Cr(P)$ se fait en temps constant. Par conséquent, comme $|Cr(P)|$ est en $\mathcal{O}(n^2)$, il n'est pas forcément nécessaire de stocker la matrice d'adjacence de $G_{cr}(P)$ dans un espace en $\mathcal{O}(n^4)$, et on peut préférer effectuer les tests d'adjacence à la demande. Les algorithmes de colorations risquent alors de demander un peu plus de temps, mais le gain en espace est important, on reste alors avec un espace en $\mathcal{O}(n^2)$. La construction d'un codage bit-vecteur de P à partir d'une coloration de $G_{cr}(P)$ demande alors un temps $\mathcal{O}(n^3)$.

Pour un ordre P, puisque $\chi(G_{simple}(P)) \leq |J(P)| \leq |P|$ et $dim_2(P) \geq \lceil \log_2(n) \rceil$, le ratio d'approximation de la 2-dimension par la simple coloration est toujours inférieur où égal à $n/\log_2(n)$ (cette borne est le ratio du plongement canonique) et dans le pire des cas, ce ratio est atteint (à une constante près). En effet, pour $n \geq 1$, toute simple coloration de l'antichaine $\mathcal{A}_n$ a pour taille n alors que $dim_2(\mathcal{A}) = sp(n) \leq 2\lceil \log_2(n) \rceil$. La figure 5.16 montre cette différence pour $n = 6$. Un autre exemple sera donné par certains peignes (voir le tableau 5.1 de la section 5.8 pour $log_2(d) \geq h$).

Quelques soient ces résultats théoriques, l'heuristique de simple coloration donne pour beaucoup d'ordres de biens meilleurs résultats que cette mauvaise borne d'approximation (voir les tableaux de performances de [KVH97], même dans le cas où aucun précalcul n'est effectué). D'ailleurs, il n'est pas évident que sur de gros ensembles ordonnées, une heuristique basée sur la coloration de $G_{cr}(P)$ donne de plus petits codages bit-vecteur que l'heuristique de simple coloration, car les performances des algorithmes de coloration chutent rapidement avec l'augmentation de la taille du graphe et quand $G_{simple}(P)$ a de l'ordre de n sommets, $G_{cr}(P)$ peut avoir de l'ordre de n^2 sommets.

Simple coloration des treillis

On peut d'abord noter que, pour un treillis, l'application $\phi_\wedge$ définie dans l'exemple 5.5 est une simple coloration. On pouvait se demander quelle est la complexité du calcul de la plus petite simple coloration d'un treillis. Cette question a été résolue par Fall [Fal98].

Théorème 5.53 (Fall). *Le problème de décision associé au calcul de la plus petite simple coloration d'un treillis est $\mathcal{NP}$-complet.*

Le preuve proposée par Fall établit une réduction à partir du problème du recouvrement des sommets d'un graphe par un nombre minimum de cliques (problème GT15 dans [GJ79]). Ici aussi, sa réduction ne permet pas d'étendre le résultat de $\mathcal{NP}$-complétude à un résultat de non-approximabilité. Pour un graphe G à n sommets, m arêtes et recouvrable par k cliques, il construit un treillis T à $2n + m + 1$ éléments et de 2-dimension $n + k$.

Conjecture de Caseau et extensions

Concernant la simple coloration, Caseau a établi une conjecture proposant une borne supérieure sur le nombre minimum de couleurs nécessaires [Cas93]. Soit $P = (X, \leq_P)$ un ordre et C une chaine $x_0 <_P x_1 <_P \ldots <_P x_h$ de P, il définit le poids de cette chaine par $poids(C) = \sum_{0 \leq i \leq h} deg(x_i)$ où $deg(x_i) = |ImSucc_P(x_i)|$. Constatant que si P est un arbre, alors $\chi(G_{simple}(P)) = \max\{poids(C) \mid C \text{ est une chaine de } P\}$, il pose comme conjecture que ce résultat s'étend à tous les treillis T sous forme de l'inégalité $\chi(G_{simple}(T)) \leq \max\{poids(C) \mid C \text{ est une chaine de } T\}$. Cependant un contre-exemple générique à cette conjecture a été trouvé dans [Ray97], il est représenté sur la figure 5.17. On peut vérifier que la plus petite simple coloration est de taille $\chi(G_{simple}(P)) = l + h$, alors que $\max\{poids(C) \mid C \text{ est une chaine de } P\} = \max\{h + 3, l + 3\}$. Ceci fournit un contre-exemple à la conjecture de Caseau dès que $l \geq 4$ et $h \geq 4$.

Une autre conjecture est proposée dans [HHN95] : elle étend l'inégalité à la 2-dimension dont on sait que la valeur est clairement inférieure à la taille

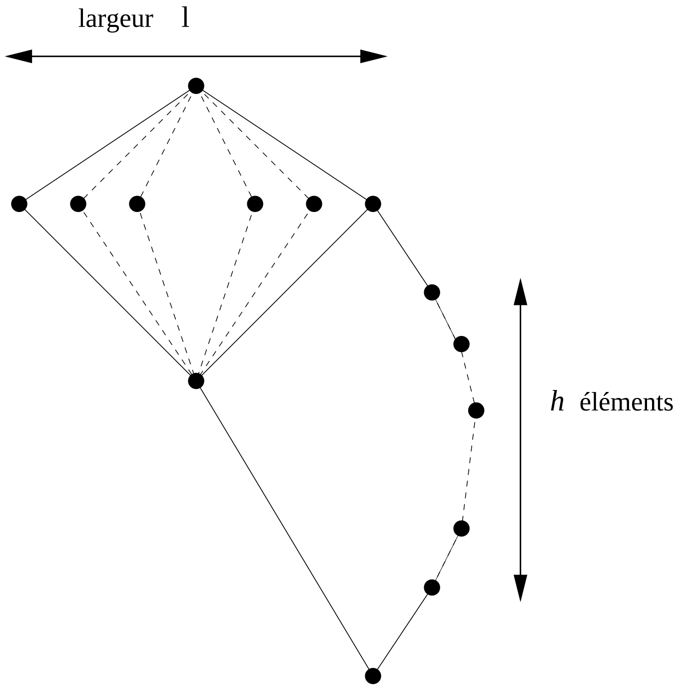


FIG. 5.17 – Contre-exemple générique à la conjecture de Caseau

de la plus petite simple coloration. Et pour l'instant on ne dispose ni de preuve, ni de contre-exemple.

Conjecture 2. *Soit $T = (X, \leq_T)$ un treillis. Alors*

$$dim_2(T) \leq \max\{poids(C) \mid C \text{ est une chaine de } T\}$$

Le contre-exemple de la figure 5.17 admet un codage bit-vecteur de taille $3 + \max\{h-1, l-1\}$ (considérer la décomposition modulaire de l'ordre et utiliser la propriété 5.22 sur l'opération de substitution) et ce n'est donc plus un contre-exemple pour cette nouvelle conjecture.

Par contre, cette conjecture ne s'étend pas aux ordres en général (ceux qui ont un minimum, ce qui permet de considérer des chaines partant de ce minimum). Pour trouver un contre-exemple, il suffit de considérer pour $n \geq 0$, l'ordre biparti $S_n = (X_n, Y_n, \leq_{S_n})$ où $X_n = \{x_1, \ldots, x_n\}$, $Y_n = \{y_1, \ldots, y_n\}$ et tel que $x_i \leq_{S_n} y_j$ si et seulement si $i \neq j$. Il est connu que son complété de Dedekind-MacNeille $DM(S_n)$ est isomorphe au treillis booléen $\underline{2}^n$, et que

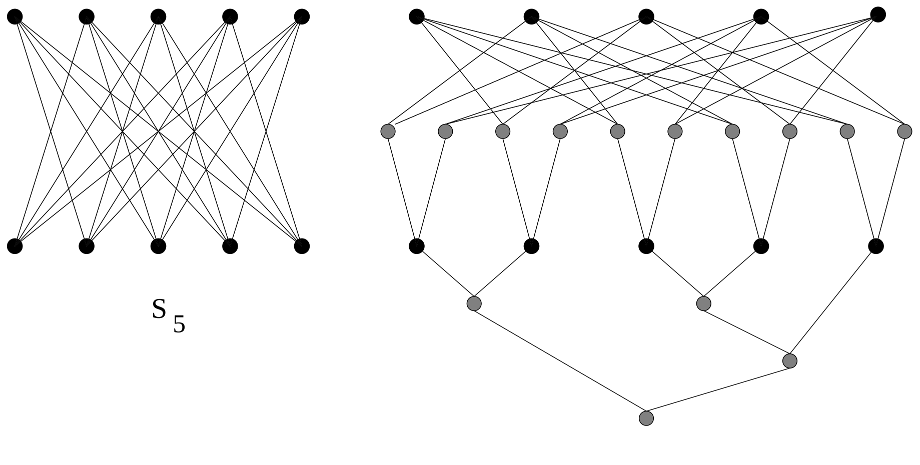


FIG. 5.18 – Contre-exemple à la conjecture 2 pour les ordres

par conséquent $dim_2(S_n) = n$. A partir de ce biparti S_n, on peut construire un nouvel ordre P_n de la manière suivante :

1. On ajoute un minimum $\perp$.
2. Pour chaque élément $x \in X_n \cup \{\perp\}$, on transforme le sous-ordre induit sur $\{x\} \cup ImSucc(x)$ en un arbre binaire complet de racine x et dont les feuilles sont les éléments de $ImSucc(x)$. Il suffit de rajouter localement $|ImSucc(x)| - 2$ éléments et le poids maximum d'une chaine allant de x à un élément de $ImSucc(x)$ est alors $2\lceil \log_2(|ImSucc(x)|) \rceil$ (alors que dans S_n, c'était $|ImSucc(x)|$).

Cette construction de P_n à partir de S_n est présentée sur la figure 5.18 dans le cas où $n = 5$. On vérifie facilement que :

$$\max\{poids(C) \mid C \text{ est une chaine de } P_n\} \leq 4\lceil \log_2(n) \rceil.$$

Mais comme S_n est un sous-ordre de P_n, $dim_2(P_n) \geq n$. L'inégalité de la conjecture 2 n'est pas vraie pour P_n.

Codages atomiques

Une heuristique de codage bit-vecteur a été proposée par Habib, Huchard et Nourine [HHN95]. Une lecture attentive de cet article montre que l'heuristique de codage obtenue en considérant l'***arbre atomique*** (et non pas l'arbre coatomique) est une heuristique de simple coloration. Elle revient à colorer les sommets de $G_{simple}(P)$ selon une extension linéaire de P bien choisie. On appelle ***codage atomique*** un codage bit-vecteur obtenu avec cette heuristique. Aucune analyse de performances n'est connue pour cette heuristique, et il est possible qu'il existe un lien avec la conjecture de Caseau dans le cas des treillis, c'est ce qui est formulé dans la question suivante.

Question 5. *Soit T un treillis, est-ce que les codages atomiques sont de taille inférieure ou égale à* $\max\{poids(C) \mid C \text{ est une chaine de } T\}$ *? Dans le cas contraire, exhiber un contre-exemple.*

En revenant à la 2-dimension des ordres en général, on peut se poser la question de savoir s'il existe des inégalités reliant, pour un ordre $P = (X, \leq_P)$, $dim_2(P)$ avec des paramètres comme $\max\{poids(C) \mid C$ est une chaine de $P\}$, $\Delta(P) = \max\{|ImSucc_P(x)| \mid x \in X\}$, $D(P) = \max\{|Succ_P(x)| \mid x \in X\}$ ou $\Delta_{comp}(P) = \max\{|Succ_P(x)| + |Pred_P(x)| \mid x \in X\}$, ou bien avec d'autres paramètres à identifier. Ces inégalités pourraient être à l'image de celles qui existent pour la *dimension* $dim(P)$ d'un ordre P. En effet, Füredi et Kahn ont établi que $dim(P) < 2(D(P)+1)\log(|P|)$ et Rödl et Trotter ont montré que $dim(P) \leq 2\Delta_{comp}(P)^2 + 2$, ce qui a été amélioré par Füredi et Kahn avec la borne $dim(P) \leq 50\Delta_{comp}(P)(\log(\Delta_{comp}(P)))^2$ [Tro91].

5.8 Heuristiques : le cas des arbres

Le cas des arbres est intéressant à plusieurs points de vue. Tout d'abord, on connait peu de classes d'ordres où le calcul de la 2-dimension est polynomial. La plupart du temps la classe des arbres est une sous-classe des classes étudiées. C'est pourquoi une question naturelle est de s'interroger sur la 2-dimension des arbres.

Problème 4 (2DIM-ARBRE). *Quelle est la complexité du calcul de la 2-dimension des arbres ? A défaut de calculer exactement la 2-dimension, peut-on approximer sa valeur dans le cas des arbres ?*

De plus, la structure d'arbre est très courante en informatique, on peut donc supposer que les cas pratiques sont souvent des arbres ou des ordres proches de la structure d'arbre (on verra dans la section 5.9.3, où sont présentés des benchmarks issues des langages orientés objets, que cette hypothèse est souvent vérifiée). Une bonne connaissance des codages bit-vecteur des arbres peut donc s'avérer très utile et déboucher sur de nouvelles heuristiques, comme nous le verrons dans la section 5.9.

Remarque 5.54. *Pour revenir sur les classes d'ordres ayant les arbres comme sous-classes, on peut mentionner les **treillis** (pour peu qu'on ait rajouté un élément maximum $\top$ à l'arbre ce qui ne change pas la 2-dimension), les **ordres de dimension 2** et plus précisément les **ordres série-parallèle** (pour des définitions de ces classes, voir par exemple la synthèse de Möhring [Moh89]).*

5.8.1 Précédents travaux

Avant de parler des codages bit-vecteur, précisons que le codage des arbres en général a beaucoup été étudié. On a pu le voir dans le Chapitre 2, avec en particulier l'algorithme de Cohen [Coh91] et surtout le ***Relative Numbering*** aussi appelé codage de Schubert [SPT83]. Cette dernière méthode est optimale à presque tous les points de vue.

Pour revenir à la méthode de codage par vecteurs de bits, Caseau est le premier à avoir précisément étudié les codages bit-vecteur dans le cas particulier des arbres [Cas93]. C'est aussi à cette occasion qu'il s'est intéressé aux codages par simple coloration. Dans le cas des arbres, il a ainsi établit que l'on peut calculer en temps linéaire la plus petite simple coloration. Soit C une chaine $x_0, x_1, \ldots, x_p$ d'un arbre T, il définit le poids de C par $poids(C) = \sum_{0 \leq i \leq p} deg(x_i)$. La taille de la simple coloration produite par l'algorithme de Caseau pour T est le maximum des poids des chaines de T, et c'est une simple coloration optimale. Un algorithme simple permet de calculer le codage réduit correspondant à une simple coloration optimale. Il suffit d'effectuer un parcours en largeur de l'arbre en partant de la racine et pour chaque élément x, si les fils du père de x ont reçu comme couleurs les entiers allant de c_{min} à c_{max}, on donne à chacun des $deg(x)$ fils de x une couleur en allant de $c_{max}+1$ à $c_{max}+deg(x)$. La génération de ce codage réduit s'effectue donc en temps $\mathcal{O}(n)$ où n est le nombre d'éléments.

Dans [CHNR99], Caseau, Habib, Nourine et Raynaud sont revenu sur cette méthode dans le cas particulier des arbres en proposant un algorithme basé sur le codage des antichaines. Une fonction de poids est à nouveau définie sur chaque chaine C de l'arbre T, par $poids(C) = \sum_{0 \leq i \leq p} sp(deg(x_i))$, avec $sp(n) = dim_2(\mathcal{A}_n) \approx \log_2(n)$. Leur algorithme produit un codage bit-vecteur dont la taille est le maximum des poids des chaines de T. Ce codage a une

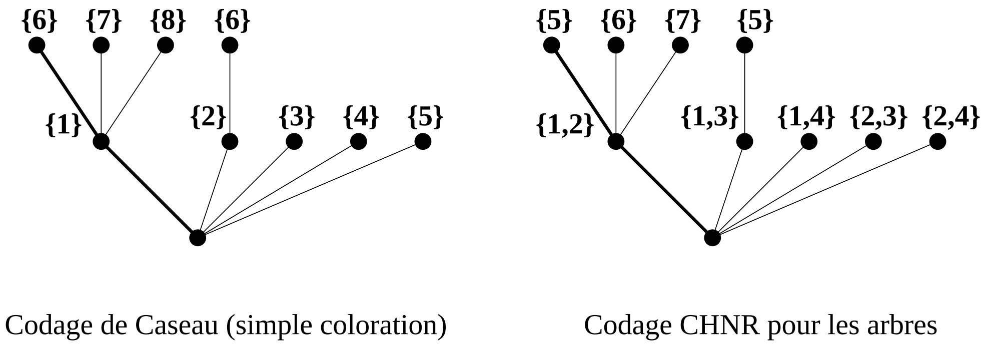


FIG. 5.19 – Codages réduits construits par l'algorithme de Caseau et celui de CHNR.

plus petite taille que la simple coloration (qui en fin de compte n'est pas la plus appropriée pour coder les arbres), la figure 5.19 représente chacun de ces deux codages sur un exemple. On peut générer ce codage par un parcours en largeur de l'arbre comme précédemment sauf qu'ici pour chaque élément x, si les fils du père de x ont reçu comme couleurs les entiers allant de c_{min} à c_{max}, on donne comme codes réduits aux $deg(x)$ fils de x un codage optimal de l'antichaine de largeur $deg(x)$ en utilisant les couleurs allant de $c_{max}+1$ à $c_{max} + sp(deg(x))$. Un tel algorithme calcule le codage réduit en temps $\mathcal{O}(n\Delta(P))$.

Les sections suivantes présentent un nouvel algorithme pour les arbres. Il est basé sur deux idées qui avaient déjà été soulevée dans [Cas93] et [KVH97] : diviser et équilibrer (“splitting and balancing”).

5.8.2 Codages dichotomiques

Nous allons présenter un algorithme calculant des codages bit-vecteur très compacts pour les arbres. D'abord, afin de formaliser les stratégies de division et d'équilibrage des arbres, nous introduisons une classe particulière de codages bit-vecteur : les ***codages dichotomiques***. Ensuite nous décrivons notre algorithme qui calcule un codage dichotomique. Nous prouverons alors qu'il s'agit du plus petit codage dichotomique possible.

L'idée consiste à diviser en deux parties l'arbre initial en donnant une couleur différente à chacune des deux parties. En répétant cette opération récursivement sur chacune des parties, on produit ce que l'on appelle un

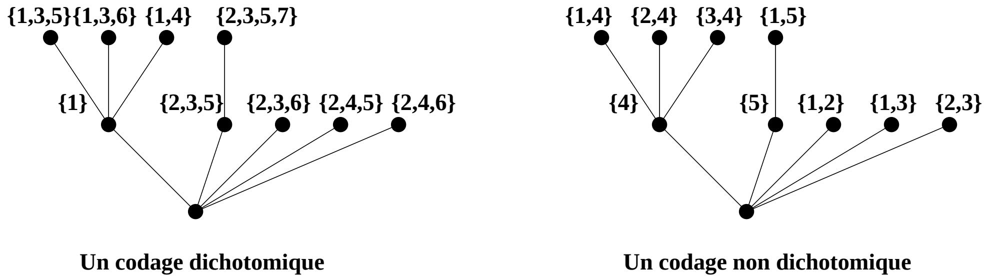


FIG. 5.20 – Un codage dichotomique et un codage qui n'est pas dichotomique.

codage dichotomique.

Soit T un ordre avec un codage bit-vecteur, soit x une couleur de ce codage, on note T_x l'ensemble des éléments de T qui contiennent x dans leur code. Nous donnons maintenant une définition récursive des codages dichotomiques et un exemple est présenté sur la figure 5.20

Définition 5.55. *Soit T un ordre, on dira d'un codage bit-vecteur de T qu'il est **dichotomique** s'il existe deux couleurs x et y telles que :*

1. *Chaque élément de T, de code non vide, contient soit x, soit y dans son code.*
2. *Les codes de T_x privés de x forment un codage dichotomique de T_x.*
3. *Les codes de T_y privés de y forment un codage dichotomique de T_y.*
4. *Si T est réduit à un unique élément, alors le code vide est un codage dichotomique de T.*

La première remarque est que seuls les arbres ou les forêts admettent des codages dichotomiques : si, dans un ordre, un élément u a deux prédecesseurs immédiats v et w, il ne peut pas exister un codage avec deux couleurs c_1 et c_2 telles que le code de v contient c_1, le code de w contient c_2 et le code de u ne contient qu'une seule de ces deux couleurs.

On appelle cette technique "dichotomique" car on divise l'ordre en deux parties qui peuvent ensuite être codées indépendamment l'une de l'autre. On peut aussi remarquer qu'un codage dichotomique est équivalent à un plongement de l'arbre dans un ***arbre binaire*** (arbre où tous les noeuds sont de degré ≤ 2).

Proposition 5.56. *Soit T un arbre. A tout codage dichotomique est associé un plongement d'ordre dans un arbre binaire et inversement.*

Preuve. Etant donné T et un codage dichotomique de T, on construit un arbre binaire BT avec un codage bit-vecteur réduit, en associant d'abord le minimum de T à celui de BT. Puis pour deux couleurs x et y vérifiant les trois premières conditions de la définition 5.55, on construit deux fils du minimum avec comme codage réduit respectivement x et y et comme successeurs respectivement T_x et T_y. Et ainsi de suite en repartant de chacun de ces deux fils avec respectivement mes codes de T_x privés de x et les codes de T_y privés de y ... Si à un moment T_x (ou T_y) est vide, il n'est pas nécessaire d'ajouter le fils lui correspondant. Pour finir, le plongement consiste simplement à faire correspondre les éléments de T avec ceux de BT ayant le même code.

Dans l'autre sens, étant donné un plongement de T dans un arbre binaire BT, il suffit de voir qu'il existe une manière très simple de construire un codage dichotomique de BT et par conséquent de T. On construit un codage réduit. Le minimum de BT reçoit le code vide, puis ses deux fils u et v recoivent respectivement comme codes réduits $\{1\}$ et $\{2\}$. Ensuite on code indépendamment les successeurs de u et ceux de v et ainsi de suite, en utilisant à chaque fois deux nouvelles couleurs pour les codes réduits des deux fils d'un noeud, sachant qu'on peut utiliser les mêmes couleurs pour deux sous-arbres disjoints puisqu'ils ont déjà été "séparés" : leurs codes hériteront respectivement de deux couleurs différentes. Par exemple si u a deux fils, ils recevront respectivement comme codes réduits $\{3\}$ et $\{4\}$, et si v a aussi deux fils, ils recevront aussi comme codes réduits $\{3\}$ et $\{4\}$ ... Si un noeud n'a qu'un seul fils, il suffit d'utiliser une seule nouvelle couleur comme code réduit pour ce fils. On s'arrête aux feuilles et le code réduit final fournit bien un codage dichotomique de BT. ■

Cette équivalence est représentée sur la figure 5.21. Pour le plongement dans l'arbre binaire, les images des éléments de l'arbre initial sont les sommets pleins.

Pour un arbre donné, il existe beaucoup de codages dichotomiques. Nous montreront que notre algorithme en construit un de taille minimum.

5.8.3 L'algorithme

C'est un algorithme glouton "du haut vers le bas" qui construit un arbre binaire à partir de l'arbre T en entrée en introduisant des noeuds intermédiaires (symbolisant l'opération de "division") et qui propage des poids sur les noeuds pour "équilibrer" cet arbre binaire. Le poids d'un élément s, noté $poids(s)$, correspond au nombre de couleurs suffisantes pour coder le sous-

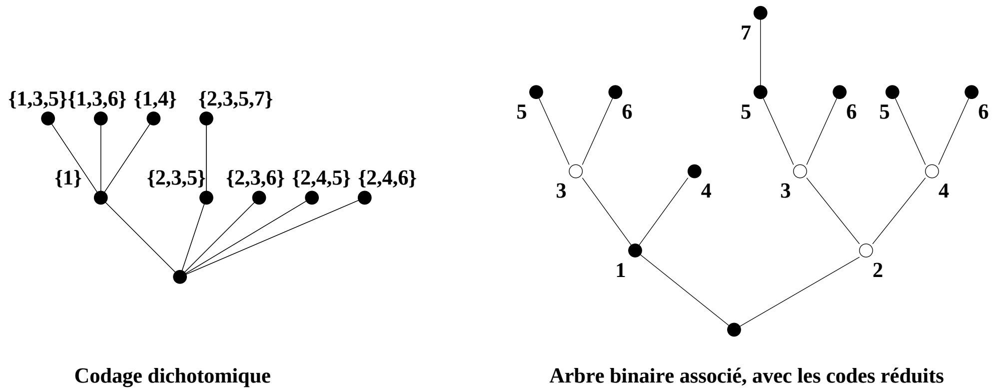


FIG. 5.21 – Un arbre avec un codage dichotomique, l'arbre binaire associé avec son codage réduit

arbre composé de s et de ses successeurs. Ce sous-arbre est noté $T(s)$. Le calcul des poids suit un parcours des éléments du haut vers le bas où il faut calculer les poids de tous les fils d'un noeud pour pouvoir calculer le poids de ce noeud.

Au départ, seules les feuilles portent un poids, égal à zéro. Ensuite en supposant que tous les fils d'un noeud s portent un poids, trois cas peuvent se produire.

- Si s a un unique fils v, alors le sous-arbre $T(s)$ peut être codé avec $poids(v) + 1$ couleurs. Une couleur pour coder v et $poids(v)$ autres couleurs pour coder le sous-arbre $T(v)$.
- Si s a exactement deux fils u et v tels que $poids(u) \leq poids(v)$, on peut coder $T(u)$ avec les couleurs utilisées par $T(v)$ et coder u et v avec deux nouvelles couleurs. Ce codage de $T(s)$ utilise $poids(s) + 2$ couleurs.
- Si s a plus de deux fils, pour que notre méthode produise un codage très compact, on va marrier les deux sous-arbres $T(u)$ et $T(v)$ de poids minimal (u et v sont les fils de s avec les deux plus petits poids). On introduit un nouveau noeud w ayant u et v comme fils. Ce noeud w remplace u et v dans la liste des fils de s et avec la même méthode que au-dessus on peut coder $T(w)$ avec $\max(poids(u), poids(v)) + 2$ couleurs.

L'algorithme 2 suit cette stratégie. Il calcule ainsi un arbre binaire BT où se plonge l'arbre T. A la fin chaque noeud s de BT (c'est à dire les noeuds de T et les nouveaux noeuds intermédiaires) a une unique couleur stockée dans $c(s)$ qui correspond au code réduit de notre codage dichotomique. La dernière étape consiste à propager ces couleurs à travers l'ordre pour obtenir le codage final.

Algorithme 2: [***Dicho***] Génère un codage dichotomique pour un arbre.

Entrée : Un arbre T.
Sortie : Un codage dichotomique de T.
début
  Associer à chaque feuille v de l'arbre T le poids : $poids(v) \leftarrow 0$;
  Les autres sommets ne portent aucun poids ;
1 **tant que** *il reste un sommet s sans poids dont tous les fils portent un poids* **faire**
    si *s a un unique fils v* **alors**
      Associer à v l'élément : $c(v) \leftarrow poids(v) + 1$;
2   Associer à s le poids : $poids(s) \leftarrow poids(v) + 1$;
    si *s a exactement 2 fils u et v* **alors**
      Associer à u l'élément : $c(u) \leftarrow \max(poids(u), poids(v))+1$;
      Associer à v l'élément : $c(v) \leftarrow \max(poids(u), poids(v))+2$;
      Donner à s le poids : $poids(s) \leftarrow \max(poids(u), poids(v))+2$;
    si *s a au moins 3 fils* **alors**
      Prendre u et v ses deux fils de plus petit poids ;
      Introduire un nouvel élément w père de u et v, et fils de s ;
      Associer à u l'élément : $c(u) \leftarrow \max(poids(u), poids(v))+1$;
      Associer à v l'élément : $c(v) \leftarrow \max(poids(u), poids(v))+2$;
      Donner à w le poids : $poids(w) \leftarrow \max(poids(u), poids(v))+2$;
3 Soit BT l'arbre obtenu à partir de T, associer à chaque sommet s de T le code suivant : $code(s) = \{c(s') | s' \in BT \text{ et } s' \leq_{BT} s\}$;
fin

La figure 5.22 montre une exécution de l'algorithme 2 qui génère l'arbre BT à partir de l'arbre T. Les noeuds intermédiaires sont en blanc, la couleur $c(s)$ associée à un noeud s est donnée par un entier, l'entier entre parenthèses correspond au poids porté par le noeud. Le dernier schéma représente l'arbre initial avec son codage final (après l'étape 3).

Concernant l'implantation de cet algorithme, on peut effectuer la boucle 1 en suivant un ordre post-fixe de l'arbre. En suivant un tel ordre, lorsqu'on ar-

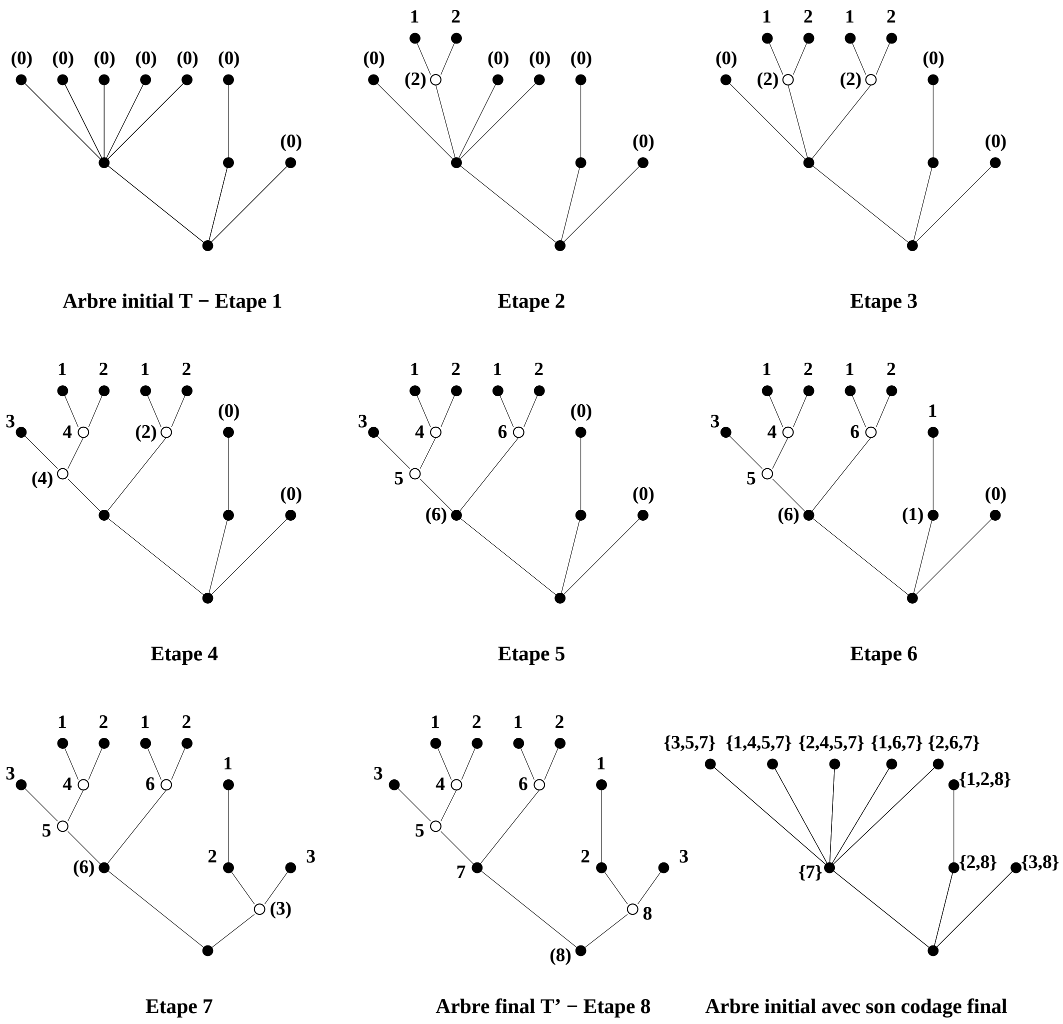


FIG. 5.22 – L’arbre initial T, l’arbre binaire BT et le *codage dichotomique* final calculé par l’algorithme 2.

rive sur un nouveau sommet s tous ses fils ont déjà été traités avant et donc ils portent tous des poids. Dans ce cas, on effectue un boucle sur l’ensemble des fils de s pour calculer $poids(s)$ en suivant les règles de l’algorithme. Soit $deg(s)$ le nombre de fils de s, alors le calcul de $poids(s)$ à partir du poids de ses fils peut s’effectuer en commençant par trier ces poids en ordre croissant (complexité en temps $\mathcal{O}(deg(s)\log(deg(s)))$ puis on balaie cette liste L de poids par ordre croissant en faisant les mises à jour qui consistent à faire apparaitre de nouveaux éléments, à les insérer dans l'arbre et à réinsérer leurs poids dans la liste L. (complexité en temps $\mathcal{O}(deg(s)\log(deg(s)))$, qui est peut être améliorable en $\mathcal{O}(deg(s))$?). Si l'arbre initial a n éléments et est donné avec sa réduction transitive, la calcul d'un ordre post-fixe se fait en temps $\mathcal{O}(n)$, l'exécution de la boucle 1 s'effectue en temps $\mathcal{O}(\log \Delta(T). \sum_{s\in X} deg(s))$ soit $\mathcal{O}(n\log(\Delta(T)))$ (on rappelle que $\Delta(P) = \max\{deg(s)|s \text{ élément de } X\}$) et le calcul du codage total à l'étape 3 en temps $\mathcal{O}(n.k)$ où k est la taille du codage dichotomique.

Les deux sections qui suivent discutent des performances de cet algorithme.

5.8.4 Optimalité

Le théorème suivant garanti que dans le cadre des stratégies "diviser et équilibrer", notre algorithme fournit un codage bit-vecteur de taille optimale. Vitek *et al* avaient conjecturé que cela était faisable en temps polynomial dans [KVH97].

Théorème 5.57. *La taille du codage généré par l'algorithme Dicho est la plus petite taille pour un codage dichotomique.*

Preuve. La preuve que nous allons développer montre que tout codage dichotomique de taille minimum peut être transformé en un codage dichotomique généré par l'algorithme *Dicho* sans changer de taille.

Produire un codage dichotomique d'un arbre T est équivalent à plonger T dans un arbre binaire BT (voir proposition 5.56). Alors la taille du codage dichotomique est égale au poids maximum d'une chaine de l'arbre binaire, au sens de Caseau (le poids d'une chaine est la somme des degrés des noeuds de cette chaine). Plus généralement, on peut associer à chaque élément s de BT un poids $p(s)$ qui est le poids maximum d'une chaine commençant par s. Ce poids est équivalent au nombre de couleurs utilisées pour coder le sous-arbre de T de racine s quand on considère les codes initiaux privés du code de s. La taille du codage est donc le poids de la racine.

Pour prouver que notre algorithme calcule un codage dichotomique de taille minimum, il suffit de prouver localement qu'il fait la meilleure construction pour chaque ***niveau*** de l'arbre initial, autrement un noeud et ses fils qui portent des poids. La figure 5.23 illustre ce qu'est un niveau.

Supposons que tous les fils d'un noeud s portent un poids. Nous allons prouver que notre construction minimise le poids qui va être propagé au

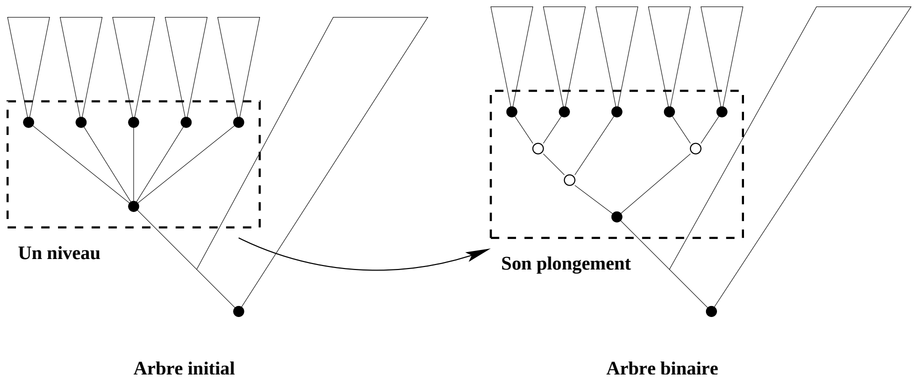


FIG. 5.23 – Un niveau de l'arbre initial et son plongement.

noeud s. Le cas où s n'a qu'un fils est trivial, on s'intéresse donc au cas où s a au moins deux fils.

1. Considérons un plongement de ce niveau dans un arbre binaire BT. Chaque feuille x porte un poids $p(x)$. Nous associons à chaque élément x sa hauteur $h(x)$ dans BT. Le poids de BT, noté $poids(BT)$, est défini comme le maximum de $p(x) + 2 \times h(x)$ sur toutes les feuilles x de BT.
2. Supposons que deux feuilles α et β de BT avec les deux poids mimimum $p(\alpha)$ et $p(\beta)$ ne soient pas "mariées" (les fils d'un même noeud). Alors on peut construire un autre arbre binaire BT' dans lequel se plonge le niveau initial, tel que α et β sont "mariées" et $poids(BT') \leq poids(BT)$. La figure 5.24 schématise cette configuration. Les noeuds α et β ont un premier ancêtre commun qui est s', α est "mariée" avec l'élement γ et β est "mariée" avec l'élément δ. La hauteur du père de α et γ (resp. β et δ) à partir de s' est notée h_γ (resp. h_δ). Supposons maintenant que $h_\gamma \leq h_\delta$, alors en échangeant α et δ on obtient un nouvel arbre binaire BT' tel que $poids(BT') \leq poids(BT)$ (si on avait eu $h_\gamma \geq h_\delta$, alors on aurait échangé β et γ).
3. Supposons maintenant que l'on supprime deux feuilles "mariées" α et β de BT et que l'on associe à leur père commun (qui est maintenant une feuille) le poids $\max(p(\alpha), p(\beta)) + 2$. Alors il est clair que ce nouvel arbre binaire BT' vérifie $poids(BT') = poids(BT)$.

Etant donné un niveau avec des poids sur les feuilles, considérons un plongement de ce niveau dans un arbre binaire BT de poids minimum. En

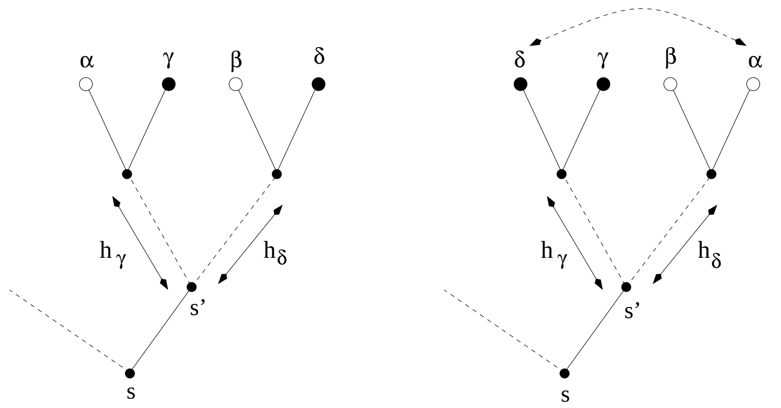


FIG. 5.24 – Transformation du plongement quand $h_\gamma \leq h_\delta$.

utilisant alternativement les transformations 2 et 3, on peut transformer BT en un autre arbre binaire BT' avec le même poids optimal et qui est un arbre binaire qui peut être construit par l'algorithme *Dicho* (et tous les arbres binaires construits par l'algorithme *Dicho* ont le même poids).

Cette preuve a la même structure que certaines preuves de l'optimalité des codes de Huffman pour la compression de données [CLR94] : il s'agit aussi d'algorithme glouton, d'arbres binaires dont on cherche à minimiser un paramètre et la preuve passe par la transformation d'un arbre binaire optimal en un arbre généré par l'algorithme. ■

5.8.5 Analyse de performances

Evaluation pour des cas particuliers

On s'intéresse ici à deux exemples génériques qui sont les peignes et les arbres complets. La définition de ces deux classes d'arbres est donnée dans la section 5.6.

Ces exemples donnent une bonne idée du comportement des algorithmes de codage. Le tableau 5.1 compare les tailles de codage de trois algorithmes : l'algorithme de Caseau [Cas93], sa variante par CHNR [CHNR99] et l'algorithme 2. La méthode décrite par KVH dans [KVH97] n'apparait pas car la taille du codage (qui est au final une simple coloration comme Caseau) dépend du choix des paramètres de précalcul (pour modifier un peu la structure de l'arbre), sachant que les critères de choix ne sont pas complètement clairs dans [KVH97].

Les instances, peigne et arbres complet, sont de degré d et de hauteur h. On rappelle que pour tout entier n, $sp(n) \approx \log_2(n)$.

Instance	Sommets	dim_2	Caseau	CHNR	*Dicho*
Arbre complet	$\frac{d^{h+1}-1}{d-1}$	$h.sp(d)?$	$h.d$	$h.sp(d)$	$2h\lceil\log_2(d)\rceil$
Peigne	$h.d+1$	$h+sp(d)$	$h.d$	$h.sp(d)$	$2(h+\lceil\log_2(d)\rceil)$

TAB. 5.1 – Comparaison de différents algorithmes de codage

On constate que l'algorithme initial de Caseau et donc la simple coloration n'est pas très appropriée pour coder directement un arbre. Par contre, l'algorithme *Dicho* qui effectue d'abord un équilibrage de l'arbre puis une simple coloration se rapproche bien des optima.

Ratios d'approximation

En regardant le cas du peigne et de l'arbre complet, ainsi que de nombreux exemples, on a remarqué que l'algorithme *Dicho* donne des résultats proches des optima. Plus précisément, ces constatations nous ont amené à faire la conjecture suivante.

Conjecture 3. *L'algorithme Dicho approxime la 2-dimension des arbres à un facteur 2 près.*

Tout d'abord à partir de la proposition suivante, nous allons prouver que cette conjecture est vraie pour les arbres de degré ≤ 4.

Proposition 5.58. *Soit T un arbre dont la racine a exactement 2 fils, soient T_1 et T_2 les deux sous-arbres de T dont les racines sont les deux fils de T. Soit n la 2-dimension de T, alors si la 2-dimension de T_1 vaut $n-1$, alors T_2 a une dimension $\leq n-3$.*

Preuve. Soient x_1 et x_2 les deux fils de la racine. Soit ϕ un plongement optimal de T dans $\underline{2}^S$, $|S| = n$. L'application ϕ_1 de T_1 qui associe à tout x dans T_1, $\phi_1(x) = \phi(x)\backslash\phi(x_1)$ est un codage bit-vecteur de taille $|S| - |\phi(x_1)|$. Comme $\phi(x_1) \neq \emptyset$ et la 2-dimension de T_1 vaut $n-1$, on a nécessairement $|\phi(x_1)| = 1$. On a donc $\phi(x_1) = \{a\}$, avec $a \in S$.

Etudions maintenant le code de x_2. Clairement $\phi(x_2) \neq \emptyset$. Supposons que $\phi(x_2) = \{b\}$, $b \in S$. Alors comme x_2 est incomparable aux éléments de T_1, pour tout x dans T_1, $b \notin \phi(x)$ et donc ϕ_1 serait en fait une application de T_1

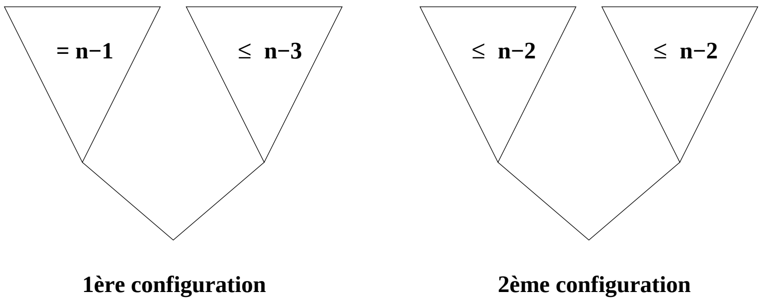


FIG. 5.25 – Configurations possibles de la 2-dimension pour un arbre de 2-dimension n dont la racine a 2 fils.

dans $\underline{2}^{S\backslash\{a,b\}}$, donc de taille $\leq n-2$, ce qui est impossible. Par conséquent, $|\phi(x_2)| \geq 2$.

Considérons l'application ϕ_2 définie pour tout x dans T_2 par $\phi_2(x) = \phi(x)\backslash\phi(x_2)$, c'est un codage bit-vecteur de T_2. Comme x_1 est incomparable à tous les éléments de T_2, pour tout x dans T_2, $a \notin \phi(x)$. Par conséquent ϕ_2 est un application de T_2 dans $\underline{2}^{S\backslash\{a\}\backslash\phi(x_2)}$, donc de taille $\leq n-3$. La 2-dimension de T_2 est donc $\leq n-3$. ■

Si la 2-dimension de T vaut n, comme la dimension de T_1, comme celle de T_2, est nécessairement $\leq n-1$, d'après la proposition précédente, seules deux configurations sont possibles. Elles sont schématisées par la figure 5.25, où les 2-dimensions possibles des sous-arbres sont indiquées dans chacun des triangles les représentant.

Proposition 5.59. *La conjecture 3 est vraie pour les arbres de degré ≤ 4.*

Preuve. Preuve par induction sur les sous-arbres. On va montrer que pour tout arbre T de degré ≤ 4, il existe un codage dichotomique de taille $\leq 2dim_2(T)$, sachant ensuite que l'algorithme *Dicho* produit un codage dichotomique de taille minimum et donc $\leq 2dim_2(T)$.

Pour un arbre réduit à un seul sommet, il suffit de considérer le code vide. Maintenant soit T un arbre de degré ≤ 4 et de 2-dimension égale à n. Nous considérons le pire des cas, c'est à dire quand la racine a 4 fils (les autres cas avec moins de fils se traitent pareil). Soient T_1, T_2, T_3, T_4 les 4 sous-arbres correspondants, de 2-dimension respectives n_1, n_2, n_3, n_4. Supposons $n_1 \geq n_2 \geq n_3 \geq n_4$. Par hypothèse d'induction, il existe des codages dichotomiques pour chacun des 4 sous-arbres, de tailles respectives $2n_1$, $2n_2$, $2n_3$ et $2n_4$. On les prend par exemple sous la forme de plongements

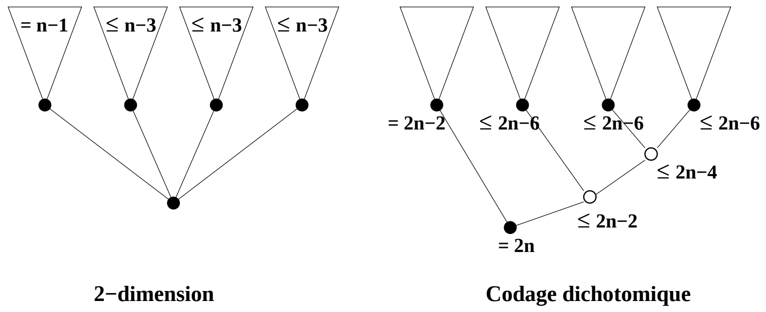


FIG. 5.26 – Pour un arbre de 2-dimension n, configuration où un des sous-arbres nécessite $n-1$ couleurs.

dans des arbres binaires.

Si $n_1 = n-1$, alors comme la racine avec T_1 et T_2 forment un sous-ordre de T donc de dimension $\leq n$ et d'après la proposition 5.58, la 2-dimension de T_2 vérifie $n_2 \leq n-3$. De même $n_3 \leq n-3$ et $n_4 \leq n-3$. Dans ce cas en utilisant les plongements de T_1, T_2, T_3 et T_4 et en terminant par la construction présentée sur la figure 5.26, on construit un plongement de T dans un arbre binaire tel que le codage dichotomique correspondant est clairement de taille $\leq 2n$. Sur la figure, on a indiqué à gauche les 2-dimensions des sous-arbres dans les triangles les représentant, et à droite le poids à coté d'un élément indique le nombre de couleurs utilisées pour coder le sous-arbre dont il est racine.

Si $n_1 \leq n-2$, alors $n_2 \leq n-2$, $n_3 \leq n-2$, $n_4 \leq n-2$. A la manière du cas précédent, on utilise les plongements de T_1, T_2, T_3, T_4, en finissant par une construction différente qui est représentée sur la figure 5.27. Les notations de la figure sont les mêmes que la précédente figure. A nouveau, on construit ainsi un codage dichotomique de T de taille $\leq n$.

■

Conjecture 4. *Soit T un arbre de racine $\perp$ avec k fils, soient T_1,T_2,...,T_k les k sous-arbres de T dont les racines sont les k fils de T. Soit n la 2-dimension de T, alors il existe une partition de ces k sous-arbres en deux, telle que chacune des deux sous-forêts résultantes soit de 2-dimension $\leq n-1$.*

Proposition 5.60. *La conjecture 4 implique la conjecture 3.*

Preuve. Preuve par induction sur la taille de l'arbre (ou la 2-dimension). L'idée est aussi de construire un codage dichotomique de T de taille $\leq$

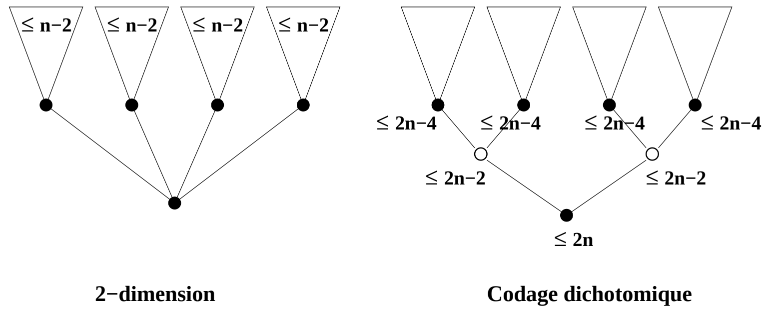


FIG. 5.27 – Pour un arbre de 2-dimension n, configuration où tous les sous-arbres nécessitent $\leq n-2$ couleurs.

$dim_2(T)$, sachant que l'algorithme *Dicho* en produit un forcément meilleur.

Pour un arbre réduit à un singleton, prendre le code vide. Sinon pour un arbre T de dimension n, prendre la partition en deux sous-forêts F_1 et F_2, chacune de 2-dimension $\leq n-1$. L'hypothèse d'induction s'applique à F_1 (quitte à rajouter une racine minimum pour en faire un arbre sans changer la 2-dimension) et de même à F_2. Il existe un codage dichotomique de F_1 (resp. F_2) de taille $\leq 2(n-1)$. On utilise les mêmes couleurs pour F_1 et F_2, et ensuite on rajoute deux nouvelles couleurs : une à ajouter aux codes des éléments de F_1, l'autre à ceux de F_2. On obtient ainsi un codage dichotomique de T à $2n$ couleurs. ■

Cette conjecture est par exemple vraie pour les antichaines (auxquelles on rajoute un minimum pour faire un arbre). Malheureusement cette conjecture s'est avérée fausse, un contre-exemple ayant été trouvé par Jean-Marc Lanlignel.

Proposition 5.61. *La conjecture 4 est FAUSSE. L'arbre T de la figure 5.28 est un contre-exemple : sa 2-dimension vaut 6 (cf. le codage représenté) mais la forêt constituée de deux des sous-arbres de racines les fils T, a aussi une 2-dimension égale à 6 (cf. proposition 5.58).*

En fait, on peut décrire précisément la fonction de l'algorithme *Dicho* qui calcule le poids d'un noeud à partir du poids de ses fils. Mais pour cela il faut introduire un petite ***variante*** de cet algorithme afin que les poids calculés soient toujours ***pairs***. Le seul cas pouvant faire apparaitre des poids impairs est le cas où un noeud a un seul fils, la variante que nous introduisons consiste à transformer l'instruction à l'étape 2

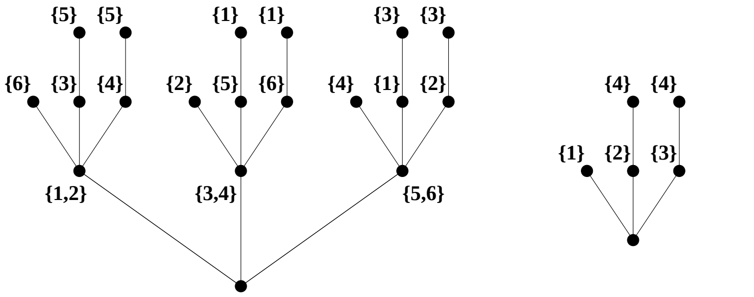


FIG. 5.28 – Un contre-exemple à la conjecture 4 à partir d'un motif de 2-dimension égale à 4.

"Associer à s le poids : $poids(s) \leftarrow poids(v) + 1$;"

par

"Associer à s le poids : $poids(s) \leftarrow poids(v) + 2$;".

Le nouvel algorithme calcule toujours un codage dichotomique pour un arbre T, mais de taille $codage(T)$ supérieure à celui généré par l'algorithme *Dicho* (égale si les noeuds de T ont toujours au moins deux fils). Le plus important est que $codage(T)$ est ainsi toujours pair et tous les poids calculés par la variante seront pairs.

On peut maintenant décrire pour cette variante quel poids est calculé à partir du poids des fils d'un noeud.

Proposition 5.62. *Soit T un arbre, soit s un noeud avec k fils qui portent respectivement les poids $2p_1, 2p_2, \ldots, 2p_k$. Alors le poids que reçoit s dans la **variante** de l'algorithme Dicho vaut $2p$ où $p = \min\{n \mid 2^{p_1} + 2^{p_2} + \ldots + 2^{p_k} \leq 2^n\}$ si $k \geq 2$, et $2.p_1 + 2$ si $k = 1$.*

Preuve. Preuve par récurrence sur le nombre de fils de s. Si s a un seul fils avec un poids $2p_1$, la variante donne comme poids à s la valeur $2p_1 + 2$ et la proposition est vérifiée. Si s a exactement deux fils avec les poids $2p_1$ et $2p_2$, en supposant $p_1 \leq p_2$, le poids calculé pour s vaut $2p_2 + 2$. Or on vérifie facilement que $p_2 + 1 = \min\{n \mid 2^{p_1} + 2^{p_2} \leq 2^n\}$.

Maintenant si $k \geq 3$, en supposant que les poids des k fils de s sont $2p_1 \leq 2p_2 \leq \ldots \leq 2p_k$, dans l'algorithme les fils de poids $2p_1$ et $2p_2$ sont remplacés par un fils de poids $2p_2 + 2$. Il reste à prouver que pour tout $n \geq 0$,

$2^{p_1} + 2^{p_2} + \ldots + 2^{p_k} \leq 2^n$ si et seulement si $2^{p_2+1} + \ldots + 2^{p_k} \leq 2^n$, ce qui montrera que le minimum des n est conservé après le remplacement. Le sens $\Leftarrow$ est clair puisque $2^{p_1} + 2^{p_2} \leq 2^{p_2+1}$.

Pour l'autre sens, supposons que $2^{p_1} + 2^{p_2} + \ldots + 2^{p_k} \leq 2^n$. On en déduit l'inégalité stricte $2^{p_2} + 2^{p_3} + \ldots + 2^{p_k} < 2^n$. En divisant cette inégalité par 2^{p_2}, on obtient $1 + 2^{p_3-p_2} + \ldots + 2^{p_k-p_2} < 2^{n-p_2}$. Et donc $2 + 2^{p_3-p_2} + \ldots + 2^{p_k-p_2} \leq 2^{n-p_2}$. En re-multipliant par 2^{p_2}, on obtient l'inégalité espérée $2^{p_2+1} + 2^{p_3} + \ldots + 2^{p_k} \leq 2^n$. ■

Conjecture 5. *Soit T un arbre de racine $\perp$ avec k fils, soient T_1,T_2,...,T_k les k sous-arbres de T dont les racines sont les k fils de T. Alors $2^{dim_2(T_1)} + 2^{dim_2(T_2)} + \ldots + 2^{dim_2(T_k)} \leq 2^{dim_2(T)}$.*

Proposition 5.63. *La conjecture 5 implique la conjecture 3.*

Preuve. Nous montrons que si la conjecture 5 est vraie, alors la ***variante*** de l'algorithme *Dicho* est une 2-approximation de $dim_2(T)$. La taille du codage engendré pour T par cette variante est toujours paire, on la note $2 \times algo(T)$. Montrons que $algo(T) \leq dim_2(T)$.

Preuve par récurrence sur le nombre d'éléments de l'arbre. Si l'arbre est réduit à un singleton, le code réduit valide la conjecture 3. Soit T un arbre dont la racine a un unique fils, soit T_1 le sous-arbre ayant ce fils pour racine, alors la variante de l'algorithme *Dicho* renvoie un codage tel que $algo(T) = algo(T_1) + 1$. Par hypothèse de récurrence $algo(T_1) \leq 2dim_2(T_1)$, or dans ce cas particulier on sait que $dim_2(T) = dim_2(T_1) + 1$, donc $algo(T) \leq dim_2(T)$.

Si la racine de T a k fils, $k \geq 2$, soient T_1,T_2,...,T_k les sous-arbres de racine ces fils. La variante produit un codage tel que $algo(T) = \min\{n \mid 2^{algo(T_1} + 2^{algo(T_2)} + \ldots + 2^{algo(T_k)} \leq 2^n\}$. Par hypothèse de récurrence, $2^{algo(T_1} + 2^{algo(T_2)} + \ldots + 2^{algo(T_k)} \leq 2^{dim_2(T_1)} + 2^{dim_2(T_2)} + \ldots + 2^{dim_2(T_k)} \leq 2^{dim_2(T)}$ en utilisant la conjecture 5 pour la dernière égalité. Donc $algo(T) \leq dim_2(T)$. ■

Proposition 5.64. *Si T est une forêt composée de chaines en parallèle, alors la conjecture 5 est vraie.*

Preuve. Soit $T = \mathcal{C}_{n_1} \uplus \ldots \uplus \mathcal{C}_{n_k}$ une forêt composée de k chaines en parallèle. On note $\perp_i$ (resp. $\top_i$) le minimum (resp. maximum) de $\mathcal{C}_{n_i}$. Considérons maintenant un plongement ϕ de T dans $\underline{2}^S$ avec $|S| = dim_2(T)$. Soit $1 \leq i \leq k$, $\phi_i(x) = \phi(x) \backslash \phi(\perp_i)$ constitue un codage bit-vecteur de $\mathcal{C}_{n_i}$ utilisant comme couleurs $\phi(\top_i) \backslash \phi(\perp_i)$. Par conséquent $|\phi(\top_i) \backslash \phi(\perp_i)| \geq dim_2(\mathcal{C}_{n_i})$. Considérons alors l'intervalle B_i de $\underline{2}^S$ défini par $B_i = \{A \subseteq S \mid \phi(\perp_i) \subseteq A \subseteq \phi(\top_i)\}$. Le cardinal de cet intervalle vaut $2^{|\phi(\top_i) \backslash \phi(\perp_i)|} \geq 2^{dim_2(\mathcal{C}_{n_i})}$.

Supposons que $\sum_{i=1}^{k} |B_i| > 2^{|S|}$. Cela implique qu'il existe au moins une partie A de S et $i \neq j$ tels que $A \in B_i \cap B_j$. Dans ce cas on aurait $\phi(\bot_i) \subseteq A \subseteq \phi(\top_j)$ et donc $\bot_i \leq_T \top_j$, ce qui est impossible. Nécessairement $\sum_{i=1}^{k} |B_i| \leq 2^{|S|}$ et donc $2^{dim_2(\mathcal{C}_{n_i})} + \ldots + 2^{dim_2(\mathcal{C}_{n_i})} \leq 2^{dim_2(T)}$.

Comme on peut le voir avec la preuve, cette inégalité est vraie plus généralement pour un ordre composé de k ordres en parallèle, chacun possédant un minimum et un maximum. ■

Corollaire 5.65. *L'algorithme Dicho approxime à 2 près la 2-dimension de chaines en parallèle.*

Remarque 5.66. *Dans ce cas très particulier, on peut remarquer que l'algorithme CHNR fournit un autre codage bit-vecteur dont on prouve facilement que c'est une 2-approximation. En effet pour k chaines en parallèle avec respectivement $n_1,\ldots,n_k$ éléments, l'algorithme CHNR produit un codage de taille $sp(k)+\max\{n_i-1 \mid 1 \leq i \leq k\}$, or on sait que la 2-dimension des k chaines en parallèle est supérieure à la hauteur de cet ordre càd $\max\{n_i - 1 \mid 1 \leq i \leq k\}$ et supérieure à $sp(k)$ car k est la largeur de cet ordre.*

La conjecture 5 est toujours ouverte. Le fait qu'elle mette en jeu des puissances de 2 n'est pas choquant au regard par exemple de la proposition 5.64 et d'autres résultats issus de la théorie de Sperner comme le théorème suivant de Seymour présenté dans le livre d'Anderson [And87] (exercice 6.4).

Théorème 5.67 (Seymour 1973). *Soit $\underline{2}^S$ le treillis booléen sur l'ensemble S, soit A et B deux sous-ensembles de $\underline{2}^S$. Si tout élément de A est incomparable avec tout élément de B, alors $2^{|A|/2} + 2^{|B|/2} \leq 2^{|S|/2}$.*

Même si la conjecture 3 reste ouverte, on peut quand même démontrer que l'algorithme *Dicho* fournit une approximation à une constante près de la 2-dimension des arbres. En fait, on va le prouver pour la variante avec les poids pairs introduite précédemment et dont les codages sont de taille supérieure à ceux de l'algorithme *Dicho*.

Proposition 5.68. *La variante de l'algorithme Dicho approxime la 2-dimension des arbres à un facteur 4 près.*

Preuve. On sait que pour un arbre T de hauteur h avec f feuilles, $dim_2(T) \geq \max(h, \lceil \log_2(f) \rceil)$. La variante de l'algorithme *Dicho* produit un codage de taille paire, cette taille est noté $2 \times algo(T)$ pour un arbre T. On va montrer par induction que $algo(T) \leq h + \log_2(f)$. On aura ainsi montré que la variante produit un codage de taille $\leq 2(h + \log_2(f))$. Sachant que $h + \log_2(f) \leq 2dim_2(T)$, il s'agit ainsi d'une 4-approximation.

La preuve que $algo(T) \leq h+\log_2(f)$ se fait par induction. Si l'arbre T est réduit à un singleton, l'algorithme renvoie le code vide, donc $algo(T) = 0$, or $h = 0$ et $f = 1$ donc l'inégalité est vérifiée.

Sinon si la racine de T a un unique fils x_1, soit T_1 le sous-arbre de T de racine x_1. L'algorithme calcule un codage de T à partir de d'un codage de T_1 tel que $algo(T) = algo(T_1) + 1$. Soit h_1 la hauteur de T_1 et f_1 le nombre de feuilles de T_1. On a $h = h_1 + 1$ et $f = f_1$. Par hypothèse de récurrence, $algo(T_1) \leq h_1+\log_2(f_1)$, et donc $algo(T) = algo(T_1)+1 \leq h_1+1+\log_2(f_1) = h + \log_2(f)$.

Maintenant si la racine de T a k fils x_1,...,x_k, avec $k \geq 2$. Pour $1 \leq i \leq k$, soit T_i le sous-arbre de T de racine x_i, on note h_i sa hauteur et f_i son nombre de feuilles. On a $h = \max_{1\leq i\leq k}(h_i) + 1$ et $f = \sum_{1\leq i\leq k} f_i$. Par hypothèse de récurrence, on a les inégalités :

$$\begin{array}{rcl} algo(T_1) & \leq & h_1 + \log_2(f_1) \\ algo(T_2) & \leq & h_2 + \log_2(f_2) \\ \dots & & \dots \\ algo(T_k) & \leq & h_k + \log_2(f_k) \end{array}$$

D'après la proposition 5.62, le codage de T calculé par l'algorithme vérifie $algo(T) = \min\{n \mid 2^{algo(T_1)} + 2^{algo(T_2)} + \ldots + 2^{algo(T_k)} \leq 2^n\}$. Or $2^{algo(T_1)} + 2^{algo(T_2)} + \ldots + 2^{algo(T_k)} \leq 2^{\max(h_i)}(f_1 + f_2 + \ldots + f_k) = 2^{h-1}f$, donc $algo(T) \leq \lceil h - 1 + \log_2(f)\rceil \leq h + \log_2(f)$. ■

D'autres variantes de l'algorithme *Dicho*

L'algorithme *Dicho* est basé sur une division en deux parties parallèles de l'arbre initial en associant une couleur différente à chacune des deux parties pour les séparer, ceci se continuant de manière récursive. On pourrait bien sûr proposer des variantes de cet algorithme : suivant les cas, diviser en autant de parties que nécessaire, sachant que pour séparer k parties parallèles il suffit d'utiliser $sp(k)$ couleurs, chaque partie recevant $\lceil sp(k)/2\rceil$ couleurs prises parmi les $sp(k)$ (codage optimal de l'antichaine à k éléments). Par exemple pour un arbre constitué d'une racine et une antichaine à n éléments, l'algorithme *Dicho* donne un codage à $2\lceil\log_2(n)\rceil$ couleurs, alors qu'en séparant directement les n éléments on obtient un codage à $sp(n)$ couleurs. On peut ainsi espérer améliorer les tailles de codage. Le problème revient alors à trouver un arbre dans lequel se plonge l'arbre initial, et de poids minimum au sens de CHNR.

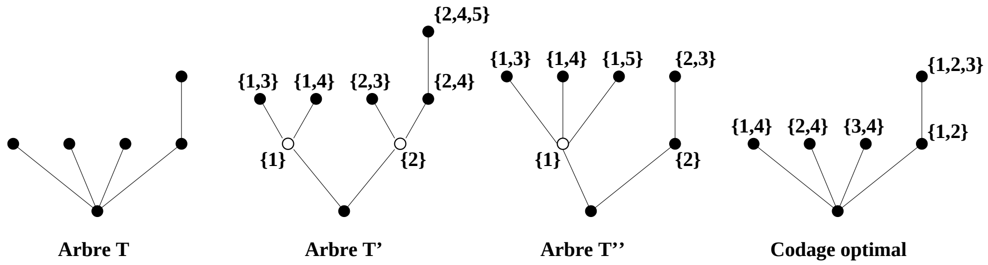


FIG. 5.29 – Un arbre T, un arbre binaire T' où se plonge T avec son codage dichotomique, un arbre T'' où se plonge T avec son codage CHNR, un codage optimal de T.

Plusieurs remarques s'imposent alors. D'abord si on autorise des divisions selon un nombre quelconque de parties, il faut définir une nouvelle stratégie d'"équilibrage" pour minimiser le nombre de couleurs total. C'était faisable en divisant à chaque fois juste en deux et en ajoutant deux couleurs, cela semble plus difficile si on doit manipuler les nombres $sp(n)$, $n \in \mathbb{N}$. De plus si on sépare k parties parallèles en utilisant $sp(k)$ nouvelles couleurs, on peut aussi les séparer avec $2\lceil \log_2(k) \rceil$ avec les codages dichotomiques. Comme $2\lceil \log_2(k) \rceil \leq 2sp(k)$, il est clair qu'on améliore au mieux d'un facteur 2 la taille du codage produit avec de tels variantes. Pour finir, même si une telle variante une fois mise au point peut améliorer les tailles de codage, elle n'assure pas d'obtenir la 2-dimension exacte comme on peut le voir sur la figure 5.29. Pour l'arbre initial T, l'arbre T'' est un arbre de poids minimum (au sens de CHNR) dans lequel se plonge T, pourtant $dim_2(T) = 4$ comme on le voit avec le codage le plus à droite.

Conclusion sur les arbres

En conclusion, le comportement de la 2-dimension n'est pas le même pour les arbres que dans le cas général puisqu'on est capable de fournir une approximation à une constante près, contrairement au cas général.

Pourtant, nous ne savons toujours pas si le calcul de la 2-dimension des arbres est polynomial, les dernières tentatives pour trouver des algorithmes et des preuves laissent penser que cela doit être le cas.

5.9 Heuristiques : application de la méthode dichotomique

5.9.1 Sous-arbre maximal

Pour des ordres dont la structure serait très proche des arbres, on pourrait imaginer la stratégie suivante :

1. Trouver le plus grand sous-ordre qui est un arbre.
2. Générer un codage bit-vecteur de ce sous-ordre avec l'algorithme 2.
3. Ajouter les éléments ne faisant pas partie de ce sous-ordre, en ajoutant de nouvelles couleurs selon la technique incrémentale de la propriété 5.17.

Cette heuristique ne peut produire un codage de taille réduite que si le nombre d'éléments à rajouter en étape 3 est très petit. De plus pour un ordre fixé, trouver le plus grand sous-ordre qui est un arbre est un problème difficile en général, comme le montre le théorème de $\mathcal{NP}$-complétude suivant.

Théorème 5.69. *Soit $T = (X, \leq_T)$ un treillis. Le problème consistant à trouver le plus grand sous-ordre de T (en nombre d'éléments) qui soit un arbre, correspond à un problème de décision $\mathcal{NP}$-complet.*

Preuve. On effectue une réduction à partir de MAXSTABLE (problème GT20 dans [GJ79]). Soit $G = (V, E)$ un graphe avec $|V| = n$ et $|E| = m$, construire le treillis T constitué d'abord d'une antichaine de n éléments v_1,...,v_n correspondant aux n sommets de V. Puis pour chaque arête de E d'extremités v_i et v_j dans G, ajouter un élément $e_{i,j}$ tel que $v_i \prec_T e_{i,j}$ et $v_j \prec_T e_{i,j}$. Pour finir ajouter un minimum $\perp$ et un maximum $\top$. La figure 5.30 illustre cette construction. On va montrer qu'il existe un stable max de cardinal $\geq k$ dans G si et seulement si il existe un sous-ordre de T de cardinal $\geq m + k + 1$ qui est un arbre.

Soit un sous-ensemble S de V qui est un stable G, c'est à dire pour tout $v_i \in S$ et $v_j \in S$, $i \neq j$, il n'existe pas d'arête entre v_i et v_j dans G. Alors considérer le sous-ordre de T constitué de $\perp$, des éléments v_i tels que $v_i \in S$ et de tous les éléments $e_{i,j}$. On vérifie que ce sous-ordre est bien un arbre, avec $m + |S| + 1$ éléments. Si $|S| \geq k$, alors ce sous-ordre est de cardinal $\geq m + k + 1$.

Inversement, soit T' un sous-ordre de T qui est un arbre et de cardinal $\geq m + k + 1$. Si $\perp$ n'appartient pas à T', on peut l'ajouter à T' qui reste un

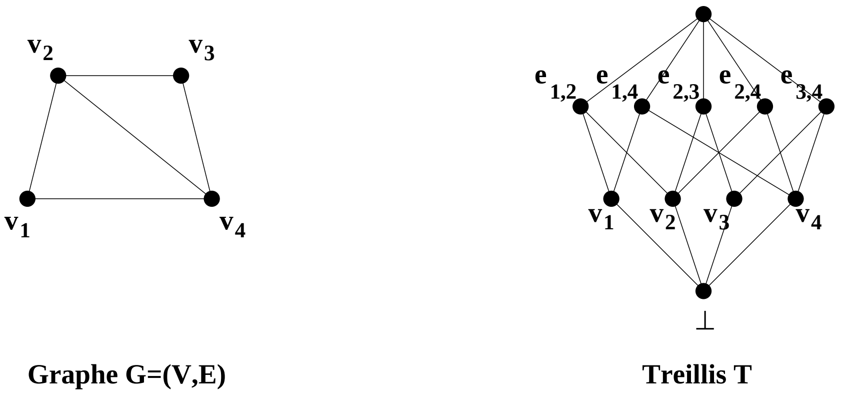


FIG. 5.30 – Un graphe $G = (V, E)$ et le treillis T construit dans la réduction.

arbre. Si v_1 appartient à T', mais qu'il existe un élément $e_{1,j}$ dans $ImSucc(v_1)$, $1 < j$, qui n'appartient pas à T' alors en enlevant v_1 de T' et en ajoutant $e_{1,j}$, T' reste un arbre de même cardinal. On fait la même transformation pour v_2, puis v_3 et ainsi de suite jusqu'à v_n : si v_i appartient à l'arbre T' en cours et qu'il existe un élément $e_{i,j}$ dans $ImSucc(v_i)$, $i < j$, qui n'appartient pas à T', alors on enlève v_i de T' mais on lui ajoute $e_{i,j}$. Au final, on obtient un sous-ordre qui est un arbre, de cardinal identique au cardinal initial de T' et donc $\geq m + k + 1$. De plus l'ensemble S des éléments v_i restant dans T' forme un stable de G et le cardinal de T' est clairement $\leq m + |S| + 1$. Par conséquent $|S| \geq k$. ■

5.9.2 Décomposition modulaire

Utiliser la décomposition modulaire est une approche courante pour aborder le codage des ordres. Plusieurs auteurs en font mention comme Ellis [Ell94] ou Fall [Fal95b], et en particulier Ait-Kaci *et al* proposent une technique de codage bit-vecteur basée sur la décomposition modulaire mais ils s'intéressent essentiellement aux treillis et aux calculs de bornes supérieures ou/et inférieures [AKBLN89].

Nous proposons une heuristique de codage bit-vecteur basée sur la décomposition modulaire et utilisant l'algorithme 2 pour traiter le cas "parallèle". Le principe est le suivant, il fait appel aux contructions de codages bit-vecteur présentées dans les preuves de certaines des propriétés de la section 5.3 :

1. A partir de la fermeture transitive, calculer l'arbre de décomposition modulaire de l'ordre.
2. Construire récursivement un codage en calculant à chaque étape le codage de l'ordre associé à un noeud de l'arbre de décomposition à partir du type de noeud rencontré et de codages des fils de ce noeud :
   - Noeud Série : utiliser la construction de la proposition 5.19.
   - Noeud Parallèle : soit k le nombre de fils de ce noeud, utiliser l'algorithme *Dicho* sur un niveau (arbre de hauteur 1) où les k feuilles portent déjà des poids qui sont les tailles des codages obtenus récursivement pour les k fils du noeud. Les codes construits par *Dicho* pour chacune de ces feuilles sont alors ajoutés aux codages respectifs des k noeuds. Il s'agit toujours d'équilibrer pour économiser des couleurs.
   - Noeud Premier : pour l'instant nous n'avons pas trouvé d'autre choix que de construire le graphe de conflit $G_{simple}(P)$ de l'ordre premier associé au noeud pour construire une simple coloration de cet ordre. On utilise alors la proposition 5.22 pour construire un codage à partir de cette simple coloration et des codages correspondants aux fils du noeuds.

Remarque 5.70. *Dans le cas des arbres, vu leur décomposition modulaire, cette heuristique renvoie le même codage que l'application directe de l'algorithme Dicho.*

Il reste malgré des questions concernant l'utilisation de la décomposition modulaire.

Question 6. *Peut-on améliorer la borne de la proposition 5.22 ? Est-ce que la décomposition modulaire est la plus appropriée pour les codages bit-vecteur ? Est-ce que d'autres décompositions seraient intéssantes à appliquer ?*

5.9.3 Analyse de performances

Nous avons comparé les tailles de codage obtenues avec différentes heuristiques sur un jeu de tests extrait de l'article de Vitek *et al* [KVH97]. Ces benchmarks peuvent être récupérés à l'adresse suivante :

http ://www.complang.tuwien.ac.at/~andi/typecheck/

ainsi que les programmes de codage de Vitek *et al.* Ce sont des exemples de bibliothèques de classes issues de langages orientés objets comme Smalltalk ou C++.

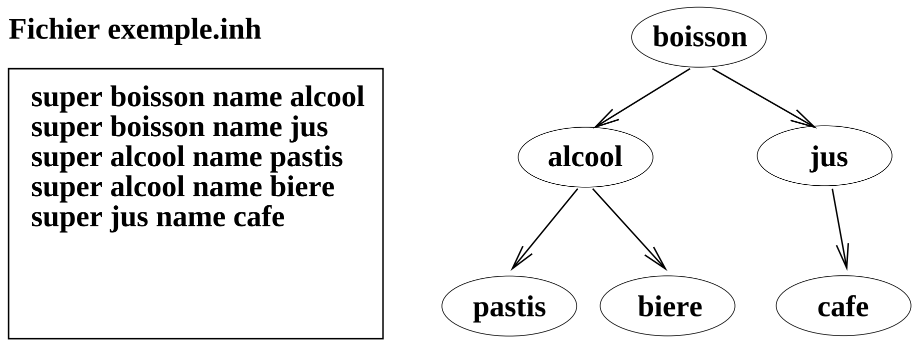


FIG. 5.31 – Exemple de syntaxe .inh avec la hiérarchie correspondante.

L'heuristique proposée dans la section précédente est en cours d'expérimentation. Nous souhaitons préciser les heuristiques de coloration qui nous semblent les meilleures pour coder le graphe de conflit des ordres premiers. Nous souhaitons aussi comparer cette heuristique utilisant la décomposition modulaire avec des heuristiques de colorations du graphe de conflit critique de l'ordre initial, censé permettre d'atteindre la 2-dimension, si bien sûr les temps d'exécution restent raisonnables.

L'implantation de nos heuristiques a été écrite en C++ (les sources sont disponibles à l'adresse *http ://www.lirmm.fr/~thierry/bitvector.html*). Les programmes prennent en entrée les hiérarchies sous le format **.inh** - le format des benchmarks. Une hiérarchie sous ce format est stockée dans un fichier où les classes sont désignées par leur nom et chaque ligne est de la forme **super class1 name class2**, ce qui signifie que **class2** est un descendant de **class1**. En termes d'ordre et donc ensuite de codage, arbitrairement on a choisi de dire que cela signifie que **class1** est **plus petit** que **class2**. Exceptionnellement, en accord avec certaines représentations pour les langages orientés objets, on a choisi dans cette section de représenter les ordres **"du haut vers le bas"**, autrement dit les plus petits éléments sont en haut. La figure 5.31 montre un exemple de fichier .inh avec la hiérarchie associée.

Le tableau 5.9.3 résume les tailles des codages bit-vecteur générés selon différentes heuristiques. Comme nous l'avons précisé, les expérimentations ne sont pas encore terminés sur le jeu de test choisi. Pour l'instant, seul l'algorithme *Dicho* a fourni des résultats, cela concerne donc les hiérarchies qui sont des arbres. Ce sont les seuls résultats présentés ici. Pour ce cas particulier des arbres, la colonne *Caseau* désigne la taille du codage obtenu par la simple

Hiérarchie	Classes	*Caseau*	*KVH97*	*CHNR99*	*Dicho*
Visualworks2	1956	420	50	58	32
Digitalk3	1357	325	36	52	29
NeXTStep	311	177	23	28	20
ET++	371	181	30	41	20

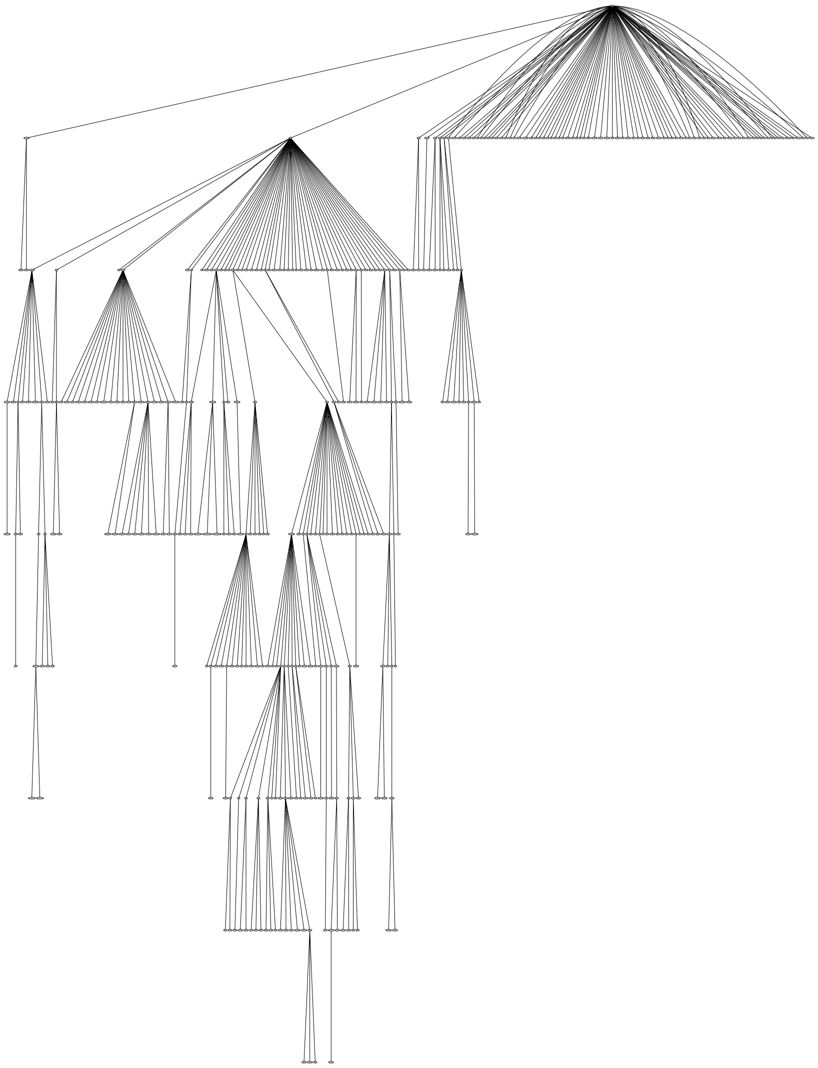
TAB. 5.2 – Tailles des codages pour le jeu de test

coloration optimale de l'arbre, la colonne *KVH97* correspond aux meilleurs codages bit-vecteur obtenus par Krall, Vitek et Horspool dans [KVH97], la colonne *CHNR99* correspond à l'heuristique proposée spécialement pour les arbres dans [CHNR99] et mentionnée dans la section 5.8 et pour finir la dernière colonne correspond à l'algorithme *Dicho*.

Les pages suivantes présentent les dessins de certaines des hiérarchies qui ont été codées. Elles illustrent la stratégie d'équilibrage utilisée pour coder les arbres : on met en vis à vis la hiérarchie initiale et l'arbre binaire de hauteur minimum que nous avons généré avec l'algorithme *Dicho*, tel que la hiérarchie initiale se plonge dedans. Ces dessins ont été obtenus avec le programme freeware *Dotty* développé à *AT&T*, qui peut être téléchargé à l'adresse :

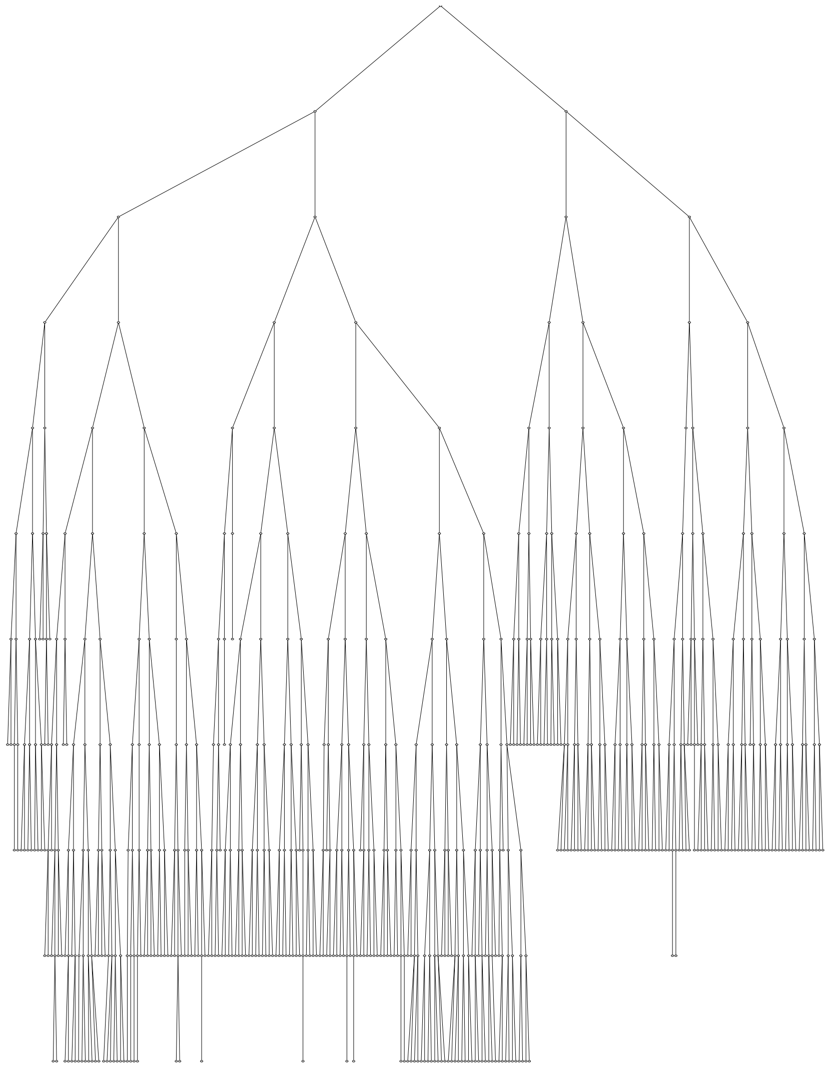
http ://www.research.att.com/sw/tools/graphviz/

ET++



Language de programmation :	C++
Nombre de classes :	371
Taille du codage bit-vecteur :	21 (nombre de bits par classe)

Arbre binaire généré par l'algorithme *Dicho* pour la hiérarchie ET++



5.10 Perspectives

Aspects dynamiques

Il est difficile de remettre à jour un codage bit-vecteur après ajout/suppression d'éléments ou de comparaisons, de manière rapide et en essayant de continuer à minimiser la taille du codage. La proposition 5.17 permet de mettre à jour un codage bit-vecteur avec 1 ou 2 bits pour tout nouveau élément. C'est malheureusement rapidement coûteux en espace.

Comme on l'a vu dans la partie I, ces difficultés ne concernent pas uniquement les codages bit-vecteurs. La plupart des méthodes connues de codage des ordres sont confontrées à la difficulté du traitement dynamique des codages.

Noter que pour la simple coloration d'un ordre P, un algorithme est proposé dans [CHNR99] pour traiter les mises à jour après ajout d'un élément maximal donné avec ses prédécesseurs immédiats. Il nécessite qu'on ait conservé une trace du graphe de conflit $G_{simple}(P)$.

Conclusion

La méthode de codage par vecteurs de bits est l'une des méthode de codage d'ordres qui a été le plus étudié d'un point de vue théorique et pratique. Ces études ont montré qu'elle fournit de bons taux de compression en espace dans de nombreux cas, ce qui permet souvent de faire tenir chaque vecteur de bits sur peu de mots mémoires et les tests de comparaisons sont donc "presque" en temps constant.

Cette méthode de codage, comme d'autres méthodes à base de plongements dans des treillis, est bien adaptée pour coder les treillis sur lesquels elle obtient ses meilleurs taux de compression (puisque pour tout ordre P, son complété de MacNeille $DM(P)$ vérifie $dim_2(DM(P)) = dim_2(P)$ alors que souvent $|DM(P)| \gg |P|$). Elle permet aussi de construire des codages pour traiter efficacement les opérations de bornes supérieures ou bien inférieures. De plus, il semble plus facile d'élaborer des heuristiques efficaces de génération de codage pour la *2-dimension* que pour d'autres méthodes à base de plongements dans des treillis comme la méthode correspondant à la *dimension* des ordres.

Il faut reconnaitre cependant que ces heuristiques de génération sont chères en temps et en espace, comparées à d'autres méthodes connues de codage d'ordres. C'est pourquoi cette méthode ne semble pas forcément la plus adaptée si le temps de génération des codages est prépondérant.

Parmi les questions soulevées dans cette partie, certaines méritent peut être une attention plus particulière :

- Résoudre la question de la complexité de la 2-dimension sur la classe des arbres apporterait des indications sur les classes d'ordres qui se comportent bien pour le calcul de la 2-dimension ? Cela permettrait peut être de proposer de nouvelles heuristiques générales de génération si le problème s'avèrait polynomial, et sinon ce serait un exemple intéressant de problème $\mathcal{NP}$-complet sur les arbres.
- Est-ce qu'il est possible pour un ordre P de relier $dim_2(P)$ à des paramètres comme $D(P)$ ou $\Delta(P)$ (présentés dans la section 5.7) ? Cela permettrait de "mesurer" un peu mieux la compression en espace possible avec cette méthode de codage.
- Proposer des algorithmes de mises à jour efficaces de ces codages dans un contexte dynamique serait bien sûr très utile.

Troisième partie

Quelques nouvelles propriétés des treillis distributifs et des extensions d'un ordre

Chapitre 6

Extensions d'un ordre et treillis de ses idéaux

Ce travail a été effectué avec Michel Morvan à partir d'une idée de Philippe Baldy et Michel Morvan. Il a consisté à généraliser une bijection connue entre les extensions linéaires d'un ordre et les chaines maximales de son treillis des idéaux établie par Bonnet et Pouzet. Il s'intègre dans une étude plus générale des extensions d'un ordre et souligne une fois de plus le rôle important du treillis des idéaux d'un ordre.

Les sections suivantes présentent la généralisation du résultat de Bonnet et Pouzet, ainsi que certaines bijections qui peuvent en être déduites.

6.1 Généralisation de la bijection de Bonnet-Pouzet

L'étude des extensions d'un ordre a fait l'objet de nombreux travaux au cours des dernières décénies. A titre d'exemples, les travaux suivants traitent de certaines classes d'extensions : extensions linéaires (Bonnet et Pouzet [BP69]), extensions intervallaires minimales (Habib et al [HMPR91]), extensions en ordres faibles (Bertet et al [BGM97, Ber99]), extensions série-parallèle (Möhring et Radermacher [MR89]) ou certains cas d'extensions intervallaires (Felsner et al [FGM99]). La structure de l'ensemble de toutes les extensions d'un ordre fixé a aussi été étudiée sous plusieurs angles dans [PRRZ95] et [PR98].

Un résultat classique de la théorie des ordres est une bijection entre les

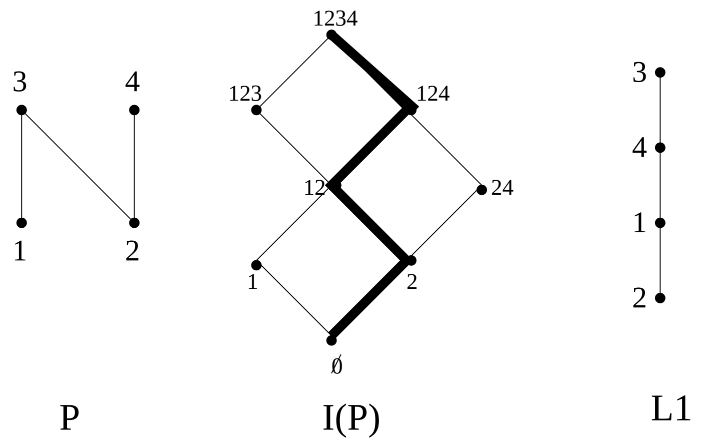


FIG. 6.1 – Un ordre P, son treillis $I(P)$, une chaine maximale de $I(P)$ (en gras), l'extension linéaire de P correspondante.

extensions d'un ordre et les chaines maximales de son treillis des idéaux. Présenté par Bonnet et Pouzet dans [BP69], ce résultat a été découvert indépendamment par Stanley dans [Sta71] et [Sta72]. Une lecture attentive de [BP69] montre que le principal résultat qui va être exposé dans cette section (une bijection pour toutes les extensions d'un ordre) est une généralisation naturelle du théorème présenté dans cet article.

Ce théorème de Bonnet et Pouzet qui relie les extensions linéaires d'un ordre avec les chaines maximales (pour l'inclusion) de son treillis des idéaux.

Théorème 6.1. *Soit $P = (X, \leq_P)$ un ordre (fini ou infini), alors l'ensemble des extensions linéaires de P est en bijection avec l'ensemble des chaines maximales (pour l'inclusion) de son treillis des idéaux $I(P)$. Plus précisément, si L est une extension linéaire de P, alors $I(L)$ est une chaine maximale de $I(P)$. Inversement, pour toute chaine maximale C de $I(P)$, il existe une unique extension linéaire L de P telle que $C = I(L)$.*

La figure 6.1 montre l'extension linéaire assocée à une chaine maximale de $I(P)$ (en gras). Dans le cas fini, si la chaine maximale C de $I(P)$ est $C = \{I_0 \subset I_1 \subset \ldots \subset I_n\}$, alors l'extension linéaire qui lui est associée est $L = \{I_1 \backslash I_0 \leq_P I_2 \backslash I_1 \leq_P \ldots I_n \backslash I_{n-1}\}$ ($|I_i \backslash I_{i-1}| = 1$ pour tout i, $1 \leq i \leq n$).

Ce théorème peut être étendu en une bijection sur l'ensemble de toutes les extensions d'un ordre de la manière suivante. On introduit une nouvelle définition : soit P un ordre, un sous-treillis T de $I(P)$ est dit ***sous-treillis à chaines maximales*** si toute chaine maximale (pour l'inclusion) de T est une chaine maximale de $I(P)$.

Théorème 6.2. *Soit $P = (X, \leq_P)$ un ordre (fini ou infini), alors l'ensemble des extensions de P est en bijection avec l'ensemble des sous-treillis à chaines maximales de $I(P)$.*

Preuve. Nous allons prouver que l'application I qui associe à un ordre q son treillis des idéaux $I(Q)$ est une bijection entre les extensions de P et les sous-treillis à chaines maximales de $I(P)$.

D'abord, l'image d'une extension Q de P est un sous-treillis à chaines maximales de $I(P)$: si A est un idéal de Q, alors A est un idéal de P (comme Q est une extension de P), et $I(Q)$ est un sous-treillis de $I(P)$ car la borne supérieure est toujours donnée par l'union et la borne inférieure par l'intersection. Maintenant considérons une chaine maximale C de $I(Q)$. Grâce au théorème 6.1, on peut associer à C une extension linéaire de P telle que $C = I(L)$. Alors L est une extension de Q et Q est une extension de P, donc L est une extension linéaire de P et $C = I(L)$ est une chaine maximale de $I(P)$.

D'après le théorème de représentation de Birkhoff qui établit une bijection entre les ordres et les treillis distributifs [Bir67], on sait que I est injective. On sait aussi que toute chaine maximale de $I(P)$ correspond à une extension linéaire de P. Soit T un sous-treillis à chaines maximales de $I(P)$. Considérons l'ensemble $\mathcal{C}(T)$ de toutes les chaines maximales de T. Ce sont toutes des chaines maximales de $I(P)$. A chaque $C \in \mathcal{C}(T)$, on associe L_C l'extension linéaire correspondante de P telle que $C = I(L)$, d'après le théorème 6.1. Soit $Q = (X, \leq_Q)$ l'ordre qui est l'intersection de toutes ces extensions L_C de P, pour $C \in \mathcal{C}(T)$. Alors Q est une extension de P et nous allons montrer que $I(Q) = T$, c'est à dire que les ensembles $I(Q)$ et T sont identiques, puisqu'ils sont tout deux ordonnés par inclusion.

Soit $x \in X$, nous allons montrer que $\downarrow_Q x \in T$. On en déduira que les éléments de $I(Q)$ sont des éléments de T. Soit L une extension linéaire de P utilisée dans la définition de Q. D'abord $\downarrow_L x \in T$ par construction de L qui est associée à une chaine maximale de T. Alors par définition de Q, on peut dire que $\downarrow_Q x$ est l'intersection des idéaux $\downarrow_L x$ pour toutes les extensions linéaires L utilisées dans la définition de Q. Par conséquent, $\downarrow_Q x \in T$ (car T est stable pour l'intersection). Maintenant soit A un idéal de Q, A est l'union des $\downarrow_Q x$ pour tous les éléments x de A. Donc $A \in T$ (car T est stable pour l'union). Au final, $I(Q) \subseteq T$.

Dans l'autre sens, soit A un élément de T. Montrons que $A \in I(Q)$. La

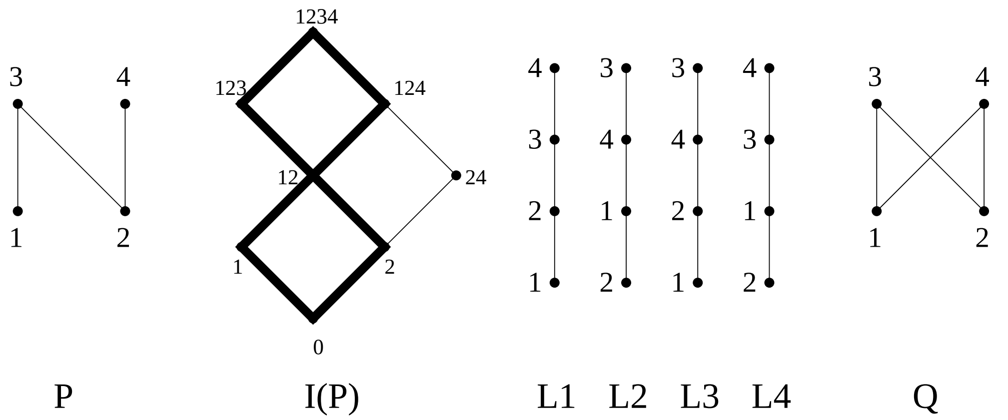


FIG. 6.2 – Un ordre P, son treillis $I(P)$, un sous-treillis de hauteur $|P|$ en gras, les extensions linéaires de P correspondantes, l'extension résultante Q.

définition de Q est équivalente à dire que $x \leq_Q y$ si et seulement si pour toute chaine maximale C et tout élément B de cette chaine, $y \in B$ implique $x \in B$. Ceci implique que A est un idéal de Q : on vérifie que $y \in A$ et $x \leq_Q y$ implique $x \in A$. Ainsi $T \subseteq I(Q)$. ■

Dans le cas fini, on obtient le corollaire immédiat.

Corollaire 6.3. *Si P est fini, l'ensemble des extensions de P est en bijection avec l'ensemble des sous-treillis de $I(P)$ de hauteur $|P|$.*

Preuve. Si P est fini, une chaine de $I(P)$ est maximale si et seulement si sa hauteur est $|P|$. Soit T un sous-treillis de $I(P)$ de hauteur $|P|$. Comme T est un sous-treillis de $I(P)$ et que $I(P)$ est distributif, T est aussi distributif et toutes les chaines maximales de T ont la même hauteur, autrement dit $|P|$ (voir par exemple [Bir67] pour cette propriété). ■

La figure 6.2 illustre le théorème 6.2 : l'extension Q de P est celle qui est associé au sous-treillis de $I(P)$ en gras.

6.2 Caractérisation d'extensions grâce au treillis des idéaux

Nombreuses classes usuelles d'ordres sont définie par structures interdites (par exemple les ordres totaux, les ordres faibles, les ordres d'intervalles, les ordres série-parallèle, les ordres N-free, les semiordres ... voir [] pour une synthèse). L'objectif de cette section est d'étudier les possibilités de caractérisation de l'ensemble des extensions appartenant à ces classes en termes de sous-treillis du treillis des idéaux ne contenant pas certaines sous-structures (par exemple ne contenant pas l'image de la structure interdite initiale). Nous allons voir qu'on peut donner de telles caractérisations sous quelques hypothèses supplémentaires.

Soit $P = (X, \leq_P)$ un ensemble ordonné. Un sous-ensemble S de X est dit ***convexe*** si pour tout couple d'éléments $x, y \in S$, pour tout $z \in X$, $x \leq_P z \leq_P y$ implique $z \in S$. L'ordre considéré sur ce convexe est l'ordre induit par P. Pour tout $a, b \in X$, $[a, b] = \{x \in X \mid a \leq_P x \leq_P b\}$ est appelé un ***intervalle*** de P, c'est un convexe de P. On peut maintenant donner le résultat suivant :

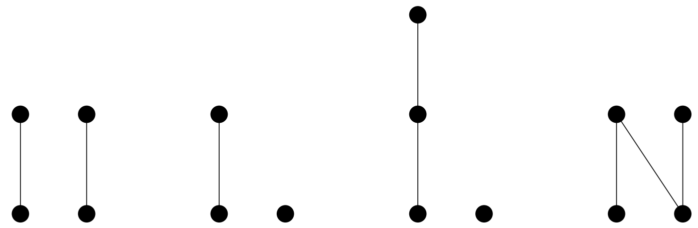
Théorème 6.4. *Soient $P = (X, \leq_P)$ et $S = (Y, \leq_S)$. Alors P contient un convexe isomorphe à S si et seulement si $I(P)$ contient un convexe isomorphe à $I(S)$.*

Preuve. Soit K un convexe de P isomorphe à S. Soit $\downarrow K$ l'idéal $\{x \in X \mid \exists z \in K,\ x \leq_P y \text{ and } x \notin K\}$. Alors l'ensemble des idéaux $\{A \in I(P) | A = \downarrow K \cup \bigcup_{x \in K'} \downarrow_P x,\ \text{pour } K' \subseteq K\}$ est isomorphe à $I(K)$ (l'isomorphisme est induit par l'application $\downarrow_P x \to \downarrow_P x \cup \downarrow K$ qui est injective pour $x \in K$), et par conséquent est isomorphe à $I(S)$. C'est un convexe de $I(P)$ du fait que K est un convexe de P.

Maintenant, soit $\mathcal{K}$ un convexe de $I(P)$ isomorphe à $I(S)$. Comme $I(S)$ a un minimum et un maximum, $\mathcal{K} = [A_{min}, A_{max}]$ où A_{min} et A_{max} sont deux idéaux de $I(P)$. Considérons $K = A_{max} \backslash A_{min}$. Alors K est un convexe de P : si $x, y \in K$ et $x \leq_P z \leq_P y$, alors $y \in A_{max}$ implique $z \in A_{max}$, et $z \notin A_{min}$ sinon on aurait $x \in A_{min}$, donc au final $z \in K$. Alors $\mathcal{K} = \{A \in I(P) | A = A_{min} \cup \bigcup_{x \in K'} \downarrow_P x,\ \text{pour } K' \subseteq K\}$, qui est isomorphe à $I(K)$ (l'isomorphisme est induit par l'application $\downarrow_K x \to \downarrow_P x \cup A_{min}$ qui est injective pour $x \in K$). Donc $I(S)$ et $I(K)$ sont isomorphe, ce qui implique que S et K sont isomorphes. ■

Pour un ensemble ordonné fixé, ce théorème fournit une bijection entre les extensions avec des structures interdites et les sous-treillis d'idéaux avec les structures correspondantes interdites aussi. Soit $\mathcal{S}$ un ensemble d'ordres, on dira qu'un ordre est ***sans $\mathcal{S}$-convexe*** s'il ne contient aucun convexe isomorphe à un ordre de $\mathcal{S}$.

Corollaire 6.5. *Soit $P = (x, \leq_P)$ un ordre et $\mathcal{S}$ un ensemble d'ordres. Les extensions de P sans $\mathcal{S}$-convexe sont en bijection avec les sous-treillis à*



$2 \dot{\cup} 2$ $2 \dot{\cup} 1$ $3 \dot{\cup} 1$ N

FIG. 6.3 – Des sous-ordres interdits.

chaines-maximales de $I(P)$ sans $I(\mathcal{S})$-convexe, où $I(\mathcal{S})$ désigne l'ensemble $\{I(S)|S \in \mathcal{S}\}$.

Dans le cas des ordres finis, les classes usuelles mentionées au début de la section précédente admettent toutes une caractérisation par sous-structures interdites. Cependant seules certaines sont définies par convexe interdit, et donc concernées par le corollaire précédent.

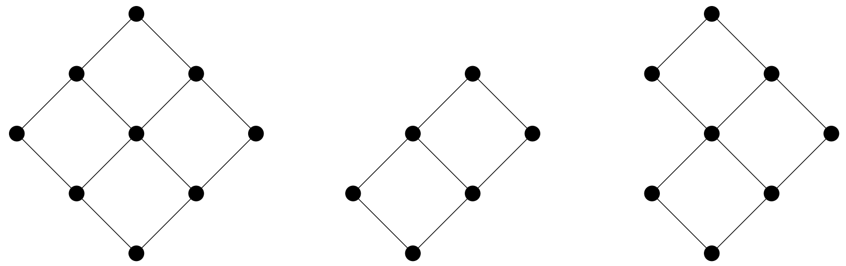
A titre d'exemple, nous donnons maintenant les définitions de trois classes d'ordres : les ordres faibles, les ordres d'intervalles et les ordres N-free. Des références sur ces classes peuvent être trouvée dans [Moh89].

Un ***ordre faible*** $P = (X, \leq_P)$ est un ordre tel que pour tout $x, y \in X$, si x et y sont incomparables, alors $Pred(x) = Pred(y)$. La caractérisation suivante est immédiate : un ordre fini est un ordre faible si et seulement si sa réduction transitive ne contient pas de sous-graphe isomorphe à l'ordre $2 \uplus 1$ (voir la figure 6.3).

Soit X un ensemble fini et $(I_x)_{x \in X}$ un ensemble d'intervalles $I_x = [g_x, d_x]$ de la ligne des réels. On peut définir un ordre strict $<_P$ sur X par $x <_P y$ si $d_x < g_y$. On rappelle qu'un ordre qui peut être obtenu de cette manière est appelé ***ordre d'intervalles***. Cette classe d'ordres admet aussi une caractérisation par sous-graphe interdit dans la réduction transitive : un ordre fini est un ordre d'intervalles si et seulement si sa réduction transitive ne contient pas de sous-graphe isomorphe à l'ordre $2 \uplus 2$ (voir figure 6.3).

Nous introduisons aussi les ordres N-free qui ont des applications variées et plusieurs caractérisations. Nous donnons juste celle par structure interdite. Un ordre fini est N-free si et seulement si sa réduction transitive ne contient pas de sous-graphe isomorphe à l'ordre N (voir la figure 6.3).

Dans les trois cas, comme les structures interdites sont des ordres de



I($2 \dot\cup 2$) I($2 \dot\cup 1$) I(N)

FIG. 6.4 – Des sous-treillis interdits.

hauteur ≤ 1, il est équivalent d'interdire ces ordres en tant que convexes de l'ordre. Le corollaire 6.5 s'applique donc :

Proposition 6.6. *Soit $P = (X, \leq_P)$ un ordre fini. Alors ses extensions de type ordre faible (resp. ordre d'intervalles, ordre N-free) sont en bijection avec les sous-treillis à chaines-maximales de $I(P)$ ne contenant pas de convexe isomorphe à $I(2 \uplus 1)$ (resp. $I(2 \uplus 2)$, $I(N)$).*

La figure 6.4 montre ces convexes interdits. Il faut aussi remarquer que les ordres totaux peuvent aussi être définis par ordre convexe interdit : il ne contiennent aucun convexe isomorphe à deux éléments incomparable (ordre noté $1 \uplus 1$). Cela signifie que les extensions linéaires d'un ordre P sont en bijection avec les sous-treillis à chaines-maximales de $I(P)$ ne contenant pas de convexe isomorphe à $I(1 \uplus 1)$, autrement les chaines maximales de $I(P)$. Avec le théorème 6.4, le théorème 6.1 sur les extensions linéaires peut être vue comme une conséquence du théorème 6.2 général.

D'autres classes d'extensions, comme les extensions série-parallèles (ordres ne contenant pas de sous-ordre isomorphe à l'ordre N), les extensions en treillis, les exetensions en semi-ordre (avec les sous-ordres interdits $2 \uplus 2$ et $3 \uplus 1$, voir la figure 6.3) n'ont pas de telles caractérisations. Les classes d'ordres correspondantes ne peuvent pas être définies par un ensemble fini de convexes interdits, aussi les théorèmes précédents ne s'appliquent pas.

Ces résultats confirment le fait que beaucoup de propriétés peuvent aussi bien se lire sur un ordre que sur son treillis des idéaux, ce qui offre la possibilité de choisir de travailler sur l'un ou sur l'autre.

Chapitre 7

Retournement de treillis

Ce chapitre étudie une nouvelle opération sur les treillis, qui se comporte bien en particulier sur les treillis distributifs. A l'origine, cette opération consistant à changer localement d'orientation provient de la réinterprétation en terme d'ordres et treillis de certaines transformations dans les systèmes dynamiques, comme les transformations de pavages par des dominos. En s'abstrayant du contexte des systèmes dynamiques, on a pu définir uniquement en termes d'ordre cette opération pour pouvoir mieux l'étudier ensuite. Elle offre un nouveau regard sur certains treillis, en mettant en valeur des propriétés "géométriques" intéressantes.

7.1 Définition et propriétés

Soit $T = (X, \leq_T)$ un treillis. On note $G(T)$ le graphe non orienté obtenu à partir de la réduction transitive de T en remplaçant chaque arc (orienté) par une arête (non orientée). Maintenant nous introduisons la nouvelles opération sur les treillis consistant à faire "basculer" certaines parties du treillis.

Définition 7.1. *Soient $T = (X, \leq_T)$ un treillis et x, y deux éléments de X. On définit T_{xy} comme l'ensemble des sommets de $G(T)$ qui appartiennent aux chaines de longueur minimum commençant en x et terminant en y (aussi appelés géodésiques de x à y).*

L'ensemble T_{xy} est ordonné de la manière suivante : soient v, w deux sommets de T_{xy}, on dit $v \leq_{T_{xy}} w$ s'il existe une géodésique dans $G(T)$ allant de x à y, passant d'abord par v puis par w.

On vérifie facilement que cette définition donne bien un ordre sur un sous-ensemble de L où x devient un minimum et y un maximum. La figure 7.1 présente un treillis T et les ordres obtenus en considérant les couples T_{xy} et T_{st} (dans chaque cas, les sommets en blancs sont les sommets des chemins de longueur minimum entre le sommet de départ et le sommet d'arrivée). Cette opération consistant à associer à T, x et y l'ordre T_{xy} est appelée ***opération de retournement***.

L'un des points que nous allons développer (par exemple pour les pavages par des dominos) concerne la stabilité de certaines classes de treillis pour l'opération de retournement. Nous utiliserons les deux définitions suivantes.

Définition 7.2. *Un treillis T est dit* ***retournable*** *si pour tout couple de sommets x et y, l'ordre T_{xy} est un treillis.*

Définition 7.3. *Une classe Ψ de treillis est dite* ***stable pour l'opération de retournement*** *si pour tout treillis T de Ψ et tout couple de sommets x et y dans T, l'ordre T_{xy} est un treillis qui appartient à Ψ.*

Tous les treillis ne sont pas retournables. La figure 7.2 montre un exemple de treillis tel que l'ordre T_{xy} n'est pas un treillis.

Remarquer que conséquence de la définition 7.1, les ordres T_{xy} sont toujours rangés. Cependant les treillis rangés ne sont pas toujours retournables comme on peut le voir sur la figure 7.3, deux ellipses ont été dessinée pour donner un point de vue 3D et mettre en valeur que les sommets importants sont sur deux niveaux.

On peut aussi définir un autre ordre à partir du graphe $G(T)$ d'un treillis T. Cet ordre est obtenu en choisissant un élément x de T et en "suspendant" le treillis par x.

Définition 7.4. *Soit T un treillis et x un élément de T. On définit l'ordre T_x sur tous les éléments de T de la manière suivante : soient v,w deux éléments de T, on dit que $v \leq_{T_x} w$ s'il existe une géodésique dans $G(T)$ allant de w à x et passant par v.*

Les propriétés de cet ordre T_x sont bien sûr fortement liées à celles des ordres T_{xy}. On a par exemple l'implication suivante, où on parle de inf-1/2-treillis. Un ordre T est appelé un ***inf-1/2-treillis*** si pour tout couple d'éléments x et y de T, il existe une borne supérieure $x \vee_T y$.

Proposition 7.5. *Soit T un treillis. Si pour tout élément x de T, l'ordre T_x est un inf-1/2-treillis, alors pour tout couple déléments x et y de T, l'ordre T_{xy} est un treillis rangé.*

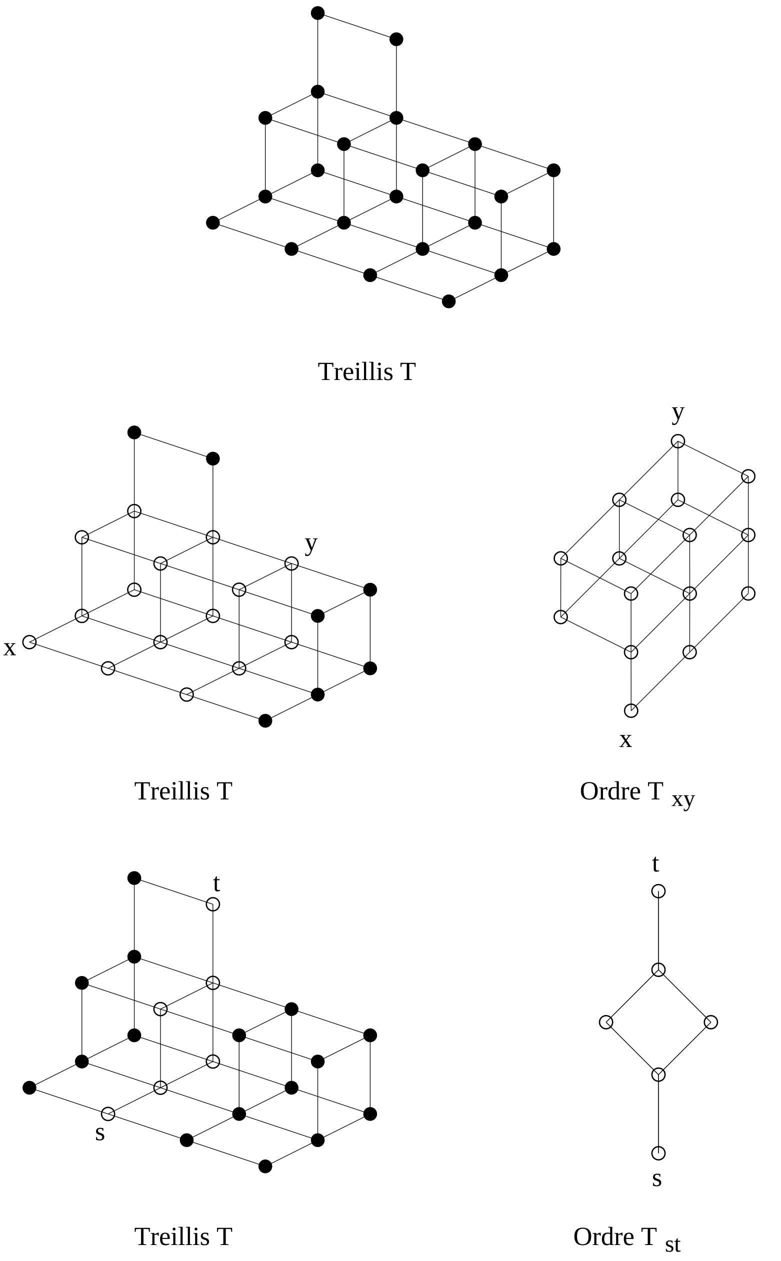


FIG. 7.1 – Exemples de “retournements”

Cette proposition est immédiate à partir des définitions. Par contre la

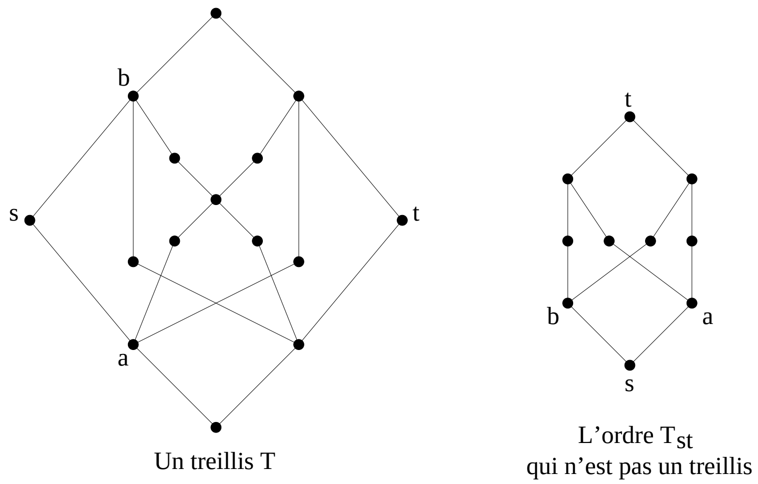


FIG. 7.2 – Un exemple de treillis non retournable.

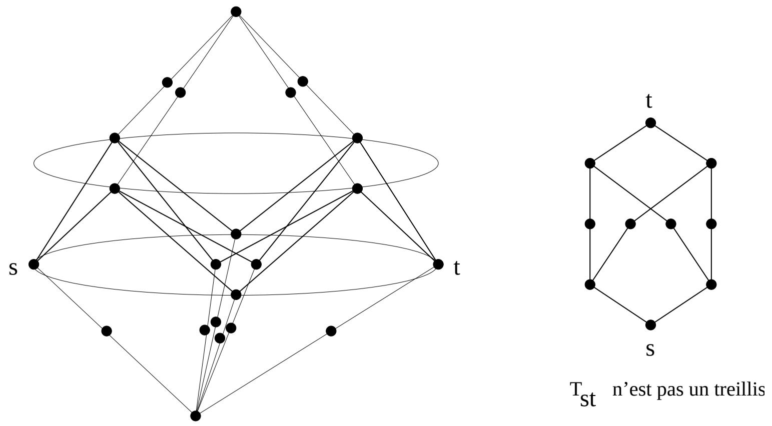


FIG. 7.3 – Un exemple de treillis rangé non retournable

réciproque est fausse, comme on peut le voir sur le contre-exemple de la figure 7.4. Il semble qu'on puisse ajouter des hypothèses pour obtenir une réciproque vraie, mais pour l'instant nous avons juste une conjecture.

Conjecture 6. *Soit T un treillis. Si pour tout couple d'éléments x et y de T, l'ordre T_{xy} est un treillis distributif, alors pour tout élément x de T, l'ordre T_x est un inf-1/2-treillis.*

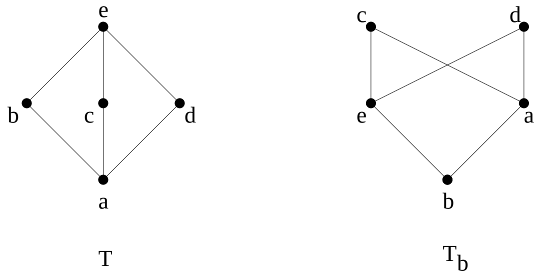


FIG. 7.4 – Un contre-exemple à la réciproque de la proposition 7.5

7.2 Quelques exemples de treillis retournables et de classes de treillis stables

7.2.1 Treillis distributifs

Proposition 7.6. *La classe des treillis distributifs est stable pour l'opération de retournement. Autrement dit, soit T un treillis distributif, A et B deux éléments de T, alors T_{AB} est un treillis distributif.*

Preuve. Grâce au théorème de représentation de Birkhoff [Bir67], le treillis distributif T est isomorphe au treillis des idéaux de $I(P)$ d'un ordre P. Nous pouvons donc considérer que A et B sont deux idéaux de P. Le graphe $G(T)$ est parfaitement caractérisé par la propriété suivante : on sait que dans la réduction transitive de $I(P)$ il existe un arc de I vers I' si et seulement si $I \subseteq I'$ et $|I' \backslash I| = 1$.

La première conséquence est que la distance entre A et B dans $G(T)$ est supérieure ou égale à $|(A\backslash B) \bigcup (B\backslash A)|$. Un chemin de A vers B consiste à supprimer de l'ensemble A les éléments de $A\backslash B$ et aussi ajouter les éléments de $B\backslash A$ pour atteindre l'ensemble B. La suite de suppressions et d'ajouts est contrainte par l'ordre P si l'on veut garder des idéaux le long du chemin. Plus formellement, on note Q le sous-ordre induit par P sur son sous-ensemble $A\backslash B$ et par R le sous-ordre induit par P sur son sous-ensemble $B\backslash A$. Nous allons prouver que l'ordre T_{AB} est isomorphe à $(Q^d \bigcup R)$. La figure 7.5 représente la configuration de ces sous-ensembles de P et le schéma d'un chemin allant de A à B dans $I(P)$.

Chaque élément de $I(Q^d \bigcup R)$ est l'union disjointe $I \bigcup J$ d'un idéal I de Q^d et d'un idéal J de R. Soit ϕ l'application qui associe à chaque élément $I \bigcup J$ de $I(Q^d \bigcup R)$ l'ensemble $(A\backslash I) \bigcup J$.

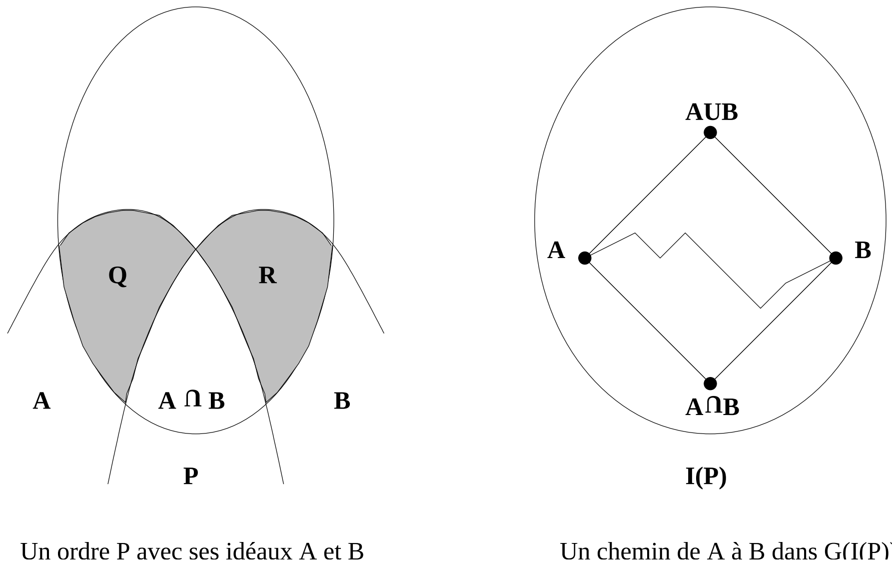


FIG. 7.5 – Configuration dans le cas d'un treillis distributif $I(P)$

D'abord on peut voir que cette application prend ses valeurs dans l'ensemble L_{AB}. Il suffit de remarquer que si J est un idéal de R, alors $J\bigcup(A\bigcap B)$ est un idéal de P. Si I est un idéal de Q^d, alors $(A\backslash I)$ est un idéal de P aussi, et au final $(A\backslash I)\bigcup J$ est un idéal de P. La définition de $G(T)$ implique qu'il existe dans $G(T)$ un chemin de longueur $|I|$ de A à $A\backslash I$ (en supprimer un par un les éléments maximaux de I), un chemin de longueur $|J|$ de $A\backslash I$ à $(A\backslash I)\bigcup J$ (en transférant un par un les élḿents minimaux de J de J vers l'idéal), un chemin de longueur $|(A\backslash I)\backslash(A\bigcap B)|$ de $(A\backslash I)\bigcup J$ vers $(A\bigcap B)\bigcup J$ et un chemin de longueur $|(B\backslash J)\backslash(A\bigcap B)|$ de $(A\bigcap B)\bigcup J$ vers B. Cela siginie qu'il existe un chemin de longueur $|(A\backslash B)\bigcup(B\backslash A)|$ de A vers B qui passe par $(A\backslash I)\bigcup J$. C'est la distance entre A et B dans $G(T)$, donc $(A\backslash I)\bigcup J$ appartient à L_{AB}.

L'application ϕ est clairement injective, car A et R sont des ensembles disjoints. Pour prouver que ϕ est surjective, nous aloons prouver que T_{AB} est inclus dans l'ensemble $\{X \in I(P) \mid A\bigcap B \subseteq X \subseteq A\bigcup B\}$ qui est un intervalle de $I(P)$ noté $[A\bigcap B, A\bigcup B]$. Supposons qu'il existe un chemin de longueur minimum de A vers B qui passe par X où $X \notin [A\bigcap B, A\bigcup B]$. Comme on l'a déjà vu, la distance entre A et X dans $G(T)$ est égale à $|(A\backslash X)\bigcup(X\backslash A)|$ et la distance entre X et B est égale à $|(X\backslash B)\bigcup(B\backslash X)|$. On peut prouver facilement que si $X \notin [A \bigcap B, A \bigcup B]$ alors $|(A \backslash X) \bigcup (X \backslash A)| + |(X \backslash B) \bigcup (B \backslash X)| > |(A \backslash B) \bigcup (B \backslash A)|$. Cela signifie que la longueur du chemin passant par X n'est pas la distance entre A et B. C'est en contradiction avec notre hypothèse, donc $X \in [A \bigcap B, A \bigcup B]$. En termes d'ajouts et de suppressions d'éléments le long du chemin, cela signifie que si $X \notin [A \bigcap B, A \bigcup B]$, on a par exemple un élément $x \in X$ such that $x \notin A$ and $x \notin B$. Pour passer par X, il faudrait ajouter x à un moment et ensuite supprimer x. Cela rallonge le chemin de A vers B, comparé aux chemins les plus courts.

Si $X \in I(P)$ et $A \bigcap B \subseteq X \subseteq A \bigcup B$, alors $(A \backslash X) \bigcup (X \backslash A)$ est un idéal de $Q^d \bigcup R$ et $\phi((A \backslash X) \bigcup (X \backslash A)) = X$. Comme T_{AB} est inclus dans l'ensemble $[A \bigcap B, A \bigcup B]$, l'application ϕ est surjective et les ensembles T_{AB} et $\{X \in I(P) \mid A \bigcap B \subseteq X \subseteq A \bigcup B\}$ sont égaux.

Concernant l'orientation des arcs pour T_{AB}, les définitions de $G(T)$ et T_{AB} impliquent que un ensemble X précède un ensemble Y sur un chemin de de longueur minimum entre A et B si et seulement si $X \subseteq Y$ et $Y \backslash X = \{y\}$ avec $y \in B$, ou $Y \subseteq X$ et $X \backslash Y = \{x\}$ avec $x \in A$. En remplaç X et Y avec $X' = (A \backslash X) \bigcup (X \backslash A)$ et $Y' = (A \backslash Y) \bigcup (Y \backslash A)$, c'est exactement la réduction transitive du treillis des idéaux de $Q^d \bigcup R$.

On peut conclure que l'ordre T_{AB} sur l'ensemble $[A \bigcap B, A \bigcup B]$ est isomorphe au treillis des idéaux $I(Q^d \bigcup R)$ qui est un treillis distributif. ■

La réciproque de cette propriété n'est pas vraie, il suffit de considérer le treillis N_5.

7.2.2 Treillis associés à des graphes symétriques

Un graphe non orienté G est dit ***symétrique*** si pour tout couple de sommets x et y, il existe un automorphisme α du graphe G tel que $\alpha(x) = y$ (citer p.ex. harary pour cette definition). Nous introduisons maintenant une nouvelle classe de treillis.Soit T un treillis avec pour minimum $\perp$, ce treillis T est dit ***symétrique*** si $G(T)$ est un graphe symétrique et si la rédution transitive de T est définie de la manière suivante : il existe un arc d'un sommet x vers un sommet y dans la réduction transitive de T si et seulement si il existe une arête dans $G(T)$ et $distance(\perp, y) = distance(\perp, x) + 1$ dans $G(T)$.

Proposition 7.7. *Les treillis symétriques sont retournables.*

Preuve. Soit $T = (X, \leq_T)$ un treillis symétrique de minimum $\perp$. soit a et b deux éléments de T. Il existe un automorphisme α de $G(T)$ tel que $\alpha(a) = \perp$.

Ceci implique que l'ordre T_{ab} est isomorphe à l'ordre $T_{\perp\alpha(b)}$. Maintenant la définition de l'ordre T and la définition de l'ordre sur $T_{\perp\alpha(b)}$ implique que l'ordre $T_{\perp\alpha(b)}$ est égal à l'intervalle $[\perp, \alpha(b)] = \{x \in X \mid \perp \leq_T x \leq_T \alpha(b)\}$ muni de l'ordre induit par T. Un chemin de longueur minimum de $\perp$ vers $\alpha(b)$ est un chemin où la distance à partir de $\perp$ augmente de 1 pour chaque nouveau sommet le long du chemin, donc tous ces sommets appartiennent à l'intervalle $[\perp, \alpha(b)]$. Et un sommet x tel que $\perp \leq_T x \leq_T \alpha(b)$ appartient nécessairement à un chemin de longueur $distance(\perp, \alpha(b))$ dans $G(T)$ par définition de $\leq_T$.

Pour conclure, commet un intervalle d'un treillis est toujours un sous-treillis de ce treillis, l'ordre T_{ab}, qui est isomorphe à $[\perp, \alpha(b)]$, est un treillis et T est retournable. ■

Plusieurs treillis classiques sont connus pour être symétriques : par exemple la classe des treillis de Coxeter, qui contient le permutoèdre, et des treillis construits à partir de graphes de Cayley (pour des définitions de ces objets, le lecteur est renvoyé à la thèse de N. Caspard [Cas99]).

Corollaire 7.8. *Les treillis de Coxeter sont retournables.*

Remarquer cependant que cette dernière classe de treillis n'est pas stable pour l'opération de retournement.

7.3 Perspectives

Nous n'avons pas encore réussi à caratériser les classes de treillis stables par retournement. La classe des treillis distributifs est un exemple intéressant. En fait, pour certains treillis issus de systèmes dynamiques, comme les treillis de pavages par des dominos qui sont distributifs [Rem99], l'opération de retournement peut s'interpréter comme une inversion de certaines des règles de transition entre les états du système dynamique. L'étude des systèmes dynamiques par le biais de la théorie de l'ordre, et en particulier des treillis, gagnerait à avoir des propriétés intéressantes sur l'opération de retournement qui permettraient ensuite de mieux qualifier le comportement de ces systèmes. Parmi les treillis issus des systèmes dynamiques et susceptibles d'être retournables, on compte les treillis de piles de sable [Pha99].

Quatrième partie

Elagage de graphes. Application aux graphes distance-héréditaires.

Chapitre 8

Elagage de graphes et optimisation sur les graphes distance-héréditaires

Ce chapitre est consacré à l'étude d'un processus particulier d'élagage des sommets d'un graphe. Initialement étudié pour caractériser une classe particulière de graphes, les ***graphes distance-héréditaires***, il se généralise à n'importe quel graphe. Au cours de ma thèse, deux aspects ont été abordés.

D'abord avec Jean-Marc Lanlignel, Lhouari Nourine et Olivier Raynaud du LIRMM, nous avons étudié la complexité du calcul d'une séquence d'élagage pour un graphe quelconque. La solution proposée la plus efficace est basée sur une idée de Lhouari Nourine : elle fait usage d'une structure de données proche des arbres digitaux de recherche qui trouve là un emploi original comme représentation d'un graphe. Ce travail a donnée lieu à une publication intitulée “Pruning graphs with digital search trees. Application to distance-hereditary graphs” dans la conférence STACS'2000 [LRT00].

Dans le cas des graphes distance-héréditaires, une telle séquence d'élagage caractérise complètement le graphe. C'est pourquoi avec Olivier Cogis du LIRMM, nous nous sommes intéressé à résoudre sur les graphes distance-héréditaires des problèmes classiques d'optimisation en utilisant juste en entrée ces séquences d'élagage, en évitant d'utiliser les structures de données classiques (listes d'adjacence, matrice d'adjacence). De nombreux problèmes d'optimisation avaient déjà été étudiés sur cette classe de graphes qui a une multitude de bonnes propriétés. Nous ne résolvons pas de nouveaux problèmes, mais nous proposons des algorithmes gloutons simples permettant de faire des calculs “à la volée”. Ce travail a fait l'objet d'un rapport de recherche intitulé “Greedy algorithms for optimization on distance-hereditary graphs” [CT01].

La partie concernant la généralisation du processus d'élagage à tous les graphes et les solutions algorithmiques permettant de l'implanter efficacement n'est pas traitée en détail. Seuls les idées générales et les résultats de complexité seront présentés ici. Pour plus de détails, le lecteur est renvoyé à l'article de STACS [LRT00] et surtout à la thèse de Jean-Marc Lanlignel intitulée “Autour de la décomposition en coupes” [Lan01].

8.1 Définitions et notation

Les premières définitions sur les graphes non-orientés ont déjà été présentées dans la section 1, on donne ici quelques définitions plus spécifiques aux graphes non orientés. Comme précédemment, en l'absence d'ambiguité on omettra certains indices.

Voisinage

Etant donné un graphe $G = (V, E)$ et un sommet $x \in V$, on appelle ***voisinage ouvert*** de x l'ensemble de ces voisins $\{y \in V | xy \in E\}$, il est noté $\mathcal{N}_G(x)$. On définit aussi le ***voisinage fermé*** de x par $\mathcal{N}_G(x) \cup \{x\}$, il est noté $\mathcal{N}_G[x]$.

Chaines et distance

Etant donné un graphe $G = (V, E)$ et deux sommets $x, y \in V$, la ***distance*** entre x et y dans G est la longueur minimum d'une chaine reliant x à y, elle est notée $d_G(x, y)$.

Graphes distance-héréditaires

Cette classe de graphe a été introduite par Howorka en 1977.

Définition 8.1 (Howorka [How77]). *Un graphe $G = (V, E)$ est* ***distance-héréditaire*** *si pour tout sous-graphe connexe H de G et pour tout couple de sommets x et y de H, $d_H(x, y) = d_G(x, y)$.*

Une autre manière de présenter cette définition consiste à dire que pour tout couple de sommets x et y de G toutes les chaines élémentaires sans corde allant de x à y (***géodésiques***) ont la même longueur. La propriété vérifiée par ces graphes est clairement héréditaire : tout sous-graphe d'un graphe distance-héréditaire est lui-aussi distance-héréditaire.

Cette classe de graphes a donné lieu à de très nombreuses caractérisations issues notamment du travail de Bandelt et Mulder [BM86] : caractérisation par sous-graphes interdits, propriétés des cycles, propriétés métriques ... Le lecteur est renvoyé au livre de Brandstädt, Le et Spinrad [BLS99], à [BM86] ou [HM90] pour leurs énoncés. Nous allons nous intéresser à l'une d'elle en particulier : la caractérisation par séquence d'élagage.

Séquences d'élagage

Etant donné un graphe $G = (V, E)$, un sommet $x \in V$ est un ***sommet pendant*** si $|\mathcal{N}(x)| = 1$. Deux sommets $x, y \in V$ sont des ***jumeaux*** si $\mathcal{N}(x)\backslash\{y\} = \mathcal{N}(y)\backslash\{x\}$. Si x et y sont adjacents, on dit que ce sont des ***faux jumeaux***, et s'ils ne sont pas adjacents ce sont des ***vrais jumeaux***. Dans ce chapitre, on appelera ***séquence d'élagage*** d'un graphe G une séquence $(x_1, x_2, \dots, x_k)$ de sommets de G pour tout i, $1 \leq i < k$, x_i est un sommet pendant ou a un sommet jumeau dans $G_i = G[x_{i+1}, \dots, x_k]$ le graphe induit par G sur $\{x_{i+1}, \dots, x_k\}$. Autrement dit une séquence d'élagage est une séquence de suppression des sommets telle que à chaque étape le sommet supprimé est un sommet pendant ou un sommet qui a un jumeau. La valeur k est appelée ***longueur*** de la séquence.

Théorème 8.2 (Bandelt, Mulder [BM86]). *Un graphe $G = (V, E)$ est distance-héréditaire si et seulement si il admet une séquence d'élagage de longueur n, où $n = |V|$.*

Par la suite pour un graphe distance-héréditaire une séquence d'élagage désignera toujours une séquence de longueur n où n est le nombre de sommets. On utilisera les notations suivantes pour les séquences d'élagage. On lit la séquence de la gauche vers la droite et xTy signifie que x et y sont des vrais jumeaux et que x doit être supprimé, xFy signifie que x et y sont des faux jumeaux et que x doit être supprimé, xPy signifie que x est un sommet pendant, que y est son unique voisin et que x doit être supprimé. Par exemple, $(aPb, dFb, eTc, ePc, ePb)$ est une séquence d'élagage du graphe G de la figure 8.1 ci-dessous. Remarquer que lue de la droite vers la gauche, la séquence peut être utilisée pour reconstruire le graphe (dans ce cas on parle de séquence de construction).

La caractérisation par séquence d'élagage est probablement la plus intéressante du point de vue algorithmique comme on va le voir : d'abord pour la reconnaissance des graphes distance-héréditaires (voir la section 8.2) et ensuite pour résoudre différents problèmes d'optimisation sur cette classe de graphes (voir la section 8.3)

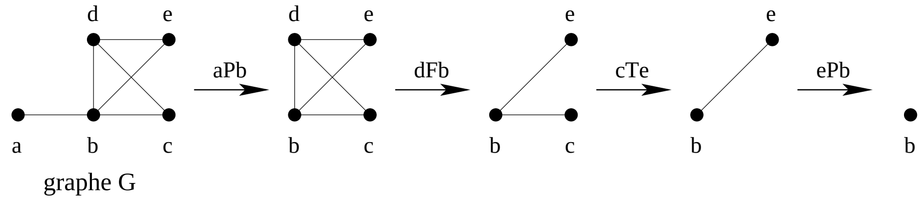


FIG. 8.1 – Un exemple de séquence d'élagage pour le graphe G

8.2 Algorithmes d'élagage

Algorithmes d'élagage

Une étude de la structure des niveaux de distance à partir d'un sommet dans les graphes distance-héréditaires effectuée dans [BM86] et retrouvée dans [HM90] a permis à Hammer et Maffray de proposer un algorithme de reconnaissance de ces graphes qui est linéaire en temps et en espace [HM90]. Etant donné un sommet du graphe en entrée, leur algorithme produit une séquence d'élagage du graphe s'il est distance-héréditaire et s'arrête en cours de route sinon. Il est basé sur un parcours en largeur pour commencer et ensuite travaille sur les niveaux de distance, il nécessite l'utilisation en sous-routine d'un algorithme de reconnaissance linéaire des cographes.

Malheureusement leur algorithme ne fonctionnait pas toujours comme cela a été remarqué par Damiand [Dam97], certains cas étant mal traités. Une version corrigé de cet algorithme a été présenté dans l'article de Damiand, Habib et Paul [DHP01], la complexité de l'algorithme corrigé restant linéaire en temps et en espace. Si le graphe en entrée est distance-héréditaire, on obtient de même en sortie une séquence d'élagage.

Ce problème d'élagage a été généralisé à un graphe quelconque grâce au théorème suivant.

Théorème 8.3. *Soit $G = (V, E)$ un graphe quelconque. Supprimer un par un les sommets qui ont un jumeau (vrai ou faux) et les sommets pendants tant qu'il en reste finit par donner un graphe unique à un isomorphisme près. La séquence de suppression est appelée **séquence d'élagage** de G et le graphe final est appelé **graphe réduit** de G, noté Reduit(G).*

Pour un graphe G donné, il existe plusieurs séquences possibles de suppression de sommets avec jumeau et sommets pendants. Le théorème consiste à prouver que quelque soit l'ordre de ces suppressions on obtient toujours le même graphe réduit (à un isomorphisme près). On peut en faire une preuve ad-hoc ou utiliser un théorème de Church-Rösser [HN97].

Plus généralement on peut définir des séquences d'élagage et la notion de graphe réduit pour des opérations de suppression qui commutent entre elles. (par exemple, pour un k fixé, un élagage consistant à supprimer les sommets avec un jumeaux et ceux de degré $\leq k$ jusqu'à ce qu'il n'en reste plus).

La reconnaissance des graphes distance-héréditaires peut alors s'écrire de la manière suivante : un graphe G est distance-héréditaire si et seulement si $Reduit(G)$ est le graphe réduit à un seul sommet. Par conséquent ce problème de reconnaissance se généralise au problème du calcul de $Reduit(G)$.

Problème 5. *Pour un graphe G donné, avec quelle complexité peut-on calculer $Reduit(G)$?*

Grâce à un algorithme d'affinage de partition [Pau98], il est possible de repérer les sommets jumeaux dans un graphe en temps et espace $\mathcal{O}(n+m)$ où n désigne le nombre de sommets du graphe et m le nombre de arêtes. En utilisant de manière répétée cette algorithme de repérage, on peut obtenir un algorithme de calcul d'une séquence d'élagage et donc du graphe réduit en temps $\mathcal{O}(n(n+m))$ et espace $\mathcal{O}(n+m)$.

En fait nous avons prouvé que ce problème peut être résolu avec des algorithmes de complexités presque linéaires en temps et en espace présentés dans [LRT00]. Nous en donnons ici les grandes lignes. Pour une description et une analyse très précises et complètes de ces algorithmes, le lecteur est renvoyé à la thèse de Jean-Marc Lanlignel intitulée "Autour de la décomposition en coupes" [Lan01].

Un principe général d'algorithme a été proposé, donnant lieu à trois variantes d'implantation dont les complexités sont résumées dans le tableau 8.2 (où n désigne le nombre de sommets du graphe initial et m son nombre d'arêtes). Pour un graphe $G=(V,E)$, l'idée consiste à réinterpréter le problème d'élagage en terme d'ensembles : à chaque sommet $x \in V$ du graphe, on associe son voisinage ouvert $\mathcal{N}(x)$ et son voisinage fermé $\mathcal{N}[x]$. Deux sommets x,y sont vrais jumeaux si et seulement si $\mathcal{N}(x)=\mathcal{N}(y)$ et faux jumeaux si et seulement si $\mathcal{N}[x]=\mathcal{N}[y]$, sachant de plus que si $x \neq y$, on n'a nécessairement $\mathcal{N}(x) \neq \mathcal{N}[y]$. Un sommet pendant $x \in V$ est un sommet tel que $|\mathcal{N}(x)|=1$. Le calcul d'une séquence d'élagage de G revient alors à être capable de détecter au fur et à mesure les ensembles égaux et ceux de cardinal 1 dans la famille d'ensembles $\mathcal{F}=\{\mathcal{N}(x),\mathcal{N}[x] \mid x \in V\}$ en mettant à jour cette famille après chaque suppression d'un sommet.

Dans cette optique, il fallait une structure de données et des algorithmes pour traiter le plus efficacement possible les opérations suivantes sur une famille $\mathcal{F}$ de sous-ensembles d'un ensemble fixé V :

1. Repérer dans $\mathcal{F}$ les ensembles égaux.
2. Supprimer un élément de V de tous les ensembles de $\mathcal{F}$.
3. Suprrimer un ensemble de $\mathcal{F}$.

La complexité à laquelle on s'intéresse est la complexité amortie puisqu'il s'agit de répéter ces opérations. La structure de données choisie a été les ***arbres lexicographiques***. Variante des arbres digitaux de recherche (aussi appelés ***tries***), cette structure de données avait déjà été utilisée dans d'autres contextes pour effectuer des opérations sur des ensembles : par exemple pour représenter les treillis distributifs [HN96], ou pour générer la fermeture par union d'une famille d'ensembles [NR99]. A notre connaissance, il s'agit de la première fois que cette structure de données est utilisée pour représenter les voisinages d'un graphe (autrement dit les listes d'adjacence d'un graphe).

Complexité en temps	$\mathcal{O}(n+m)$	$\mathcal{O}(m \log n)$	$\mathcal{O}(n^2)$
Complexité en espace	$\mathcal{O}(n^2)$	$\mathcal{O}(m \log n)$	$\mathcal{O}(n+m)$

TAB. 8.1 – Complexités des trois variantes obtenues pour le calcul de $Reduit(G)$ (chaque colonne correspond à l'un des algorithmes)

Le choix de la numérotation des sommets influe beaucoup sur la structure des arbres lexicographiques utilisés et donc sur les complexités des algorithmes. Dans certains cas, par exemple si on ne supprime que les sommets pendants et les sommets avec un vrai jumeau (resp. un faux jumeau), le calcul d'une séquence d'élagage devient linéaire en temps et en espace pour peu qu'on ait numéroté les sommets selon un ordre *Lex-BFS* [LRT00].

Nous conjecturons (voir [Lan01]) qu'il existe un moyen de calculer le graphe $Reduit(G)$ avec un algorithme linéaire en temps ***et*** en espace (par une bonne numérotation des sommets, par une meilleure gestion de l'arbre lexicographique en choisissant bien les structures de données pour le représenter, ou bien avec une toute autre approche).

Pour les graphes ayant un petit graphe réduit, il serait aussi intéressant de savoir quels problèmes peuvent se résoudre efficacement en passant par le graphe réduit, et donc quels invariants de graphes (nombre chromatique, stable maximum, clique maximum, etc) se transmettent bien entre un graphe et son graphe réduit.

Question 7. *Quels problèmes sur un graphe G peuvent être résolus efficacement à partir de la donnée d'une séquence d'élagage et de solutions à ces problèmes pour le graphe $Reduit(G)$?*

Le cas des graphes dont le graphe réduit est un singleton, c'est à dire les graphes distance-héréditaires, est étudié dans la section suivante.

8.3 Optimisation sur les graphes distance-héréditaires

Nous nous intéressons maintenant à la résolution de problèmes classiques d'optimisation pour les graphes distance-héréditaires. De nombreux auteurs se sont intéressés à cette classe de graphes du fait de ses bonnes propriétés structurelles. Dans ce qui suit, n désignera toujours le nombre de sommets du graphe initial et m son nombre d'arêtes.

La classe des graphes distance-héréditaires est une sous-classe de celle des ***brittle graphs***. Les graphes distance-héréditaires héritent des brittle graphs le fait que ce sont des ***perfectly orderable graphs*** (une classe introduite par Chvatal [Chv84]) : il existe un ***perfect order*** sur les sommets du graphe. Un perfect order d'un graphe $G = (V, E)$ est un ordre total $v_1 < v_2 < \ldots < v_n$ sur les n sommets du graphe tel que si $v_i v_j v_k v_l$ est un P_4 du graphe G (autrement dit $v_i v_i j \in E$, $v_j v_k \in E$, $v_k v_l \in E$ et $v_i v_k \notin E$, $v_i v_l \notin E$, $v_j v_l \notin E$) alors on n'a pas à la fois $v_i < v_j$ et $v_l < v_k$. Le ***nombre chromatique*** de ces graphes peut être calculé en temps et espace $\mathcal{O}(n)$ une fois que ce perfect order est donné : l'algorithme glouton de coloration suivant ce perfect order et consistant à donner à un sommet la plus petite couleur non utilisée par ses voisins fournit une coloration optimale du graphe.

Il faut remarquer que les séquences d'élagage ne sont pas nécessairement des perfect order comme on peut le voir sur l'exemple de la figure 8.2 où aucune séquence d'élagage n'est un perfect order. De plus l'algorithme de coloration glouton suivant une séquence d'élagage ne donne pas non plus nécessairement une coloration optimale comme on peut le voir sur la même figure, où aucune séquence d'élagage ne fournit un ordre de coloration gloutonne donnant une coloration optimale. Prendre par exemple la séquence de suppressions $(b, a, c, g, e, f, d, h, l, j, k)$, la coloration gloutonne utilise 4 couleurs alors que le nombre chromatique du graphe vaut 3.

Les graphes distance-héréditaires ont d'intéressantes propriétés sur les

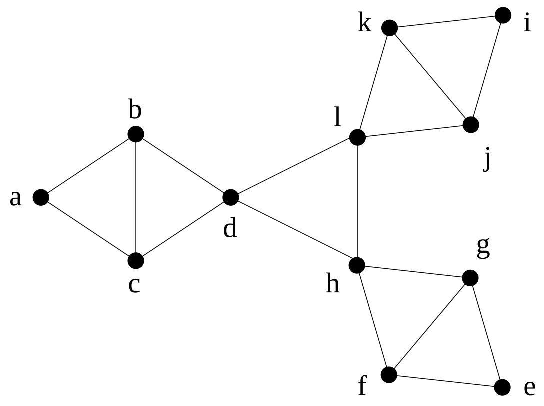


FIG. 8.2 – Auncune séquence d'élagage de ce graphe n'est un ordre de coloration gloutonne optimale

distances et des caractères comme le diamètre [CDHP98, DN00], la distance moyenne [Oel94] ou le sommet central [Dra95] peuvent être calculés en temps et espace linéaires. On notera l'utilisation fructueuse des parcours en largeur $Lex-BFS$ pour cette classe de graphes dans [CDHP98], [DN00] et [LRT00].

En se basant sur une décomposition propre à ces graphes (***skeleton tree***), les auteurs de [BDM$^+$88] ont montré que le problème de l'isomorphisme peut être résolu en temps polynomial pour ces graphes.

Plusieurs problèmes $\mathcal{NP}$-complets ont été étudiés sur les graphes distance-héréditaires : voir par exemple [DM88, Dra94, DOS94, OS95, BD98] pour les arbres de Steiner et des problèmes d'ensembles dominants, voir [BDK00] pour les problèmes de fill-in et de tree-width. Certains problèmes hamiltoniens ont été étudiés dans [MN93]. A partir d'une réécriture des séquences d'élagage sous forme d'arbres, les auteurs de [HHH$^+$98] ont proposé un cadre pour unifier et résoudre en temps linéaire plusieurs problèmes $\mathcal{NP}$-complets sur les graphes distance-héréditairesṖarmi les problèmes résolus on peut relever : le problème du stable maximum, de la clique maximum, de l'ensemble dominant minimum, du stable dominant minimum, de l'ensemble connnexe dominant minimum, le problème des arbres de Steiner, le calcul de connectivité par rapport aux sommets et par rapport aux arêtes (voir par exemple [GJ79] et [Gol80] pour des définitions de ces problèmes). Cependant la catégorie de problèmes qu'il se disent capables de résoudre est exprimée de manière assez compliquée avec de lourdes notations et elle ne s'apparente pas à quelque chose de connu.

D'un point de vue encore plus théorique, il a été montré que les graphes distance-héréditaires sont de clique-width ≤ 3 [GR99] avec une construction linéaire d'une 3-expression (toujours grâce à une séquence d'élagage). Pour k fixé, tout graphe de clique-width $\leq k$ est représentable par une k-expression (pour ces définitions, voir par exemple [CMR00]), et un théorème de Courcelle, Makowski et Rotics présente une classe logique telle que tout problème exprimable dans cette logique peut se résoudre en temps linéaire pour les graphes de clique-width $\leq k$ donnés par une k-expression [CMR00]. Cela implique qu'il existe des algorithmes linéaires pour résoudre sur les graphes distance-héréditaires une grande quantité de problèmes classiques exprimables dans cette classe logique précise. Les problèmes suivants de [GJ79] en font partie : *vertex cover* (GT1), *dominating set* (GT2), *domatic number* pour k fixé (GT3), *c-colorability* pour c fixé (GT4), *partition into cliques* pour k fixé (GT15), *clique* (GT19), *independent set* (GT20), *induced path* (GT23), *unweighted Steiner trees* (ND12), mais par exemple la colorabilité dans le cas général n'en fait pas partie. Pour les graphes de clique-width bornée, il a été montré que d'autres problèmes que ceux présentés ci-dessus peuvent aussi être résolus en temps polynomial (voir [KR01] et [EGW01] où un schéma général de résolution est suggéré). Parmi ces autres problèmes résolus en temps polynomial (pas forcément linéaire), on peut citer *partition into cliques/independent sets/forests/complete bipartite subgraphs*, *Hamiltonian path* ou des problèmes variés de séparateurs avec des sommets ou des arêtes.

Le paramètre de la clique-width est un outil très puissant pour situer la complexité des problèmes sur certaines classes de graphes. Cependant les algorithmes associés à ce paramètre peuvent présenter des constantes cachées très importantes dépendant de la clique-width et pour les algorithmes polynomiaux non linéaires les exposants dépendent aussi de la clique-width. C'est pourquoi il est parfois intéressant de poursuivre la recherche d'algorithmes linéaires simples exploitant toutes les propriétés spécifiques de la classe étudiée.

En fin de compte, les nombreuses caractérisations de cette classe de graphes permettent différentes approches des problèmes d'optimisations. Parmi les outils, on peut mentionner le calcul d'une bonne numérotation des sommets (ordres *Lex-BFS*, perfect orders), le calcul de bonnes décompositions (tree skeleton, décomposition modulaire) ou l'utilisation de la programmation dynamique dans [CHC97] ou [HHH+98]. Des algorithmes parallèles pour ces graphes ont été étudiés dans [Dah95] et [HHH+99].

Nous allons proposer dans ce qui suit un schéma général de résolution de plusieurs problèmes d'optimisation : l'idée est de construire des algorithmes très simples, n'utilisant en entrée qu'une séquence d'élagage du graphe distance-héréditaire, qui plus est une séquence quelconque, et résolvant le problème en une seule passe sur la séquence. C'est pourquoi on pourra parler de résolution "à la volée". Cette approche se situe entre l'écriture d'algorithmes ad-hoc écrits pour les graphes distance-héréditaires et la très forte propriété qu'il s'agit de graphes de clique-width bornée avec toutes les conséquences algorithmiques qui en découlent. Elle se rapproche de la tentative d'unification de problèmes proposée par [HHH+98], même si pour l'instant nous n'avons étudié que peu de problèmes.

Un tel type d'algorithme est présent dans [HM90] où Hammer et Maffray résolvent le problème de la clique de poids maximum en temps et espace $\mathcal{O}(n)$, une fois une séquence d'élagage donnée en entrée. Ils y présentent aussi un algorithme pour l'ensemble indépendant (aussi appelé "stable") de poids maximum, mais malheureusement leur algorithme ne fonctionne pas.

8.3.1 Un schéma d'algorithmes

Les algorithmes que nous allons présenter résolvent des problèmes d'optimisation et suivent tous le schéma suivant :

1. Ce sont des algorithmes gloutons prenant en entrée une séquence d'élagage (quelconque) d'un graphe distance-héréditaire, ainsi qu'une fonction de poids définie sur les sommets.
2. La sortie est une solution au problème d'optimisation considéré, avec la valeur optimum pour le graphe étiqueté ainsi qu'une structure réalisant cet optimum.
3. Ils résolvent le problème considéré en une seule passe (lecture) sur la séquence d'élagage.
4. Suivant le problème à résoudre, les sommets sont étiquetés par des entiers, des réels, des ensembles de sommets ou d'autres types de variables. Les algorithmes mettent à jour ces variables selon trois règles différentes qui dépendent du type de suppression dans la séquence (T, F ou P) et en utilisant des opérations élémentaires (par exemple sur les

entiers : min,max,+). Chaque étape de mise à jour s'effectue en temps constant.

5. Comme ces algorithmes résolvent le problème considéré en une seule passe (lecture) sur la séquence d'élagage, ils s'exécutent en temps et espace $\mathcal{O}(n)$.

6. Les preuves de ces algorithmes ont la même structure.

Par commodité, on continuera à désigner par G le graphe distance-héréditaire G une fois étiqueté. Les problèmes d'optimisation qui vont être résolus dans les sections 8.3.2, 8.3.3, 8.3.4, peuvent s'exprimer sous la forme suivante. Un ensemble Λ_G est défini à partir de G (par exemple dans la section 8.3.2 ce sera l'ensemble des stables de G). Une fonction de poids W_G est définie sur Λ_G (par exemple dans la section 8.3.2 chaque sommet de G est étiqueté par un couple de réels $(p(x), q(x))$ et pour tout $S \in \Lambda_G$ la fonction de poids est définie par $W_G(S) = \sum_{x \in S} p(x) + \sum_{x \notin S} q(x))$. Le problème d'optimisation consiste alors à calculer $\max\{W_G(S)|S \in \Lambda_G\}$ et trouver une solution $S \in \Lambda_G$ qui atteint ce maximum.

La structure de nos algorithmes et de leurs preuves peut s'exprimer de la manière suivante. En supposant que la suppression d'un sommet de G (pendant ou jumeau) donne le graphe G', on définit des règles de mise à jour des étiquettes de G pour calculer les nouvelles étiquettes de G', ainsi qu'une fonction Φ de Λ_G dans $\Lambda_{G'}$ vérifiant la propriété $\mathcal{R}(\Phi)$ signifiant que Φ est surjective et pour tout $S' \in \Lambda_{G'}$, $W_{G'}(S') = \max\{W_G(S)|S \in \Phi^{-1}(S')\}$. Cela assure que $\max\{W_G(S)|S \in \Lambda_G\}$ reste constant tout au long de la séquence d'élagage.

Remarquer que dans les algorithmes qui vont être présentés à aucun moment on ne recalcule les listes d'adjacence ou la matrice d'adjacence du graphe considéré. On travaille uniquement avec un codage des graphes distance-héréditaires qui est la donnée d'une séquence d'élagage et on cherche à exploiter au mieux ce codage. Même si les problèmes que nous avons traités ont déjà été résolus dans la littérature, les algorithmes qui suivent ont comme atout leur simplicité et comme originalité le fait qu'ils résolvent les problèmes "à la volée" en une seule passe sur la séquence d'élagage.

Les algorithmes exposés seront déroulé sur le graphe distance-héréditaire de la figure 8.3 dont une séquence d'élagage est iTh,hFc,gPc,fPb,eFa,dPa,cPb,bPa. On utilisera cette séquence en entrée des algorithmes.

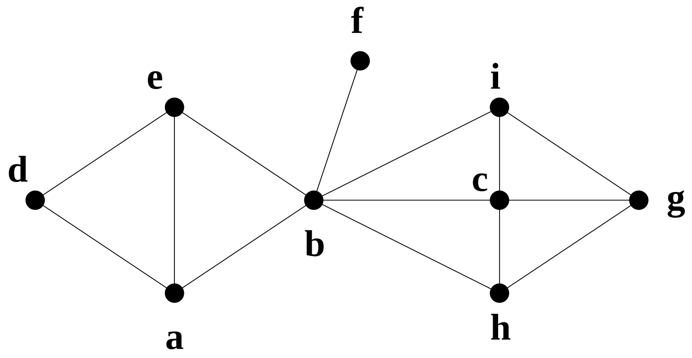


FIG. 8.3 – Graphe distance-héréditaire choisi comme exemple

8.3.2 Stable de poids maximum

L'énoncé du problème est rappelé ci-dessous (les ***stables*** d'un graphe sont aussi appelés ***ensembles indépendants***). Le problème de décision associé est $\mathcal{NP}$-complet dans le cas général (problème GT20 dans [GJ79]).

Stable de poids maximum :
Données : un graphe $G = (V, E)$ et chaque sommet $x \in V$ a un poids $w(x) \in \mathbb{R}^+$.
Résultat : le poids maximum d'un stable S de G (un ***stable*** S de G est un sous-ensemble de V tel que $\forall x, y \in S, xy \notin E$, et son ***poids*** vaut $\sum_{x \in S} w(x)$).

Dans leur article, Hammer et Maffray avaient proposé un algorithme toujours selon le même schéma, mais leur algorithme (non prouvé) ne produit pas l'optimum.

La figure 8.4 résume les différentes règles de mise à jour des poids de sommets suivant le type de suppression dans la séquence d'élagage.

Le déroulement de l'algorithme 3 pour le graphe de la figure 8.3 est présenté sur la figure 8.5. Dans cette illustration, les deux poids associés à chaque sommet sont entre parenthèses et les ensembles entre accolades.

Algorithme 3: Stable de poids maximum pour les graphes distance-héréditaires

Données : Une séquence d'élagage du graphe distance-héréditaire G avec un poids $w(x) > 0$ pour chaque sommet x.

Résultat : Un stable S de G qui maximise $\sum_{x \in S} w(x)$.

begin

A chaque sommet x est assocé un couple de poids $(p(x), q(x))$ et un couple $(P(x), Q(x))$ de sous-ensembles de V :

pour chaque *x dans V* **faire**

- $p(x) \leftarrow w(x)$, $q(x) \leftarrow 0$;
- $P(x) \leftarrow \{x\}$, $Q(x) \leftarrow \emptyset$;

tant que *la séquence d'élagage n'est pas vide* **faire**

Soit $x * y$, où $* \in \{F, T, P\}$, la tête de la séquence d'élagage. Mettre à jour de la manière suivante :

cas où xTy

Remplacer le couple de poids associés à y par :
$(\max(p(x) + p(y), p(x) + q(y), p(y) + q(x)), q(x) + q(y))$

Parmi les sommes $p(x) + p(y)$,$p(x) + q(y)$ et $p(y) + q(x)$, choisir une somme de valeur maximum, et remplacer le couple d'ensembles associés à y de la manière suivante :

cas où *c'est* $p(x)+p(y)$: remplacer par $(P(x) \bigcup P(y), Q(x) \bigcup Q(y))$;
cas où *c'est* $p(x)+q(y)$: remplacer par $(P(x) \bigcup Q(y), Q(x) \bigcup Q(y))$;
cas où *c'est* $p(y)+q(x)$: remplacer par $(P(y) \bigcup Q(x), Q(x) \bigcup Q(y))$;

cas où xFy

Remplacer le couple de poids associés à y par :
$(\max(p(x) + q(y), p(y) + q(x)), q(x) + q(y))$

Parmi les sommes $p(x) + q(y)$ et $p(y) + q(x)$, choisir une somme de valeur maximum, et remplacer le couple d'ensembles associés à y de la manière suivante :

cas où *c'est* $p(x)+q(y)$: remplacer par $(P(x) \bigcup Q(y), Q(x) \bigcup Q(y))$;
cas où *c'est* $p(y)+q(x)$: remplacer par $(Q(x) \bigcup P(y), Q(x) \bigcup Q(y))$;

cas où xPy

Remplacer le couple de poids associés à y par :
$(p(y) + q(x), \max(p(x) + q(y), q(x) + q(y)))$

Parmi les sommes $p(x) + q(y)$ et $q(x) + q(y)$, choisir une somme de valeur maximum, et remplacer le couple d'ensembles associés à y de la manière suivante :

cas où *c'est* $p(x)+q(y)$: remplacer par $(P(y) \bigcup Q(x), P(x) \bigcup Q(y))$;
cas où *c'est* $q(x)+q(y)$: remplacer par $(P(y) \bigcup Q(x), Q(x) \bigcup Q(y))$;

Supprimer la tête de la séquence d'élagage et supprimer x du graphe.

Il ne reste plus qu'un sommet x, avec les poids $(p(x), q(x))$ et les ensembles $(P(x), Q(x))$.

si $p(x) \geq q(x)$ **alors** retourner l'ensemble $P(x)$ et le poids $p(x)$
sinon retourner l'ensemble $Q(x)$ et le poids $q(x)$.

end

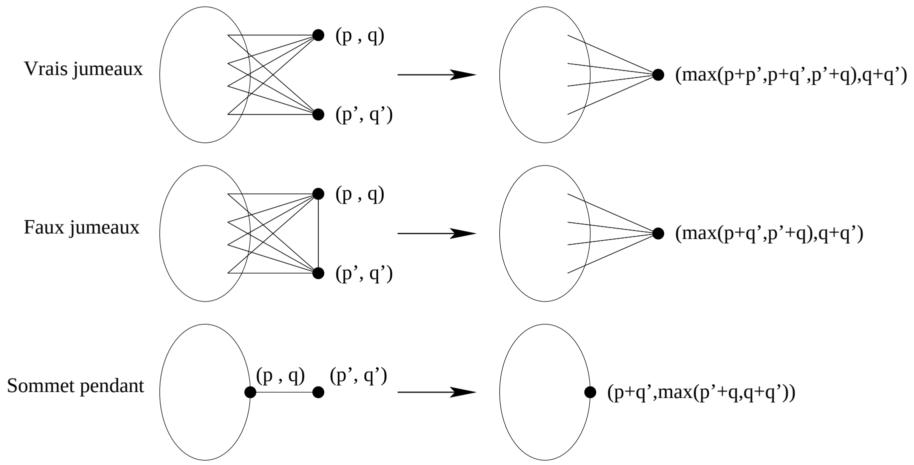


FIG. 8.4 – Mise à jour des poids pour les trois différents types d'élagage

Théorème 8.4. *L'algorithme 3 retourne un stable de poids maximum du graphe initial. Il peut être implanté pour s'exécuter en temps et en espace $\mathcal{O}(n)$.*

Preuve. On suit le schéma général présenté dans la section 8.3.1. On note Λ_G l'ensemble de tous les stables du graphe G. La fonction de poids W_G est définie pour tout $S \in \Lambda_G$ par $W_G(S) = \sum_{x \in S} p(x) + \sum_{x \notin S} q(x)$. La fonction Φ de Λ_G dans $\Lambda_{G'}$ correspondant aux mises à jour de l'algorithme 3 est définie par :

– Si on a xTy ou xFy, alors pour tout $S \in \Lambda_G$,

$$\Phi(S) = \begin{cases} S & \text{if } x \notin S \\ (S \backslash \{x\}) \bigcup \{y\} & \text{if } x \in S \end{cases}$$

– Si on a xPy, alors pour tout $S \in \Lambda_G$, $\Phi(S) = S \backslash \{x\}$.

Soit G_0 le graphe distance-héréditaire initial. Trouver $\max\{W_{G_0}(S) | S \in \Lambda_{G_0}\}$ revient exactement à trouver le poids maximum d'un stable de G_0 du fait de l'étape d'initialisation de l'algorithme 3.

La vérification que la propriété $\mathcal{R}(\Phi)$ est vraie est une étude cas, qui n'est pas développée ici et est laissée au lecteur. La conséquence de cette propriété est que $\max\{W_G(S) | S \in \Lambda_G\} = \max\{W_{G_0}(S) | S \in \Lambda_{G_0}\}$ est un invariant de l'algorithme 3. A la fin de l'algorithme, une fois la séquence délagage terminée, le graphe courant G est réduit à un seul sommet x et donc les deux seuls stables de G sont $\emptyset$ et $\{x\}$ avec $W_G(\emptyset) = q(x)$ et $W_G(\{x\}) = p(x)$. Le dernier test retourne donc un poids qui est le poids maximum d'un stable de G_0, grâce à l'invariant.

Il reste à prouver que l'ensemble retourné est bien un stable du graphe initial G_0 dont le poids réalise bien l'optimum. Pour cela, il suffit de prouver les invariants suivants (ici aussi nous ne développons pas la preuve qui est une simple étude de cas) :

(1) Pour tout couple de sommets $x, y \in V$, $x \neq y$,

$$(P(x) \cup Q(x)) \cap (P(y) \cup Q(y)) = \emptyset.$$

(2) Pour tout sommet $x \in V$,

$$\sum_{v \in P(x)} w(v) = p(x) \text{ and } \sum_{v \in Q(x)} w(v) = q(x).$$

(3) Pour tout couple de sommets $x, y \in V$, $x \neq y$, les ensembles associés vérifient :

$$\begin{cases} Q(x) \bigcup Q(y) \in \Lambda_{G_0}, & (i) \\ P(x) \bigcup Q(y) \in \Lambda_{G_0}, & (ii) \\ P(x) \bigcup P(y) \in \Lambda_{G_0} \text{ si } xy \notin E. & (iii) \end{cases}$$

(4) Pour tout sommet $x \in V$, $P(x)$ et $Q(x)$ sont tous deux dans Λ_{G_0}.

L'algorithme utilise uniquement une séquence d' élagage, il peut s'implanter avec une complexité $\mathcal{O}(n)$ en temps et en espace. ■

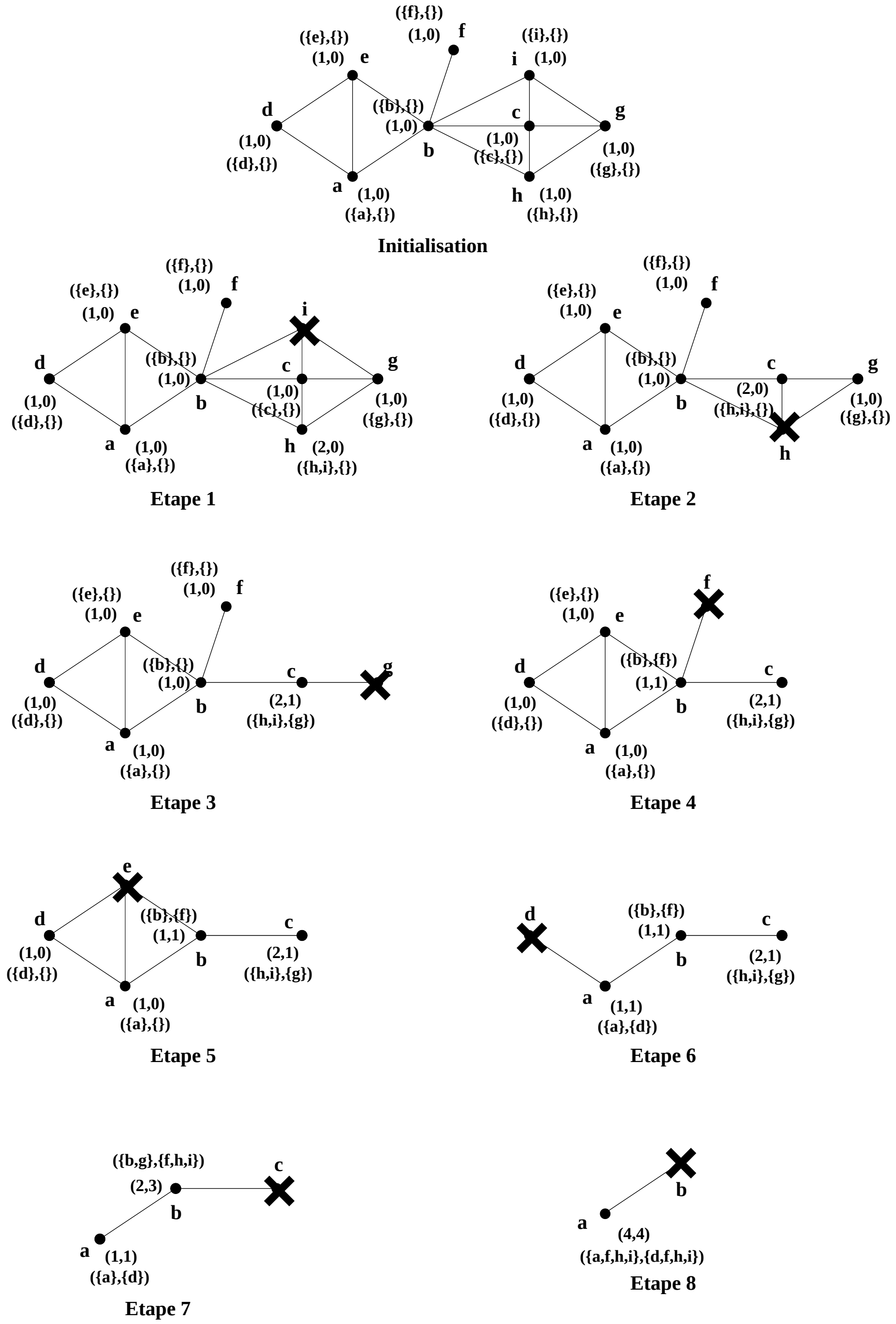


FIG. 8.5 – Exemple d'exécution pour le problème du stable de poids maximum

8.3.3 a, b-Séparateur de poids minimum

a, b-Séparateur de poids minimum :
<u>Données</u> : un graphe $G = (V, E)$, deux sommets $a, b \in V$ et chaque sommet $x \in V$ a un poids $w(x) \in \mathbb{R}^+$.
<u>Résultat</u> : la valeur minimum de $\sum_{x \in S} w(x)$ pour S un a, b-séparateur de G (un a, b-***séparateur*** S de G est un sous-ensemble de V tel que a et b appartiennent à deux composantes connexes distinctes dans le sous-graphe induit sur $V \backslash S$).

Ce problème est polynomial, mais non linéaire dans le cas général.

La figure 8.6 résume les règles de mises à jour des poids et des sommets marqués suivant le type de suppression dans la séquence d'élagage. L'algorithme 4 maintient deux sommets marqués (et seulement deux), initialisé à a et b. Sur la figure, les sommets marqués sont encerclés et les pointillés relient des sommets jumeaux (qu'ils soient vrais ou faux jumeaux).

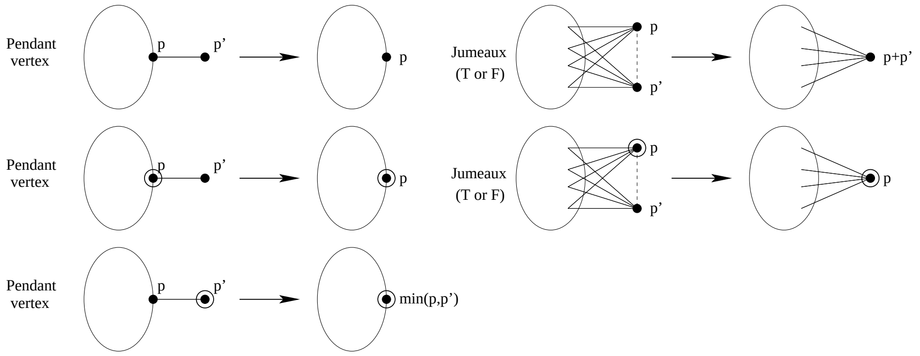


FIG. 8.6 – Mise à jour des poids pour les différents types d'élagage

Algorithme 4: a, b-Séparateur de poids minimum pour les graphes distance-héréditaires

Données : Une séquence d'élagage du graphe distance-héréditaire G avec un poids $w(x) > 0$ pour chaque sommet x, et deux sommets a et b de ce graphe.

Résultat : Un a, b-séparateur S de G qui minimise $\sum_{x \in S} w(x)$.

```
begin
    Marquer juste les sommets a et b ;
    Chaque sommet x reçoit un poids p(x) et un sous-ensemble P(x) de V ;
    p(a) ← ∞ ; p(b) ← ∞ ;
    P(a) ← ∅ ; P(b) ← ∅ ;
    foreach x ∈ V\{a, b} do
        p(x) ← w(x) ;
        P(x) ← {x} ;
    Soit x ∗ y, où ∗ ∈ {F, T, P}, la tête de la séquence d'élagage ;
    tant que x et y ne sont pas tous les deux marqués faire
        si x est marqué alors
            Marquer y ;
            si xTy ou xFy ou (xPy et p(x) < p(y)) alors
                p(y) ← p(x) ; P(y) ← P(x) ;
        sinon
            si y n'est pas marqué et (xTy ou xFy) alors
                p(y) ← p(x) + p(y) ;
                P(y) ← P(x) ⋃ P(y) ;
        Supprimer la tête de la séquence d'élagage et supprimer x du graphe ;
        Soit x ∗ y, où ∗ ∈ {F, T, P}, la tête de la séquence d'élagage ;
    Quand x et y sont marqués, il y a deux cas :
    cas où xFy ou xPy
        Entre les couples (P(x), p(x)) et (P(y), p(y)), retourner l'ensemble associé
        au poids minimum ;
    cas où xTy
        En utilisant ce qui reste de la séquence d'élagage, calculer ⋃_{v∈N(y)} P(v) et
        ∑_{v∈N(y)} p(v)) (résultats stockés dans sum et set) de la manière suivante :
        Supprimer la tête de la séquence d'élagage et supprimer x du graphe ;
        Soit v ∗ w, où ∗ ∈ {F, T, P}, la nouvelle tête de la séquence d'élagage ;
        sum ← 0 ; set ← ∅ ;
        tant que la séquence d'élagage n'est pas vide et non(v est marqué et vPw)
        faire
            cas où v est marqué
                Marquer w ;
                si vFw alors sum ← sum + p(w) ; set ← set ∪ P(w) ;
            cas où w est marqué et (vPw ou vFw) : sum ← sum + p(w) ;
            set ← set ∪ P(w) ;
            cas où vFw ou vTw : sum ← sum + p(w) ; set ← set ∪ P(w) ;
            Supprimer la tête de la séquence d'élagage et supprimer v du graphe ;
            Soit v∗w, où ∗ ∈ {F, T, P}, la nouvelle tête de la séquence d'élagage ;
        si v est marqué et vPw alors sum ← sum + p(w) ; set ← set ∪ P(w) ;
        Des trois couples (P(x), p(x)), (P(y), p(y)) et
        (⋃_{v∈N(x)} P(v), ∑_{v∈N(x)} p(v)), retourner l'ensemble associé au poids
        minimum ;
end
```

Théorème 8.5. *L'algorithme 4 retourne un a,b-séparateur de poids minimum du graphe initial. On peut l'implanter pour qu'il s'exécute en temps et espace $\mathcal{O}(n)$.*

Preuve. On suit le schéma général présenté dans la section 8.3.1. Pendant l'exécution de l'algorithme 4 il y a exactement deux sommets marqués, on les note z_G et z'_G. On note $\Lambda_G^{z_G,z'_G}$ la famille de tous les z_G, z'_G-séparateurs de G et par Λ_G l'ensemble $\{S \subseteq V | S \in \Lambda_G^{z_G,z'_G}$ or $S = \{z_G\}$ or $S = \{z'_G\}\}$. La fonction de poids W_G est définie pour tout $S \in \Lambda$ par $W_G(S) = \sum_{x \in S} p(x)$. La fonction Φ de Λ_G dans $\Lambda_{G'}$ correspondant aux mises à jour dans l'algorithme 4 est définie par :

- Si on a ((xTy ou xFy) avec y marqué) ou (xPy avec x marqué), alors pour tout $S \in \Lambda_G$, $\Phi(S) = S \backslash \{x\}$.
- Si on a xPy avec x marqué, alors pour tout $S \in \Lambda_G$,

$$\Phi(S) = \begin{cases} \{y\} & \text{si } S = \{x\} \\ S & \text{sinon} \end{cases}$$

- Si on a (xTy ou xFy) avec x marqué, alors pour tout $S \in \Lambda_G$,

$$\Phi(S) = \begin{cases} \{y\} & \text{si } S = \{x\} \\ S \backslash \{x\} & \text{sinon} \end{cases}$$

- Si on a (xTy ou xFy) avec x et y non marqués, alors pour tout $S \in \Lambda_G$,

$$\Phi(S) = \begin{cases} S \backslash \{y\} & \text{si } \{x,y\} \subseteq S \\ S \backslash \{x,y\} & \text{sinon} \end{cases}$$

Soit G_0 le graphe distance-héréditaire initial. Trouver $\min\{W_{G_0}(S) \mid S \in \Lambda_{G_0}\}$ revient exactement à trouver le poids minimum d'un a,b-séparateur de G_0 du fait de l'étape d'initialisation de l'algorithme 4. Comme la propriété $\mathcal{R}(\Phi)$ est vérifiée (preuve par cas omise ici), $\min\{W_G(S) | S \in \Lambda_G\} = \min\{W_{G_0}(S) | S \in \Lambda_{G_0}\}$ est un invariant de l'algorithme 4. Pour prouver que les sous-ensembles étiquetant les sommets fournissent bien un a,b-séparateur de poids optimum, nous donnons les invariants vérifiés par l'algorithme ? ? ? que la premiere phase ? ? ?

(1) Pour tout couple de sommets $x, y \in V$, $x \neq y$, $P(x) \bigcap P(y) = \emptyset$.
(2) Pour tout sommet $x \in V$ tel que $p(x) < \infty$, $\sum_{v \in P(x)} w(v) = p(x)$.
(3) Soient z_G et z'_G les sommets marqués de G. LE graphe courant vérifie :

$$\begin{cases} \forall S \in \Lambda_G^{z_G z'_G}, \ \bigcup_{x \in S} P(x) \in \Lambda_{G_0}^{ab}, & (i) \\ \forall x \in \{z_G, z'_G\}, \ p(x) < \infty \text{ implique } P(x) \in \Lambda_{G_0}^{ab}. & (ii) \end{cases}$$

Comme l'algorithme 3, l'algorithme 4 peut s'implanter avec une complexité $\mathcal{O}(n)$ en temps et en espace. ■

Le déroulement de l'algorithme 4 pour le graphe de la figure 8.3 est présenté sur la figure 8.7. Dans cette illustration, le poids de chaque sommet est indiqué dans un disque gris et les deux sommets marqués sont encerclés. Les ensembles sont mis entre accolades.

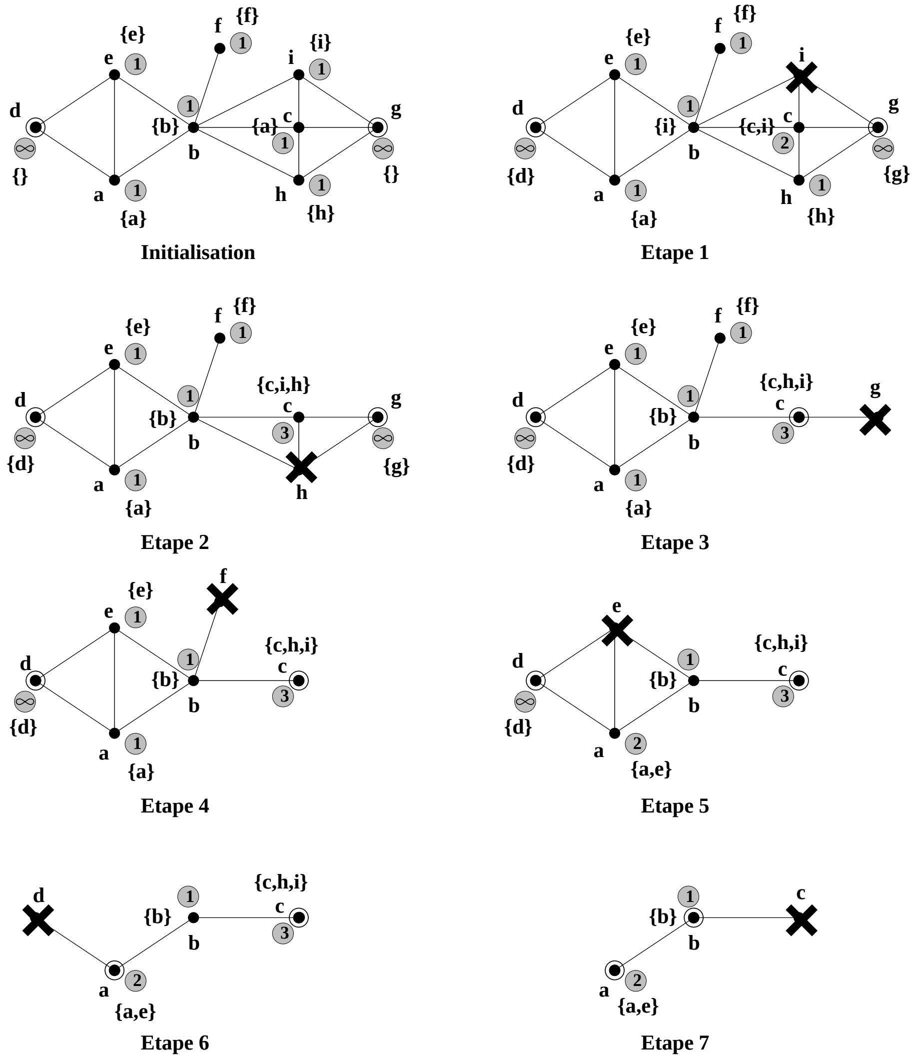


FIG. 8.7 – Exemple d'exécution pour le problème du a, b-séparateur de poids minimum

8.3.4 Diamètre

Diamètre :
Donnée : un graphe $G = (V, E)$.
Résultat : la valeur minimum de la distance $d_G(x, y)$ pour x, y deux sommets de G.

Le calcul du diamètre a été étudié pour de nombreuses classes particulières de graphes. Dans [CDHP98], un algorithme linéaire est décrit pour calculer le diamètre des graphes distance-héréditaires. Leur algorithme est basé sur un ordre *Lex-BFS des sommets*. Nous donnons ici une nouvelle solution utilisant les séquences d'élagage.

L'algorithme 5 calcule le diamètre d'un graphe distance-héréditaire en fournissant une chaine sans corde de longueur maximum du graphe initial. En effet, contrairement au cas général, pour les graphes distance-héréditaires, il suffit de fournir une chaine sans corde de longueur maximum pour obtenir le diamètre du graphe (c'est une simple conséquence de la définition originale des graphes distance-héréditaires par Howorka).

L'algorithme 5 utilise les définitions et notations suivantes. Les listes utilisées représentent des chaines du graphe initial. Pour une chaine S de sommets $x_1, x_2, \ldots, x_k$, on note $debut(S)$ le premier sommet x_1, et $fin(S)$ le dernier sommet x_k. De plus $longueur(S)$ désigne la longueur de la chaine, c'est à dire $k - 1$. On note $S + S'$ la concaténation de deux chaines S et S' (quand $fin(S)$ et $debut(S')$ sont des sommets adjacents) et S^{-1} la chaine S inversée, c'est à dire $x_k, \ldots, x_2, x_1$.

La figure 8.8 résume les règles de mises à jour des poids des sommets suivant les types de suppression dans la séquence d'élagage. La valeur conservée est un candidat pour la valeur du diamètre (elle est comparée à la valeur de $diam$ dans l'algorithme).

Théorème 8.6. *L'algorithme 5 retourne une chaine sans corde du graphe initial dont la longueur est égale au diamètre de ce graphe. On peut l'implanter pour qu'il s'exécute avec une complexité $\mathcal{O}(n)$ en temps et en espace.*

Preuve. On suit le schéma général présenté dans la section 8.3.1. On note Λ_G^* la famille de toutes les chaines sans cordes de G et Λ_G the famille $\Lambda_G^* \bigcup \{D\}$. La fonction de poids W_G est définie pour tout $S \in \Lambda_G$ par $W_G(S) = p(debut(S)) + longueur(S) + p(fin(S))$ si $S \neq D$, et $W_G(D) = diam$. En supposant que la suppression d'un sommet du graphe G (avec la liste D et le poids $diam$) donne le graphe G' (avec la liste D' et le poids $diam'$), la

Algorithme 5: Calcul du diamètre d'un graphe distance-héréditaire

Données : Une séquence d'élagage d'un graphe distance-héréditaire G.

Résultat : Le diamètre $diam(G)$ de G et une chaine sans corde D de longueur $diam(G)$.

```
begin
    A chaque sommet x on associe un poids p(x) et une liste P(x) ;
    pour chaque x ∈ V faire p(x) ← 0 ; P(x) ← ∅ ;
    diam ← 0 ; D ← ∅ ;
    tant que la séquence d'élagage n'est pas vide faire
        Soit x ∗ y, où ∗ ∈ {F, T, P}, la tête de la séquence d'élagage ;
        cas où xTy
            si p(x) + p(y) + 2 > diam alors
                diam ← p(x) + p(y) + 2 ;
                Choisir un sommet v ∈ N(x) et D ← P(x)^{-1} + v + P(y) ;
            si p(x) > p(y) alors P(y) ← P(x) ;
            p(y) ← max(p(x), p(y)) ;
        cas où xFy
            si p(x) + p(y) + 1 > diam alors
                diam ← p(x) + p(y) + 1 ;
                D ← P(x)^{-1} + P(y) ;
            si p(x) > p(y) alors P(y) ← P(x) ;
            p(y) ← max(p(x), p(y)) ;
        cas où xPy
            si p(x) + p(y) + 1 > diam alors
                diam ← p(x) + p(y) + 1 ;
                D ← P(x)^{-1} + P(y) ;
            si p(x) + 1 > p(y) alors P(y) ← P(x) ;
            p(y) ← max(p(x) + 1, p(y)) ;
        Supprimer la tête de la séquence d'élagage et supprimer x du graphe. ;
    Retourner la valeur diam comme le diamètre et D comme chaine sans corde
    de longueur diam.
end
```

fonction Φ de Λ_G dans $\Lambda_{G'}$ correspondant aux mises à jour de l'algorithme 5 est définie pour tout $S \in \Lambda_G$ par :

$$\Phi(S) = \begin{cases} D' & \text{si } S = D \text{ ou } x, y \in S, \\ (S\backslash\{x\}) \bigcup \{y\} & \text{si } x \in S \text{ et } x \neq debut(S), x \neq fin(S), \\ S\backslash\{x\} & \text{si } x = debut(S) \text{ ou } x = fin(S), \\ S & \text{sinon.} \end{cases}$$

Soit G_0 le graphe distance-héréditaire initial. Trouver $\min\{W_{G_0}(S) \mid S \in$

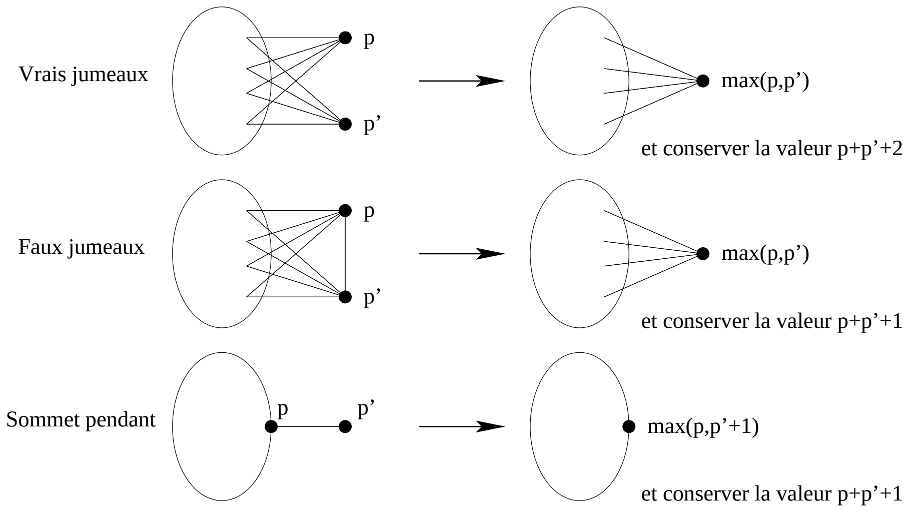


FIG. 8.8 – Mise à jour des poids pour les différents types d'élagage

$\Lambda_{G_0}\}$ revient exactement à trouver la longueur maximum d'une chaine sans corde de G_0 du fait de l'étape d'initialisation de l'algorithme 5. Comme la propriété $\mathcal{R}(\Phi)$ est vérifiée (une preuve par cas omise ici), $\max\{W_G(S)|S \in \Lambda_G\} = \max\{W_{G_0}(S)|S \in \Lambda_{G_0}\}$ est un invariant de l'algorithme 5. Pour prouver que à partir des listes étiquetant les sommets, la liste D finit bien par contenir une chaine sans corde de longueur le diamètre du graphe initial, on a les invariants suivants :

(1) Pour tout couple de sommets $x, y \in V$, $x \neq y$, $P(x) \bigcap P(y) = \emptyset$.

(2) Pour tout sommet $x \in V$, on a $P(x) \in \Lambda^*_{G_0}$ et $longueur(P(x)) = p(x)$.

(3) Pour tout couple de sommets $x, y \in V$, pour tout sommet s dans $P(x)$ et t dans $P(y)$,

$$st \in E \text{ dans } G_0 \implies s = fin(P(x)),\ t = debut(P(y)) \text{ et } xy \in E.$$

(4) $D \in \Lambda^*_{G_0}$ et $longueur(D) = diam$.

L'algorithme peut s'implanter avec une complexité $\mathcal{O}(n)$ en temps et en espace. ■

Le déroulement de l'algorithme 5 pour le graphe de la figure 8.3 est présenté sur la figure 8.9. Dans cette illustration, le poids de chaque sommet est indiqué dans le disque gris et la liste associée dans un petit tableau.

8.4 Perspectives

Ces graphes distance-héréditaires sont un terrain d'expérimentation très intéressant pour les outils mathématiques et algorithmiques de la théorie des graphes : ces graphes se “comportent bien”, les problèmes classiques finissent par trouver une solution, mais malgré tout non triviale.

Question 8. *Est-il possible de caractériser simplement les problèmes sur les graphes distance-héréditaires que l'on peut résoudre avec notre schéma d'algorithmes ?*

Il semble très nécessaire de répondre à cette question au vue des autres travaux dans le domaine, en particulier les résultats très généraux sur la clique-width et aussi la classe de problèmes présentée dans [HHH+98]. Est-ce qu'il est nécessaire de passer par des considérations de complexité descriptive pour identifier ces problèmes et est-ce qu'il est possible d'exhiber une caractérisation plus concise par exemple que celle donnée pour la classe de problèmes de [HHH+98] ? Quels sont d'ailleurs les liens entre notre schéma d'algorithmes et cette classe de [HHH+98] ? Quels sont les liens entre notre schéma d'algorithmes et les algorithmes linéaires pour les graphes distance-héréditaires utilisant les 3-expressions correspondant à la clique-width ?

D'autre part, le problème de l'élagage d'un graphe quelconque a trouvé une solution très élégante grâce aux arbres lexicographiques.

Les arbres lexicographiques apparaissent comme une structure de données très intéressante pour représenter une famille d'ensembles, en particulier dans le cadre de l'algorithmique de graphes. Présentant de très intéressantes bornes de complexité amortie, elle pourrait se révèler très utile pour traiter des problèmes dynamiques sur les graphes, correspondant de fait à une problématique en plein essor actuellement. On ne peut que souhaiter l'élargissement de son champ d'applications.

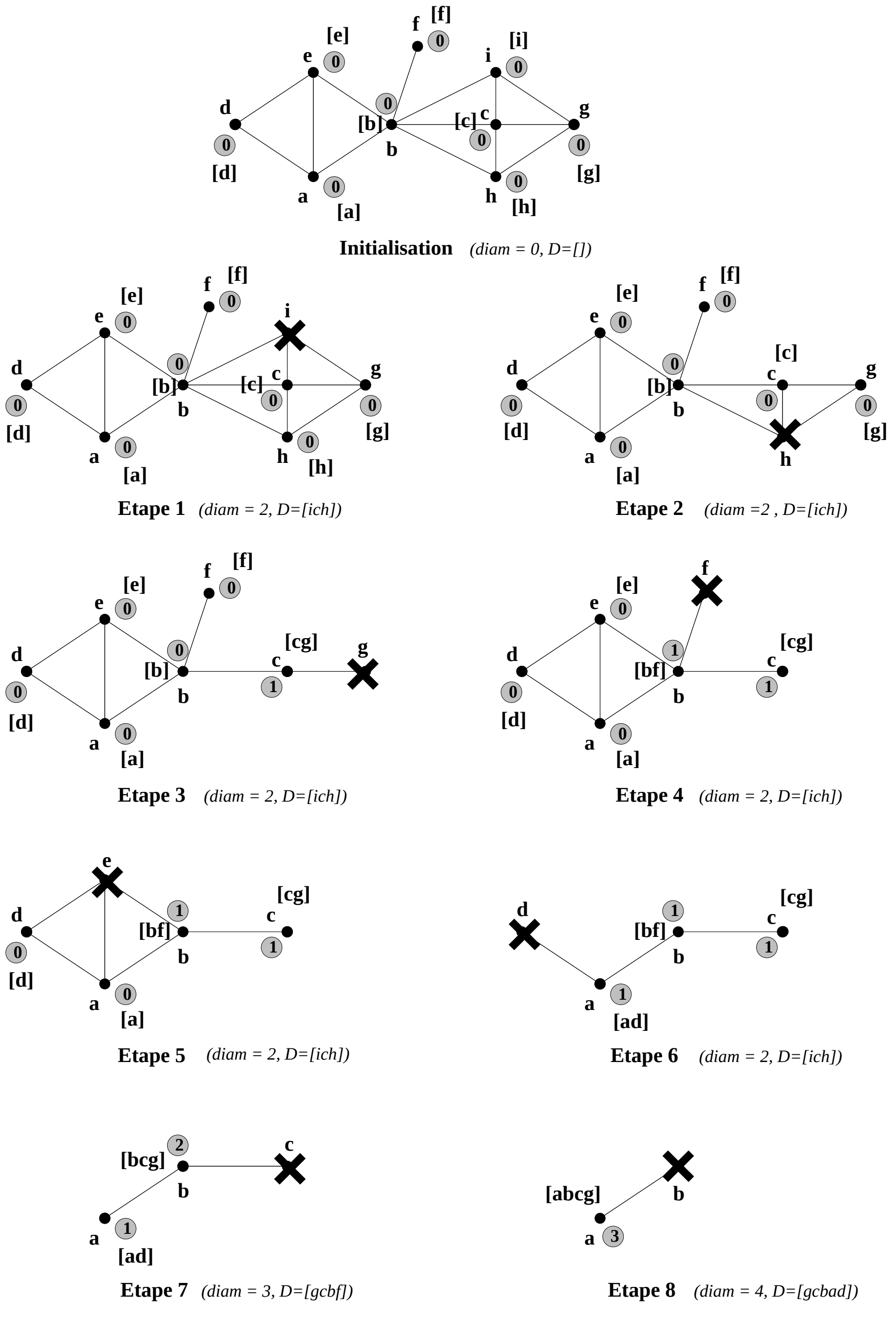


FIG. 8.9 – Exemple d'exécution pour le problème du diamètre

Annexe A

Permutations aléatoires dans le modèle CGM

Ce travail est issus d'une collaboration avec Isabelle Guérin-Lassous rencontrée au LIAFA et maintenant membre de l'équipe ReMap de l'INRIA Rhône-Alpes, dans le cadre de son étude du modèle parallèle à gros grains ***CGM***. Pour une présentation et des références sur ce modèle ainsi que sur d'autres modèles parallèles, le lecteur est renvoyé à la thèse d'Isabelle Guérin-Lassous intitulée "Algorithmes parallèles de traitement de graphes : une approche basée sur l'analyse expérimentale" [GL99].

Un premier travail auquel j'ai participé, a consisté à étudier le comportement d'un algorithme de calcul de composantes connexes d'un graphe dense, dans le modèle CGM. Il s'agissait de vérifier les caractères portable et prédictif de l'algorithme, implanté sur plusieurs plates-formes (plusieurs grappes de PCs et une machine parallèle T3E). Les résultats ont donnés lieu à une présentation courte à la conférence francophone RenPar'97.

L'étude de cet algorithme de composantes connexes, ainsi que d'autres algorithmes dans le modèle CGM, passait par une analyse expérimentale et donc par le choix des entrées sur lesquels les programmes devaient être testés. Pour obtenir des performances moyennes des implantations, il fallait que celles-ci fonctionnent sur un grand nombre d'entrées aléatoires, ces entrées étant des données de grandes tailles réparties sur les différents processeurs. C'est pourquoi nous nous sommes intéressé au problème de la génération de permutations aléatoires dans le modèle CGM.

Plusieurs points ont motivés cette étude :

- Les permutations aléatoires sont souvent utilisés comme base pour construire aléatoirement d'autres objets combinatoires (listes, arbres, graphes, ...).
- Utilité pour la simulation (par exemple, pour évaluer les performances d'un programme).
- Certains algorithmes ont de meilleures performances pour une bonne distribution des données en entrée (permuter aléatoirement les données peut alors être vu comme un précalcul).
- Utilité pour des algorithmes randomisés (des permutations aléatoires peuvent être utilisé pendant l'exécution de l'algorithme).

Il est toujours possible de générer séquentiellement une permutation, puis de répartir ensuite les éléments de la permutation sur les différents processeurs. Nous n'avons pas retenu cette solution car elle ne permettait pas de générer rapidement des permutations de grande taille. En effet, la taille des entrées sur lesquels on souhaite travailler ne tient pas dans la mémoire d'un seul processeur. Par conséquent, le processeur qui calcule une telle permutation se trouve dans sa zone de swap et met donc énormément de temps pour parvenir à une solution. Nous nous sommes donc intéressé à la génération de permutations en parallèle.

Nous nous sommes placés dans le modèle à gros grains CGM pour rester le cadre des autres algorithmes étudiés. Rappelons qu'un tel modèle de calcul parallèle a pour objectif de reflèter les contraintes de machines existantes et peut être de futures machines (***réaliste***), de s'appliquer à des machines variées (***portable***) et de permettre des prédictions de performances (***prédictivité***).

La génération de permutations aléatoires avait déjà été étudié dans le modèle PRAM en particulier. Les trois résultats suivants correspondent à de tels algorithmes de génération de permutations aléatoires de n éléments (on donne la distribution de probabilités obtenue sur les permutations, le temps d'exécution et le nombre de processeurs) :

- Distribution uniforme, en temps $\mathcal{O}(\log n)$ avec $\mathcal{O}(n/\log n)$ processeurs [Rei85].
- Distribution presque uniforme, en temps $\mathcal{O}(n/p + n^{2/3}\log n)$ avec p processeurs [And90].
- Distribution uniforme, en temps $\mathcal{O}(\log^2 n)$ avec $\mathcal{O}(n)$ processeurs [AS96].

Nous avons choisi de travailler avec un modèle parallèle à mémoire distribuée, à gros grains, qui est schématisé par la figure A.1 où les P_i désignent

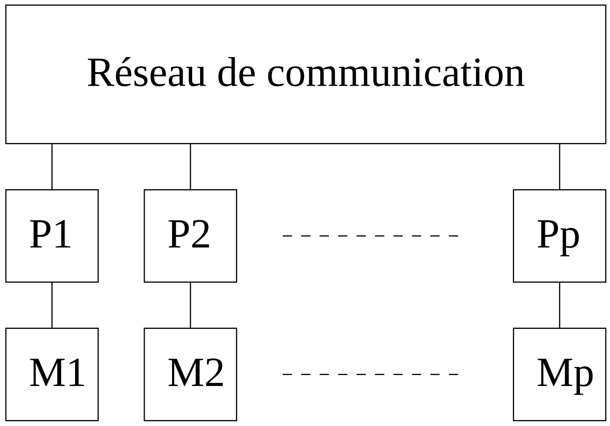


FIG. A.1 – Modèle à gros grain

les processeurs, les M_i leurs mémoires locales, le réseau d'interconnection est pris au sens large : cela peut être un réseau éthernet, un réseau sans fil, une mémoire partagée ou un autre type de réseau. Il existe plusieurs modèles à gros grains comme BSP [Val90], LogP [CDP+93] ou CGM [DFRC93]. Notre choix s'est porté sur le modèle CGM, dont nous rappelons maintenant les principales caratéristiques :

- Tous les processeurs exécutent le même programme.
- Chaque mémoire locale est de taille $\mathcal{O}(n/p)$, où n est la taille totale des données en entrée, p est le nombre de processeurs et $n \gg p$.
- Un algorithme est une alternance de calculs locaux (étape de calcul) et de communications globales (ronde de communication).
- Deux mesures de complexité importantes sont :
  - Le nombre total de calculs locaux.
  - Le nombre de rondes de communications.

Les conséquences lorsqu'on élabore un algorithme dans ce modèle sont les suivantes :

- Chaque processeur reçoit au plus $\mathcal{O}(n/p)$ données pendant une ronde de communication.
- On essaie de minimiser le nombre de rondes de communication (constant ou dépendant de p plutôt que de n).
- On peut réutiliser pour les calculs locaux des algorithmes séquentiels connus et optimaux.
- On essaie de minimiser le volume de commmunication (total des don-

nées échangées pendant les rondes de communication), en particulier par rapport au travail d'algorithmes séquentiels résolvant le même problème.

Avec toutes ces contraintes, nous avons proposé dans [GLT00] trois algorithmes de génération de permutations aléatoires de n éléments avec p processeurs. On suppose que au départ ces éléments sont répartis de manière égale entre les processeurs (chaque processeurs possède n/p éléments). Les résultats sont présentés dans le tableau A, la première colonne donne le numéro de l'algorithme, la deuxième le nombre de rondes de communication, la troisième la mémoire locale utilisé par chaque processeur, la quatrième le volume de communication et la dernière colonne indique la distribution de probabilité obtenue sur les permutations.

Algo.	Rondes de comm.	Taille de la mémoire locale	Volume des comm.	Distribution uniforme
1	1	$\mathcal{O}(n/p)$ (forte proba.)	$\mathcal{O}(n)$	oui
2	2	$\mathcal{O}(n/p)$	$\mathcal{O}(n)$	non
3	$2\log_2 p$	$\mathcal{O}(n/p)$	$\mathcal{O}(n\log_2 p)$	oui

TAB. A.1 – Tableau récapitulatif

Les algorithmes proposés sont simples et faciles à implanter. La difficulté réside dans les preuves de leurs bonnes propriétés. En particulier, on a montré que si on répète l'algorithme 2 plusieurs fois, on tend vers la distribution uniforme sur l'ensemble des permutations. La question de la vitesse de convergence reste ouverte. De plus, aucun des trois algorithmes ne réussit à générer des permutations aléatoires avec les propriétés suivantes : un nombre constant de rondes de communication, des mémoires locales restant de taille $\mathcal{O}(n/p)$ et au final une distribution uniforme sur l'ensemble des permutations. Quelles méthodes peut-on employer pour parvenir à ces propriétés ?

L'article en anglais, reproduit dans les pages suivantes, détaille ces trois algorithmes. Il a été présenté à la conférence OPODIS'2000 [GLT00].

GENERATING RANDOM PERMUTATIONS IN THE FRAMEWORK OF COARSE GRAINED MODELS

ISABELLE GUÉRIN LASSOUS AND ERIC THIERRY

ABSTRACT. We present three algorithms for generating random permutations in the coarse grained model CGM. For each of the proposed algorithms, we study the number of supersteps, the size of the local memory, the overall communication cost and we check if it gives a permutation with the uniform distribution or not. The proposed algorithms are intended to be simple and of practical relevance; the difficulty, in this paper, lies in proving that they have the desired properties.

Keywords: coarse grained models, CGM, random permutations, algorithms

1. INTRODUCTION

Permutations often constitutes the basis to construct random combinatorial objects (lists, trees, graphs, etc) in computer science and thus are key data to test and experiment programs [Den94, FZC94, CC86, CF89].

Some algorithms require good distribution properties of the input in order to be efficient. Therefore it is interesting to have efficient algorithms that randomly permutes the data according to a good distribution. For instance, some randomized distributed algorithms number the objects and assume that in average for each object the numbers assigned to its neighbors are far enough from its own number. A preprocessing phase that randomly permutes the input data makes this assumption correct whatever the input. For an example, see [Sib97]. Note that random permutations of data is equivalent to random permutations of integers.

Few articles deal with the permutation of integers in parallel/distributed setting [Rei85, And90, AS96]. Moreover, all these articles propose algorithms

*I. Guérin Lassous, INRIA Rhône-Alpes, ENS Lyon, 46 alle d'Italie, 69364 LYON Cedex 07 - Email: Isabelle.Guerin-Lassous@ens-lyon.fr, Fax: +33 4 72 72 80 80 – corresponding author

†E. Thierry, LIRMM, Université de Montpellier, 34392 Montpellier Cedex 5 - Email: thierry@lirmm.fr - Fax : +33 4 67 41 85 00

Algorithms	Nb of supersteps	Size of local memory	Overall com. cost	Uniform distribution
1	1	$O(\frac{n}{p})$ (high proba.)	$O(n)$	yes
2	2	$O(\frac{n}{p})$	$O(n)$	no
3	$2\log_2 p$	$O(\frac{n}{p})$	$O(n \log p)$	yes

TABLE 1. Overview on the proposed algorithms (n is the input size and p is the number of processors)

in the classical PRAM model [Jàj92]. But this model is far from being realistic because it assumes that $p = \theta(n^k)$ and that any processor can exchange data with any other processor at constant cost. In this paper, we consider more realistic models that are the coarse grained models like BSP, LogP or CGM [Val90, CKP$^+$93, DFRC96]. These models are devoted to the design of distributed algorithms by reflecting constraints of real machines without being specific to a peculiar machine. They have been developped to facilitate the implementations of such algorithms and lead to efficient and portable code.

In this paper, we address the problem of the random permutation of integers in the framework of coarse models. Our aim is to design algorithms with a small number of supersteps (superstep is defined in Section 2) and a small overhead in communication cost, without overloading the local memories and with a specified random distribution on the output. We propose three different algorithms. For each algorithm, we consider the number of supersteps, the size of the used local memory of each processor, the overall communication cost and we check if the algorithm gives a permutation with the uniform distribution or not. Table 1 shows these features for each proposed algorithm.

In Section 2, we quickly describe the coarse grained models and especially CGM (Coarse Grained Multicomputer). In Section 3, we present Algorithm 1 which is based on the choice of one processor for each element. Then we give Algorithm 2 that uses the division by packets of equal size. Finally we show Algorithm 3 that requires only point-to-point communications.

Note, that all the algorithms that are presented are intended to be simple, but that the difficulty lies in proving that they have the desired properties.

2. The Coarse Grained Models

The coarse grained models assume that p the number of processors is small with respect to the input size n. This asumption includes many existing machines. Valiant is the first to have proposed such a model with BSP [Val90]. Some variations of BSP have been developped like LogP and CGM [CKP$^+$93, DFRC93]. In this paper, we chose to work on the simplest model (that is

CGM) because we think that it is sufficient to design efficient algorithms for the random permutation of integers. Moreover, it has be shown in [GLGM00] that algorithms written in CGM turn out to be feasible, efficient, predictable and portable in most cases.

CGM assumes that: 1) all processors execute the same program, 2) each processor holds $O(\frac{n}{p})$ data and 3) an algorithm is an alternation of local computations and global communications. A phase of local computations followed by a global communication is called a *superstep*.

When designing an algorithm in this model, the goal is to have a small number of local computations, of supersteps and a small overhead in communication costs. It is usually assumed that the number of supersteps does not have to depend on the input size n because it would lead to poor performances.

All of our algorithms use arrays to maintain their data. To explain the algorithms, we often assume that the data are put in a *global array* and that each processor works on a part of this array. In fact, each processor has a *local array* and the union of these arrays forms the global array (which is virtual). We give now an example of a CGM algorithm that we will use in the following. This algorithm re-balances the input such that each processor stores exactly $\frac{n}{p}$ data.

Algorithm 0: Balance

Input: array T of n elements, n_i data per processor i $(1 \leq i \leq p)$
Output: array T of n elements evenly distributed among the processors

1 each processor i sends n_i to all the processors j such that $j > i$
2 each processor i computes the global index of its elements (equal to the local index $+n_1 + .. + n_{(i-1)}$)
foreach *processor* i **do**
3   **for** $(j = 1; j \leq n_i; j = j + 1)$ **do**
      send element $T[j]$ and its new position (*global index of* $T[j]$ mod $(\frac{n}{p} + 1)$) to the processor $\lfloor \frac{global\ index\ of\ T[j]}{\frac{n}{p}+1} \rfloor + 1$

each processor i receives its data

Note that Algorithm 0 has exactly two communication steps (Steps 1 and 3) and requires $O(p + \frac{n}{p})$ local computations at Step 2. Note that at Step 2 we assume that $p \leq \frac{n}{p}$. The overall communication cost is $O(n)$.

Henceforth, we will number input data with integers 1 to n where n is the number of data we want to permute. Our problem thus consists of permuting these integers with the constraints exposed above (features 1), 2) and 3) of CGM and, in addition a minimal number of supersteps).

3. Independent choice of processors

The idea of Algorithm 1 is to adapt the simple PRAM algorithm proposed in [Rei85] to the coarse grained models: each processor independently chooses for each of its elements a target processor and sends it to this processor.

Algorithm 1: Independent choice of processors

Input: array T of n elements divided evenly among the p processors
Output: the random permutation of the data of T stored in an array P

each processor randomly chooses an integer between 1 and p for each element of T
1 **for** $(j = 1; j \leq p; j = j + 1)$ **do**
  Each $T[i]$ that is associated to the integer j is sent to processor j
the received values are stored in an array P
2 each processor locally permutes at random the data of P

Proposition 1. *Algorithm 1 randomly permutes the input data according to the uniform distribution in $O(\frac{n}{p})$ local computations with high probability and in one communication step. The overall communication cost is $O(n)$.*

Proof. At Step 2, each processor permutes its data with any sequential algorithm generating permutations with the uniform distribution [Knu81]. It is easy to see that Algorithm 1 requires only one communication step of total size $O(n)$ at Step 1 and for each processor the local computations are linear in the size of its data.

During the execution of Algorithm 1, it is possible that the data are not equally distributed among the processors so that the memory of some processors might be overloaded compared to what is allowed by the chosen model. Nevertheless, it is easy to balance the data equally likely among the processors by using Algorithm 0. By a similar proof as in [Rei85], it follows that with very high probability the number of elements of P in each processor is smaller than $O(\frac{n}{p})$: $|P|_k$, $1 \leq k \leq p$, denotes the number of elements of P on the processor k. There exists a constant c such that:

$$\forall \alpha \geq 1, Prob(|P|_k \leq c\alpha \frac{n}{p}) \geq 1 - \frac{1}{e^{\alpha \frac{n}{p}}} \geq 1 - \frac{1}{e^{\frac{n}{p}}}. \tag{1}$$

In order to prove the uniform distribution on permutations, we study the reverse construction of Algorithm 1: we start from any permutation that we randomly cut in p packets (some packets may be empty). Now we know the initial value v ($1 \leq v \leq p$) given to each element of T. It means that this cut is associated to a unique sequence of random samples at Step 1. This construction is independent of the starting permutation, therefore the distribution of permutations generated by Algorithm 1 is uniform. □

4. Division by packets of equal size

4.1. **The Algorithm.** Unlike Algorithm 1, Algorithm 2 ensures that the data are always equally distributed among the processors during its execution. Without loss of generality, we can assume that p divides $\frac{n}{p}$.

Algorithm 2: Division by packets

Input: array T of n elements divided evenly among the p processors
Output: the random permutation of the data of T stored in an array P

```
1 each processor locally permutes the data of T in a random way
  each processor divides T in p packets of size n/p^2
2 for (j = 1; j ≤ p; j = j + 1) do
  └ each processor sends the packet j to processor j
3 each processor receives the distinct packet and puts them in P
  each processor locally permutes at random the data of P
```

It is easy to see that the data size on one processor is fixed during the execution of Algorithm 2 and is equal to $\frac{n}{p}$. This algorithm requires also only one communication step of total size $O(n)$ at Step 2.

The drawback of Algorithm 2 is that the generated permutations have not a uniform distribution: at the end of this algorithm, we know that each processor has $\frac{n}{p^2}$ elements numbered between $(i-1)\frac{n}{p}$ and $i\frac{n}{p} - 1$ $(1 \leq i \leq p)$. Algorithm 2 can not generate all the permutations if we run it once. However it is interesting to study what is happening when we iterate this algorithm, as shown in Section 4.2.

Note that [LS94] and [BHJ96] deal with the routing problem according to a given permutation but their context and their aim are different from ours: the target machine of [LS94] is fine grained and we do not think that their results can be extended to the coarse grained models; using the algorithm of [BHJ96] may lead to cases where some processors hold more than $\frac{n}{p}$ data.

4.2. **Iteration of Algorithm 2 .**

Proposition 2. *Any permutation can be obtained after exactly two iterations of Algorithm 2.*

Proof. Since Algorithm 2 starts and ends with local permutations of the elements on each processor, we don't need to consider the order of the elements in the arrays. So we reason on packets of elements on each processor. We assume $n = kp^2$ where k is an integer.

The set of elements on a processor after the first application of Algorithm 2 will be called *transition packet.* The problem is to route the elements from the source packets to the target packets through the transition packets, with respect to the rules of Algorithm 2. In other words, what we need is to show that there exists a configuration of p transition packets where each transition packet holds exactly k elements of each source packet as well as k elements of each target packet, so that it corresponds to two iterations of Algorithm 2.

To show the existence of such a configuration fulfilling the properties given above, we build the following network:

- a vertex for each source packet, a vertex for each target packet, a "source" vertex and a "target" vertex,
- an edge with weight 1 between the "source" and each source packet,
- an edge with weight 1 between each target packet and the "target",
- an edge between each source packet and each target packet weighted by the number of elements going from the source packet to the target packet.

Figure 1 shows a configuration of source and target packets and the associated network.

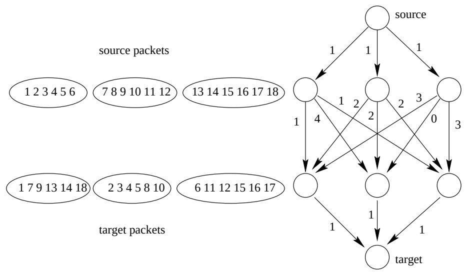


FIGURE 1. An example of network associated to a set of target packets

We now prove the existence of a "perfect matching": there exists a set of p elements composed of exactly one element of each source packet such that this set is also divided into one element by target packet.

The network is such that all the sums of the weights of the edges starting from source packets and all the sums of the weights of the edges ending at target packets are equal (to $\frac{n}{p}$). As a consequence, a flow of weight p actually exists: we use the min-max theorem which shows that the maximal flow is equal to the minimal cut and the computation of the minimal cut gives p (see the annex for the details of the computation) [CLR90]. Thus we choose a flow of weight p and for each edge between a source packet and a target packet whose flow is 1, we arbitrarily choose an element belonging to the source packet and the target packet. We keep this set of p elements and remove these elements from the source and target packets.

With the remaining elements, we can build the same network as above. This network verifies the same property concerning the sums of weights. Thus we can remove again a set of p elements evenly distributed among the p source packets and among the p target packets. We repeat the process until there is no element left in the packets.

At the end, we have pk sets of p elements such that each set is composed of one element per source packet and is also evenly distributed among the target packets. Then we arbitrarily gather these sets by groups of k sets in order to obtain p transition packets of pk elements. These transition packets verify the properties we expected: they receive exactly k elements from each source packet and they send exactly k elements towards each target packet. So it is possible to reach the target packets from the source packets after two iterations of Algorithm 2. □

Figure 2 shows an example of network, a sequence of flows, the associated sets and two iterations of Algorithm 2 which lead to the target packets.

4.3. **Convergence of the distribution.**

Proposition 3. *Whatever the initial distribution, by iterating Algorithm 2 the limit of the distribution is the uniform distribution.*

Proof. Algorithm 2 transforms any initial permutation of n elements into another one with the following properties:

- given the initial permutation, two different sequences of random samples lead to two different permutations,
- therefore, as the random samples have the same probability, all the permutations which can be generated from an initial permutation are equally likely to appear. By symmetry, this probability is the same for any initial permutation,
- if a permutation can be transformed into another one, the second one can be inversely transformed into the first one.

The random transformation of permutations can be seen as a Markov process: the states are the permutations and the directed graph of transitions corresponds to the possible transformations from a permutation into another one with the associated probability. Considering the three previous remarks,

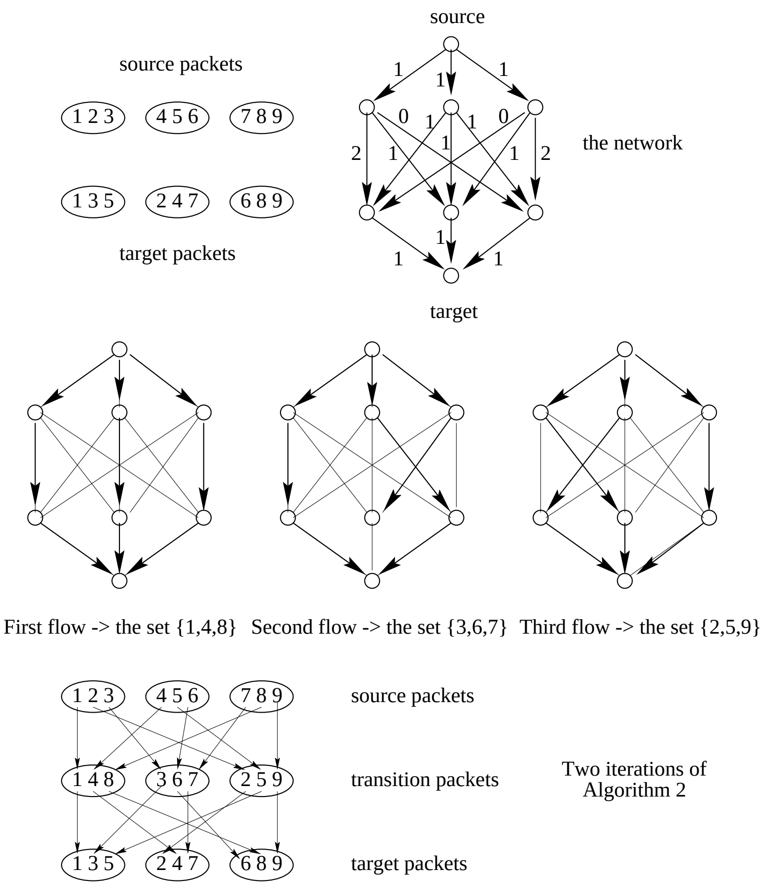


FIGURE 2. The construction of transition packets, in order to route the elements to the target packets.

it is a symmetric regular graph, of degree r ($r = \dfrac{((kp)!)^{2p}}{(k!)^{p^p}}$) and where all the edges have the same weight $(1/r)$. As this graph is symmetric, it can be considered as an undirected graph.

Let A be the adjacency matrix of the graph of the transformation between permutations. Let $M = [m_{i,j}]$ be the associated Markov matrix, i.e. defined by $M = \frac{1}{r}A$:

$$m_{i,j} = \begin{cases} 1/r & \text{if and only if } a_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases}$$

If π_t is a distribution on the $n!$ permutations (a vector with $n!$ non-negative coefficients of sum 1), then after applying Algorithm 2, the new distribution π_{t+1} verifies: $\pi_{t+1} = M\pi_t$.

Then we can use the following theorem about the convergence of finite Markov chains:

Theorem 1. [Nor97] *If the transition matrix M is such that at least one of its powers has no coefficient equal to zero, then when $t \longrightarrow \infty$, $\pi_t \longrightarrow \pi$ for any initial distribution π_0. This limit π is the unique stationary distribution of the Markov chain, i.e. verifying $\pi = M\pi$.*

This theorem applies to our transformation: as a consequence of Proposition 2, all the coefficients of the matrix M^2 are positive (there is always a path of length 2 in the graph linking two permutations). It remains to solve the matricial equation $\pi = M\pi$, which is equivalent to $r\pi = A\pi$.

It is known that, the adjacency matrix A of any regular graph of degree r verifies the following properties:

- The eigenvectors associated to the eigenvalue r belong to the line generated by vector $(1, 1, \ldots, 1)$. So π is the uniform distribution.
- The other eigenvalues of A have absolute values strictly less than r.

□

5. Point-to-point communications

In order to ensure that the memory of each processor can not get overloaded, another solution consists in following a special rule: during a communication step, a processor sends data to only one processor and receive data from only one processor. We now describe an algorithm based on this rule. Without loss of generality, we assume that $p = 2^k$.

The arrows on Figure 3 show which pairs of processors exchange data at each round when $p = 8$.

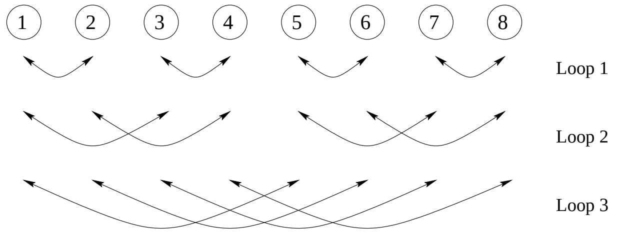


Figure 3. Example of the execution of Algorithm 3.

Algorithm 3: Point-to-point communications

Input: array T of n elements evenly divided among the p processors
Output: the random permutation of the data of T

for $(i = 1; i < k+1; i = i+1)$ **do**
  foreach *processor num* **do**
    if $\lceil \frac{num}{2^{i-1}} \rceil \pmod 2 = 1$ **then**
1   choose equally likely at random a subset of the elements of T
      count the number $size$ of elements of this subset
      send $size$ to processor $num + 2^{i-1}$
      send the chosen subset of T to processor $num + 2^{i-1}$
      receive the data sent by the processor $num + 2^{i-1}$
    else
      receive the value $size$ sent by the processor $num - 2^{i-1}$
2   choose a random subset of its elements of size $size$ with equiprobability
      send this chosen subset of T to processor $num - 2^{i-1}$
      receive the data sent by the processor $num - 2^{i-1}$

Proposition 4. *Algorithm 3 randomly permutes the input data according to the uniform distribution in $O(\frac{n}{p})$ local computations per superstep and in $2\log_2 p$ communication steps. The overall communication cost is in $O(n \log p)$.*

Proof. At Step 1, in each initiator processor (processor having $work$ equal to $true$), each element of T is chosen with probability $\frac{1}{2}$, which constitutes a random subset of T. Step 2 can be realized with a local permutation on the elements of T and then by taking the $size$ first elements of T.

It is straightforward to show that Algorithm 3 requires $2\log_2 p$ communication steps and $O(\frac{n}{p})$ local computations per superstep. Each step communicates $O(n)$ data, then the overall communication cost is in $O(n \log p)$.

Algorithm 3 generates permutations with uniform distribution: all the sequences of exchanges of sets have the same probability, and given a permutation there exists one and only one sequence of exchanges which leads to this permutation. □

Note that, if $p \leq \frac{n}{p}$, then Algorithm 3 requires at least $\log_2 p$ supersteps (as shown in Proposition 5). The assumption that $p \leq \frac{n}{p}$ is a usual hypothesis in the coarse grained models, where we often assume that each processor can at least hold the table of all the processors.

Proposition 5. *If $p \leq \frac{n}{p}$, an algorithm which generates all the permutations with the same probability and respects the rule of point-to-point communications requires at least $\log_2 p$ communication steps.*

Proof. Consider p elements in the first processor. As the algorithm is supposed to generate any permutation, it should be able to generate a permutation such that each of the p elements is allocated to a different processor. Owing to the rules we follow at each round of communication, the elements of a processor are either left on this processor or sent to a unique other processor. As a consequence, after k steps the elements of one processor are spread over at most 2^k different processors. It means that the algorithm needs at least $\log_2 p$ steps of communication to send the p elements over the p processors. □

6. Conclusion

We have presented three distributed algorithms for generating random permutations in the coarse grained models setting. Note that none of these algorithms succeeds in generating permutation with the uniexactly two iterations of Algorithm 2form distribution in a constant number of supersteps and without exceeding $O(\frac{n}{p})$ data per processor.

It will be interesting either to design such an algorithm with these properties, or to show that it is impossible to obtain such an algorithm with such constraints.

We are currently working on the implementation side of these algorithms in order to show they are practically relevant.

References

[And90] R. Anderson. Parallel algorithms for generating random permutations on a shared memory machine. In *Proc. SPAA'90*, pages 95–102, 1990.

[AS96] L. Alonso and R. Schott. A parallel algorithm for the generation of permutation and applications. *Theoretical Computer Science*, 159(1):15–28, 1996.

[BHJ96] D. A. Bader, D. R. Helman, and J. Jj. Practical Parallel Algorithms for Personalized Communication and Integer Sorting. *ACM Journal of Experimental Algorithmics*, 1, 1996.

[CC86] B. Chan and M. S. Chern. Parallel generation of permutations and combinaisons. *BIT*, 26:277–283, 1986.

[CF89] M. Cosnard and A. Ferreira. Generating permutations on a VLSI suitable linear network. *The Computer Journal*, 32(6):571–573, 1989.

[CKP+93] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proceeding of 4-th ACM SIGPLAN Symp. on Principles and Practises of Parallel Programming*, pages 1–12, 1993.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[Den94] A. Denise. *Méthodes de génération d'objets combinatoires de grande taille et problèmes d'énumération*. PhD thesis, Université de Bordeaux I, 1994.

[DFRC93] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputer. In *ACM 9th Symposium on Computational Geometry*, pages 298–307, 1993.

[DFRC96] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. International Journal on Computational Geometry, Vol. 6(No. 3):pp 379–400, 1996.

[FZC94] P. Flajolet, P. Zimmermann, and B. V. Custem. A calculus for the random generation of combinatorial structures. *Theoretical Computer Science*, 132:1–35, 1994.

[GLGM00] I. Guérin Lassous, J. Gustedt, and M. Morvan. Feasibility, Portability, Predictability and Efficiency: Four Ambitious Goals for the Design and Implementation of Parallel Coarse Grained Graph Algorithms. Technical Report RR-3885, INRIA Lorraine, 2000. http://www.inria.fr/RRRT/publications-fra.html.

[Jàj92] J. Jàjà. *An Introduction to Parallel Algorithm*. Addison Wesley, 1992.

[Knu81] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, 1981.

[LS94] J. Y-T. Leung and S. M. Shende. On Multidimensional Packet Routing for Meshes with Buses. *Journal of Parallel and Distributed Computing*, 20:187–197, 1994.

[Nor97] J. R. Norris. *Markov chains*. Cambridge University Press, 1997.

[Rei85] J. H. Reif. An optimal parallel algorithm for integer sorting. In *26th Symposium on Foundations of Computer Science*, pages 496–503, 1985.

[Sib97] J. F. Sibeyn. Better Trade-offs for Parallel List Ranking. In *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures (SPAA'97)*, pages 221–230, 1997.

[Val90] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, Vol. 33(8):103–111, 1990.

Appendix

Probability of overloading a processor with Algorithm 1. Given a fixed processor, for instance processor 1, for any element, the probability to be sent to processor 1 is $1/p$. Therefore the probability that processor 1 receives more than k elements corresponds to the queue of the binomial distribution with parameter $1/p$:

$$\begin{aligned} Pr(|P|_1 \geq k) &= \sum_{i=k}^{n} C_n^i \frac{1}{p^i}(1-\frac{1}{p})^{n-i} \\ &\leq C_n^k \frac{1}{p^k} \end{aligned}$$

Thanks to the Stirling approximation formula, we also have the following upper bound:

$$C_n^k \leq (\frac{en}{k})^k$$

In our problem, $k = \alpha\frac{n}{p}$, with $\alpha \geq 1$. It gives us the upper bound:

$$Pr(|P|_1 \geq \alpha\frac{n}{p}) \leq \frac{1}{e^{\alpha(\log\alpha - 1)\frac{n}{p}}}$$

Computation of the minimal cut of the network built in section 3.1. For the basic definitions on flow networks, see [CLR90].

We suppose that the sums of the weights of the edges starting at each source packet and the sums of the weights of the edges ending at each source are all equal to d, $d \geq 1$. Details of the computation:

- considering the cut between the "source" vertex and the other vertices, the sum of the weights of the edges is p.
- given any cut as the one drawn on Figure 4, we add the weights of edges between the "source" vertex s and the vertices in A, between the vertices in B' and the "target" vertex t, between the vertices in A' and the vertices in B. The sum gives:

$$\begin{aligned} sum &= \sum_{a\in A} weight(s,a) + \sum_{b\in B'} weight(b,t) + \sum_{a\in A', b\in B} weight(a,b) \\ &= |A| + |B'| + \sum_{a\in A', b\in B} weight(a,b) \end{aligned}$$

- If $|A'| \leq |B'|$,

$$sum \geq |A| + |B'| \geq |A| + |A'| = p$$

- If $|A'| > |B'|$,

$$\begin{aligned}\sum_{a\in A',b\in B} weight(a,b) &= \sum_{a\in A',b\in B\cup B'} weight(a,b) - \sum_{a\in A',b\in B'} weight(a,b)\\ &\geq \sum_{a\in A',b\in B\cup B'} weight(a,b) - \sum_{a\in A\cup A',b\in B} weight(a,b)\\ &\geq |A'|d - |B'|d\end{aligned}$$

As a consequence:

$$\begin{aligned}sum &\geq |A| + |B'| + (|A'| - |B'|)d\\ &\geq |A| + |B'| + (|A'| - |B'|)\\ &\geq |A| + |A'| = p\end{aligned}$$

- Therefore the minimal cut is p

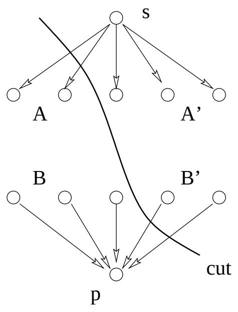


FIGURE 4. A cut of the network, the notations

Authors addresses:
I. Guérin Lassous, INRIA Rhône-Alpes, ENS Lyon, 46 alle d'Italie, 69364 LYON Cedex 07 - Email: Isabelle.Guerin-Lassous@ens-lyon.fr - Fax: +33 4 72 72 80 80
E. Thierry, LIRMM, Université de Montpellier, 34392 Montpellier Cedex 5 - Email: thierry@lirmm.fr - Fax : +33 4 67 41 85 00

Bibliographie

[ABJ89a] R. Agrawal, A. Borgida, and J. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *ACM SIGMOD International Conference on Management of Data*, pages 115–146, 1989.

[ABJ89b] R. Agrawal, A. Borgida, and J. V. Jagadish. Efficient management of transitive relationships in large data bases, including is-a hierarchies. Technical report, Rutgers University, New Jersey, 1989.

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation - Combinatorial optimization problems and their approximability properties.* Springer Verlag, 1999.

[ACP95] G. Ausiello, P. Crescenzi, and M. Protasi. Approximate solution of np optimization problems. *Theoretical Computer Science*, 150 :1–55, 1995.

[AHU74] A. V. Aho, I. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Welsey, 1974.

[AHU71] A. V. Aho, I. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Welsey, 1974, exercise 2.12 on page 71.

[AKBLN89] H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1) :115–146, jan 1989.

[And87] I. Anderson. *Combinatorics of Finite Sets.* Clarendon Press, Oxford University Press, 1987.

[And90] R. Anderson. Parallel algorithms for generating random permutation on a shared memory machine. In *Proceedings of SPAA'90*, pages 95–102, 1990.

[AS88] N. Alon and E. R. Scheinerman. Degrees of freedom versus dimension for containment orders. *Order*, (5) :11–16, 1988.

[AS96] L. Alonso and R. Schott. A parallel algorithm for the generation of permutation and applications. *Theoretical Computer Science*, 159(1) :15–28, 1996.

[Bal94] P. Baldy. *Ensembles ordonnés : algorithmes, structures et applications aux systèmes distribués.* PhD thesis, LIRMM, Université Montpellier II, 1994.

[BD98] A. Brandstaädt and F. F. Dragan. A linear-time algorithm for connected r-domination and steiner tree on distance-hereditary graphs. *Networks*, 31 :177–182, 1998.

[BDK00] H.J. Broersma, E. Dahlhaus, and T. Kloks. A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs. *Discrete Applied Mathematics*, 99(1-3) :367–400, 2000.

[BDM+88] H.-J. Bandelt, A. D'Atri, M. Moscarini, H.-M. Mulder, and A. Schultze. Operations on distance-hereditary graphs. Technical report, Istituto di Analisi dei Sistemi ed Informatica, 1988. R. 226, 7/1988.

[Beh88] G. Behrendt. Maximal antichains in partially ordered sets. *Ars Combin.*, C(25) :149–157, 1988.

[Ber99] K. Bertet. *Sur quelques aspects algorithmiques et structurels des treillis.* PhD thesis, LIAFA, Université Paris VII, 1999.

[BFC00] M. A. Bender and M. Farach-Colton. The lca problem revisited. In *Proceedings of LATIN'2000*, pages 88–94, 2000.

[BFK90] K. P. Bogart, R. Fresse, and J. P. Kung, editors. *The Dilworth Theorems : Selected papers of Robert P. Dilworth.* Birkhauser, 1990.

[BGLT97] B. Bretelle, I. Guérin-Lassous, and E. Thierry. Analyse d'un algorithme de calcul des composantes connexes d'un graphe dans le modèle cgm. Présentation courte à RenPar'97, 1997. Lausanne.

[BGM97] K. Bertet, J. Gusted, and M. Morvan. Weak-orders extensions of an order. In *Proceedings of WG'97*, LNCS 1335, pages 65–77, 1997.

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps and non-approximability - towards tight results. *SIAM J. Comp.*, pages 804–915, 1998.

[BHJR95] V .Bouchitté, A. Hilali, R. Jegou, and J.-X. Rampon. Contiguity orders. In *Franco-Japanese and (n-4th) Franco-Chinese Conference Combinatorics and Computer Science*, volume 8 of *LNCS*, 1995. Rapport de recherche INRIA RR-1970 (1993).

[Bir67] G. Birkhoff. *Lattice Theory*, volume 25 of *Coll. Publ. XXV*. American Mathematical Society, 3rd edition, 1967.

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

[BLS99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes : a survey.* SIAM Monographs on Discrete Mathematics and Applications, 1999.

[BM86] H.-J. Bandelt and H. M. Mulder. Distance-hereditary graphs. *J. Combin. Theory*, Ser. B, 41 :182–208, 1986.

[BM93] P. Baldy and M. Morvan. A linear time and space algorithm to recognize interval orders. *Discrete Applied Mathematics*, 46 :173–178, 1993.

[BMT99] P. Baldy, M. Morvan, and E. Thierry. Extending a partially ordered set : Links with its lattice of ideals. *Order*, 16(4) :305–312, 1999.

[Bou71] A. Bouchet. *Etude combinatoire des ordonnés finis, Applications.* PhD thesis, Université scientifique et médicale de Grenoble, 1971.

[Bou84] A. Bouchet. Codages et dimensions de relations binaires. *Annals of Discrete Math.*, 23, 1984.

[BP69] R. Bonnet and M. Pouzet. Extensions et stratifications d'ensembles dispersés. *C.R. Acad. Sci.*, t. 268-Série A :1512–1515, 1969.

[Cap94] C. Capelle. Representation of an order as union of interval orders. In *Proceedings of ORDAL'94*, LNCS 831, pages 143–161, 1994.

[Cap96] C. Capelle. *Décomposition de graphes et permutations factorisantes*. PhD thesis, LIRMM, Université Montpellier II, 1996.

[Cas93] Y. Caseau. Efficient handling of multiple inheritance hierarchies. In *Proceedings of OOPSLA'93*, pages 271–287, 1993.

[Cas99] N. Caspard. *Etude structurelle et algorithmique de classes de treillis obtenus par duplication*. PhD thesis, Université Paris X|, 1999.

[CDHP98] D. G. Corneil, F. F. Dragan, M. Habib, and C. Paul. Diameter determination on restricted graph families. In *WG'98*, pages 192–202, 1998.

[CDP+93] D. Culler, R. Karp D., Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp : Towards a realistic model of parallel computation. In *Proceedings of 4th ACM SIGPLAN Symp. on Principles and Practises of Parallel Programming*, pages 1–12, 1993.

[Cer00] S. Ceroi. *Ordres et géométrie plane : application au nombre de sauts*. PhD thesis, LIRMM, Université Montpellier II, 2000.

[CHC97] M. S. Chang, S. Y. Hsieh, and G. H. Chen. Dynamic programming on distance-hereditary graphs. In *Proceedings of ISAAC'97*, pages 344–353, 1997.

[CHNR99] Y. Caseau, M. Habib, L. Nourine, and O. Raynaud. Encoding of multiple inheritance hierarchies and partial orders. *Computational Intelligence*, 15 :50–62, 1999.

[Chv84] V. Chvátal. Perfectly orderable graphs. *Annals of Discrete Mathematics*, (21) :63–65, 1984.

[CLR94] T. Cormen, C. Leiserson, and R. Rivest. *Introduction à l'algorithmique*. Dunod, 1994.

[CMR00] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique width. *Theory of Computing Systems*, 33 :125–150, 2000.

[Coh91] N. H. Cohen. Type-extension type tests can be performed in constant time. *ACM Transactions on Programming Languages and Systems*, 13(4) :626–629, 1991.

[CR73] S. A. Cook and R. A. Reckhow. Time bounded random access machines. *Comput. System Sci.*, 7 :354–375, 1973.

[CT01] O. Cogis and E. Thierry. Greedy algorithms for optimization on distance-hereditary graphs. Technical report, LIRMM, 2001.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progression. In *Proceedings of the 19th ACM Symposiom on Theory of Computing*, pages 1–6, 1987.

[Dah95] E. Dahlhaus. Efficient parallel recognition algorithms of cographs and distance hereditary graphs. *Discrete Applied Mathematics*, 57(1) :29–44, 1995.

[Dam97] G. Damiand. Quelques propriétés des graphes distances héréditaires. Master's thesis, Université de Montpellier II, LIRMM, 1997.

[DF93] V. Dahl and A. Fall. Logical encoding of conceptual graph type lattices. In *First International Conference on Conceptual Structures, Canada*, pages 216–224, 1993.

[DFRC93] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputer. In *ACM 9th Symposium on Computational Geometry*, pages 298–307, 1993.

[DHP01] G. Damiand, M. Habib, and C. Paul. A simple paradigm for graphs recognition : application to cographs and distance hereditary graphs. *Theoretical Computer Science*, 263(1-2) :99–111, 2001.

[DI00] C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure : Breaking through the $o(n^2)$ barrier. In *Proceedings of FOCS'2000*, pages 381–389, 2000.

[Dij60] E. W. Dijkstra. Recursive programming. *Numerische Mathematik*, 2 :312–318, 1960.

[Dil50] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51 :161–165, 1950.

[dlHN97] C. de la Higuera and L. Nourine. Drawing and encoding two dimensional posets. *Theoretical Computer Science*, 175(2) :293–308, 1997.

[DM41] B. Dushnik and E. W. Miller. Partially ordered sets. *Amer. J. Math.*, 63 :600–610, 1941.

[DM88] A. D'Atri and M. Moscarini. Distance-hereditary graphs, steiner trees, and connected domination. *SIAM Journal on Computing*, 17(3) :521–538, June 1988.

[DN00] F. F. Dragan and F. Nicolai. Lexbfs-orderings of distance-hereditary graphs with application to the diametral pair problem. *Discrete Appl. Math.*, 98 :191–207, 2000. Technical Report SM-DU-303 , University of Duisburg 1995.

[DOS94] D.P. Day, O. Oellermann, and H.C. Swart. Steiner distance hereditary graphs. *SIAM J. Discrete Math.*, 7 :437–442, 1994.

[DP91] B. A. Davey and H. A. Priestley. *Introduction to lattices and orders*. Cambridge University Press, 1991.

[Dra94] F. F. Dragan. Dominating cliques in distance-hereditary graphs. In *"Algorithm Theory - SWAT'94 " 4th Scandinavian Workshop on Algorithm Theory*, LNCS 824, pages 370–381, 1994.

[Dra95] F. F. Dragan. A central vertex of a distance-hereditary graph in linear time. In *Abstracts of the 8th Meeting of the EURO Working Group on Locational Analysis*, page page 11, September 1995.

[EG85] Engel and Gronau. *Sperner Theory in Partially Ordered Sets*. Number 78. Taubner-Texte zur Mathematik, 1985.

[EGW01] W. Espelage, F. Gurski, and E. Wanke. How to solve np-hard problems on clique-width bounded graphs in polynomial time. In *Proceedings of WG'2001*, 2001.

[EKT91] P. Erdös, H. Kierstead, and T. Trotter. The dimension of random ordered sets. *Random Structures and Algorithms*, 2 :253–275, 1991.

[Ell94] G. Ellis. Implementing hierarchical operations using lattice encodings. Technical report, RMIT University, Australie, 1994.

[Eng97] Engel. *Sperner Theory*. Cambridge University Press, 1997.

[Fal95a] A. Fall. An abstract framework for taxonomic encoding. In *Proceedings of KRUSE'95, Santa Cruz*, 1995.

[Fal95b] A. Fall. Heterogenous encoding. In *Proceedings of KRUSE'95, Santa Cruz*, 1995.

[Fal98] A. Fall. The foundations of taxonomic encodings. *Computational Intelligence*, 14 :598–642, 1998. Simon Fraser University Technical Report 94-20, 1994.

[FFT99] S. Felsner, P. Fishburn, and T. Trotter. Finite three dimensional partial orders which are not sphere orders. *Discrete Math.*, (201) :101–132, 1999.

[FGM99] S. Felsner, J. Gusted, and M. Morvan. Interval reductions and extensions of orders : bijections to chains in lattices. *Order*, 15 :221–246, 1999.

[FH93] P. Fishburn and P. Hammer. Bipartite dimension and bipartite degrees of graphs. Technical Report 76, DIMACS, 1993. Présent à l'adresse http ://dimacs.rutgers.edu/TechnicalReports.

[FHM94] S. Felsner, M. Habib, and R. H. Möhring. On the interplay between interval dimension and dimension. *SIAM Journal on Discrete Mathematics*, 7(1) :32–40, 1994.

[FK98] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *J. Comput. System Sci.*, 57 :187–199, 1998.

[FM71] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *Conference Record 12th Annual Symposium on Switching and Automata Theory*, pages 129–131. IEEE Computer Society, 1971.

[Fro95] V. Froidure. *Rangs des relations binaires, semigroupes de relations non ambigues*. PhD thesis, Université Paris VI, 1995.

[FRS99] S. Felsner, V. Raghavan, and J. Spinrad. Recognition algorithms for orders of small width and graphs of small dilworth number. In *Proceedings of ORDAL'99*, Montpellier, 1999.

[FT99] P. Fishburn and T. Trotter. Geometric containment orders : a survey. *Order*, (15) :167–182, 1999.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[GK79] A. Goralcikova and V. Koubek. A reduct and closure algorithm for graph. In *Mathematical Foundations of Computer Science*, LNCS 74, pages 301–307. 1979.

[GKK$^+$01] C. Gavoille, M. Katz, N. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *Proceedings of ESA'2001*, 2001.

[GL99] I. Guérin-Lassous. *Algorithmes parallèles de traitement de graphes : une approche basée sur l'analyse expérimentale.* PhD thesis, LIAFA, Université Paris VII, 1999.

[GLT00] I. Guérin-Lassous and E. Thierry. Generating random permutations in the framework of coarse grained models. In *Proceedings of OPODIS'2000*, 2000.

[GNT87a] G. Gambosi, J. Nesestril, and M. Talamo. Posets, boolean representation and quick path searching. In *Proceedings of ICALP'87*, pages 404–424, 1987.

[GNT87b] G. Gambosi, J. Nesestril, and M. Talamo. Efficient representation of taxonomies. In *Proceedings of TAPSOFT'87*, pages 232–240, Vol.1 1987.

[GNT90] G. Gambosi, J. Nesestril, and M. Talamo. On locally presented posets. *Theoretical Comp. Sci.*, 70(2) :251–260, 1990.

[Gol80] M. C. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, 1980.

[GPPR01] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. In *Proceedings of SODA'2001*, pages 210–219, 2001.

[GR99] M.C. Golumbic and U. Rotics. On the clique-width of perfect graph classes. In *Proceedings of WG'99*, LNCS 1665, pages 135–147, 1999.

[GS01] V. K. Garg and C. Skawratananand. String realizers of posets with aplications to distributed computing. In *Proceedings of the 20th ACM Symposium in Principles of Distributed Computing (PODC'2001)*, pages 72–80, 2001.

[Hag98] T. Hagerup. Sorting and searching on the word ram. In D. Krob eds M. Morvan, C. Meinel, editor, *Proceedings of STACS'98*, LNCS 1373, pages 366–398. Springer, 1998.

[Hes01] W. Hesse. The dynamic complexity of transitive closure is in tc0. In *Proceedings of the Eighth International Conference on Database Theory (ICDT'2001)*, 2001.

[HHH+98] S.-Y. Hsieh, C.-W. Ho, T.-S. Hsu, M.-T. Ko, and G.-H. Chen. Characterization of efficiently solvable problems on distance-hereditary graphs. In *ISAAC'98*, pages 257–266, 1998.

[HHH+99] S.-Y. Hsieh, C.-W. Ho, T.-S. Hsu, M.-T. Ko, and G.-H. Chen. Efficient parallel algorithms on distance-hereditary graphs. *Parallel Processing Letters*, 9(1) :43–52, 1999.

[HHN95] M. Habib, M. Huchard, and L. Nourine. Embedding partially ordered sets into chain-products. In *Proceedings of KRUSE'95*, pages 147–161, 1995.

[Hir55] T. Hiragushi. On the dimension of orders. *Sci. Rep. Kanazawa Univ.*, 4 :1–20, 1955.

[HK73] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2 :225–231, 1973.

[HK95] M. Henzinger and V. King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of FOCS'95*, pages 664–672, 1995.

[HM90] P. Hammer and F. Maffray. Completely separable graphs. *Discrete Applied Mathematics*, 27 :85–99, 1990.

[HMPR91] M. Habib, M. Morvan, M. Pouzet, and J.-X. Rampon. Extensions intervallaires minimales. *C. R. Acad. Sci.*, I(313) :893–898, 1991.

[HN93] M. Habib and L. Nourine. Linear time recognition algorithm for distributive lattices. Technical report, LIRMM, 1993.

[HN94] M. Habib and L. Nourine. Bit-vector encoding for partially ordered sets. In *Proceedings of ORDAL'94*, LNCS 831, pages 1–12, 1994.

[HN95] M. Habib and L. Nourine. A new dimensional parameter and a removal theorem for boolean dimension. In *Proceedings of the 4th Twente Workshop on Graphs and Combinatorial Optimization*, pages 127–132, 1995.

[HN96] M. Habib and L. Nourine. Tree decomposition for distributive lattices and its applications. *Theoretical Computer Science*, 165(2) :391–405, 1996.

[HN97] M. Habib and L. Nourine. Liste de problèmes ouverts, 1997. LIRMM.

[HNR97] M. Habib, L. Nourine, and O. Raynaud. A new lattice-based heuristic for taxonomy encoding. In *Proceedings of KRUSE'97*, pages 60–71, 1997.

[How77] E. Howorka. A characterization of distance-hereditary graphs. *Quart. J. Math. Oxford*, Ser. 2, 28 :417–420, 1977.

[HT84] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13 :338–355, 1984.

[IK83] T. Ibaraki and N. Katoh. On-line computation of transitive closure for graphs. *Information Processing Letters*, pages 95–97, 1983.

[Ita86] G. F. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48(2-3) :273–281, 1986.

[Ita88] G. F. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Information Processing Letters*, (28) :5–11, 1988.

[Jag90] H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Transactions on Database Systems*, 15(4) :558–598, 1990.

[Jou95] G.-V. Jourdan. *L'analyse d'exécutions réparties en utilisant la théorie de l'ordre*. PhD thesis, Université de Rennes 1, 1995.

[Kie96] H. A. Kierstead. On the order dimension of 1-sets versus k-sets. *J. of Comb. Theory*, 73 Series A :219–228, 1996.

[Kin99] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of FOCS'99*, 1999.

[KMW96] S. Khanna, R. Motwani, and R. H. Wilson. On certificates and lookhead on dynamic graph problems. In *Proceedings of SODA'96*, pages 222–231, 1996.

[KNR92] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 5 :596–603, 1992.

[KR75] D. J. Kleitman and B. L. Rothschild. Asymptotic enumeration of partial orders. *Trans. of the American Mathematical Society*, 205, 1975.

[KR01] D. Kobler and U. Rotics. Polynomial algorithms for partitioning problems of graphs with fixed clique-width. In *Proceedings of SODA'2001*, pages 468–476, 2001.

[KS99] V. King and G. Sagert. A fully dynamic algorithm for maintaining the transitive closure. In *Proceedings of STOC'99*, pages 492–498, 1999.

[KVH97] A. Krall, J. Vitek, and R.N. Horspool. Near optimal hierarchical encoding of types. In *Proceedings of ECOOP'97*, pages 128–145, 1997.

[Lan01] J.-M. Lanlignel. *Autour de la décomposition en coupes.* PhD thesis, LIRMM, Université Montpellier II, 2001.

[LdR96] R. Lassaigne and M. de Rougemont. *Logique et Complexité.* Collection Informatique, Hermes, 1996.

[LRT00] J.-M. Lanlignel, O. Raynaud, and E. Thierry. Pruning graphs with digital search trees. application to distance hereditary graphs. In *Proceedings of STACS'2000*, LNCS 1770, pages 529–541, 2000.

[Lub90] A. Lubiw. The boolean basis problem and how to cover some polygons by rectangles. *SIAM Journal of Discrete Math.*, 3(1) :98–115, 1990.

[Mac37] H. M. MacNeille. Partially ordered sets. *Trans. Amer. Soc.*, 42 :416–460, 1937.

[Mar80] G. Markowski. The representation of posets and lattices by sets. *Algebra Universalis*, 11 :173–192, 1980.

[Mar92] G. Markowski. Prime, irreducibles and extremal lattices. *Order*, 9 :265–290, 1992.

[Mat89] F. Mattern. Virtual time and global states in distributed systems. In M. Cosnard et al, editor, *Parallel and Distributed Algorithms*, pages 215–226. Elsevier, 1989.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 2 : Graph Algorithms and NP-completness.* EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1984.

[MN93] M. Morvan and L. Nourine. Sur la distributivité du treillis des antichaînes maximales d'un ensemble ordonné. *C. R. Acad. Sci.*, t. 317-Série I :129–133, 1993.

[MN96] M. Morvan and L. Nourine. Simplicial elimination schemes, extremal lattices and maximal antichain lattices. *Order*, 13(2) :159–173, 1996.

[Moh89] R. H. Mohring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Orders*, volume 255 of *C*, pages 105–193. Kluwer Academic Publishers, 1989.

[MR89] R. H. Mohring and F. J. Radermacher. The order-theoretic approach to scheduling : the deterministic case. In R. Slowinski and J. Weglarz, editors, *Advances in project scheduling*, pages 29–66. Elsevier Science Publishers B. V., 1989.

[MS97] R. M. McConnell and J. Spinrad. Linear-time transitive orientation. In *Proceedings of SODA'97*, pages 19–25, 1997.

[Mul88] J. H. Muller. *Local Structure in Graphs*. PhD thesis, School of Infomation and Computer Science, 1988.

[Mül96] H. Müller. On edge perfectness and classes of bipartite graphs. *Discrete Mathematics*, 149 :159–187, 1996.

[Mun71] I. Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 52 :56–58, 1971.

[MW91] T. Madej and D. West. The interval inclusion number of partially ordered set. *Discrete Math.*, 88 :259–277, 1991.

[MW95] T. Madej and D. West. Interval number of special posets and random posets. *Discrete Math.*, 144 :67–74, 1995.

[NMWA78] D. S. Nau, G. Markowski, M. A. Woodbury, and D. B. Amos. A mathematical analysis of human leukocyte antigen serology. *Math. Biosciences*, 40 :243–270, 1978.

[Nou93] L. Nourine. *Quelques propriétés algorithmiques des treillis*. PhD thesis, LIRMM, Université Montpellier II, 1993.

[Nou00] L. Nourine. Une structuration algorithmique de la théorie des treillis. Habilitation à diriger des recherches, Université Montpellier II, 2000.

[Nov63] V. Novak. On the pseudo-dimension of ordered sets. *Czechoslovak Math. Journal*, 13 :587–598, 1963.

[NR99] L. Nourine and O. Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71 :199–204, 1999.

[NT00] L. Nourine and E. Thierry. A new proof of the *decreasing dimension theorem*. Technical report, LIRMM, 2000.

[NTV99] E. Nardelli, M. Talamo, and P. Vocca. A general framework for succint implicit encoding of graphs and orders. In *Preproceedings of ORDAL'99*, pages 15–23, 1999.

[Nuu95] E. Nuutila. *Efficient Transitive Closure Computation in Large Digraphs*. PhD thesis, Helsinki University of Technology, 1995.

[Oel94] O. Oellermann. Computing the average distance of a distance hereditary graph in linear time. *Congressus Numerantium*, 103 :219–223, 1994.

[Ore62] O. Ore. Theory of graphs. *AMS Colloq. Publ.*, 38, 1962.

[Orl76] J. Orlin. Contentment in graph theory : covering graphs with cliques. unpublished manuscript, 1976.

[OS95] O. Oellermann and J. Spinrad. On a polynomial algorithm for testing whether a graph is 3-steiner distance hereditary. *Inform. Proc. Letters*, 55 :149–154, 1995.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[Pau98] C. Paul. *Parcours en largeur lexicographique : un algorithme de partitionnement. Application aux graphes et generalisations*. PhD thesis, LIRMM, Université Montpellier II, 1998.

[Pha99] H. D. Phan. *Structures ordonnées et dynamiques de piles de sable*. PhD thesis, LIAFA, Université Paris VII, 1999.

[PR98] M. Pouzet and I. Rival. Structural arithmetic of the extension lattice of an order. To appear in Algebra Universalis, 1998.

[PRRZ95] M. Pouzet, K. Reuter, I. Rival, and N. Zaguia. A generalized permutahedron. *Algebra Universalis*, 34 :496–509, 1995.

[PvL88] J. A. La Poutré and J. van Leeuwen. Maintenance of transitive closure and transitive reduction of graphs. In *Proceedings of WG'88*, LNCS 314, pages 106–120, 1988.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization and complexity classes. *Journal of Computer and System Sciences*, pages 425–440, 1991.

[Ray97] O. Raynaud. Codage des graphes orientés sans circuit. Master's thesis, Université Montpellier 2, LIRMM, 1997.

[Ray00] O. Raynaud. *Codage et représentation des ensembles ordonnées*. PhD thesis, LIRMM, Université Montpellier II, décembre 2000.

[Rei85] J. H. Reif. An optimal parallel algorithm for integer sorting. In *26th Symposium on Foundations of Computer Science*, pages 496–503, 1985.

[Rem99] E. Remila. The lattice structure of the set of domino tilings of a polygon. In *Proceedings of ORDAL'99*, 1999.

[Reu91] K. Reuter. The jump number and the lattice of maximal antichains. *Discrete Mathematics*, 88 :289–307, 1991.

[RR79] I. Rabinovitch and I. Rival. The rank of distributive lattice. *Discrete Math.*, 25 :275–279, 1979.

[RT00] O. Raynaud and E. Thierry. Calcul de la 2-dimension d'un ensemble ordonné. Rapport de Recherche 00082, LIRMM, Montpellier, France, 2000.

[RT01] O. Raynaud and E. Thierry. A quasi optimal bit-vector encoding of tree hierarchies. application to efficient type inclusion tests. In *Proceedings of ECOOP'2001*, LNCS 2072, pages 165–180, 2001.

[Sim88] K. Simon. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Computer Science*, 58 :325–346, 1988.

[Sim90] H. U. Simon. On approximate solutions for combinatorial optimization problems. *SIAM J. Disc. Math.*, 3 :294–310, 1990.

[Sko92] M. Skorsky. *Endliche Verbände Diagramme und Eigenschaften.* PhD thesis, Darmstad, Germany, 1992. (écrit en anglais).

[Spi] J. Spinrad. Liste de problèmes ouverts en théorie des graphes et des ordres. http ://www.vuse.vanderbilt.edu/ spin/open.html.

[Spi94] J. Spinrad. Dimension and algorithms. In *Proceedings of ORDAL'94*, LNCS 831, pages 33–52, 1994.

[Spi97] J. Spinrad. *Efficient Graph Representations.* http ://www.vuse.vanderbilt.edu/∼spin/research.html, 1997.

[SPT83] L. K. Schubert, M. A. Papalaskaris, and J. Taugher. Determining type, part, color and time relationships. *Computer*, 16 :53–60, 1983. Special issue on Knowledge Representation.

[ST91] H. Shum and L. E. Trotter. Cardinality-restricted chains and antichains in partially ordered sets. Technical report, Cornell University, 1991.

[Sta71] R. Stanley. *Ordered structures and partitions.* PhD thesis, Harvard University, 1971.

[Sta72] R. Stanley. Ordered structures and partitions. *Mem. Amer. Math. Soc.*, 119, 1972.

[Sta84] J. Stahl. On the 2-dimension of the crown s_k^n. Technical Report 800, TH Darmstad, 1984.

[SV88] B. Schieber and U. Vishkin. On finding lowest common ancestors : simplification and parallelization. *SIAM Journal on Computing*, 17 :1253–1262, 1988.

[SW84] J. Stahl and R. Wille. Preconcepts and set representation of contexts. In *Proc. Universal Algebra*, Sienna, 1984.

[Szp30] E. Szpilrajn. Sur l'extension de l'ordre partiel. *Fund. Math.*, 60 :175–186, 1930.

[Tar72] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2) :146–160, 1972.

[Tro75] W. T. Trotter. Embedding finite posets in cubes. *Discrete Mathematics*, 12 :165–172, 1975.

[Tro91] W. T. Trotter. *Combinatorics and Partially Ordered Sets : Dimension Theory.* John Hopkins University Press, Baltimore, 1991.

[Tro95] T. Trotter. Graphs and partially ordered sets : recent results and new directions. Technical report, Arizona state university, 1995.

[TV97] M. Talamo and P. Vocca. A data structure for lattice representation. *Theoretical Computer Science*, 175 :373–392, 1997.

[Val90] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8) :103–111, 1990.

[VHK97] J. Vitek, R.N. Horspool, and A. Krall. Efficient type inclusion tests. In *OOPSLA'97*, pages 142–157, 1997.

[Weg87] I. Wegener. *The complexity of boolean functions.* Wiley-Teubner, 1987.

[Wes85] D. B. West. Parameters of partial orders and graphs : packing, covering and representation. In I. Rival, editor, *Graphs and Orders*, pages 267–350. NATO, D. Reidel publishing company, 1985.

[Wes01] D. West. *Introduction to Graph Theory.* Prentice Hall, second edition edition, 2001. 1st edition in 1996.

[Wil82] R. Wille. Restructuring lattice theory : an approach based on hierarchies of contexts. In I. Rival, editor, *Ordered Sets*, number 83 in NATO ASI, pages 445–470. 1982.

[Wil83] R. Wille. Subdirect decomposition of concept lattices. *Algebra Universalis*, 17 :275–287, 1983.

[Yan82] M. Yannakakis. The complexity of partial order dimension problem. *SIAM J. Alg. Disc. Math.*, 3 :351–358, 1982.

[Yel93] D. M. Yellin. Speeding up dynamic transitive closure for bounded degree graphs. *Acta Informatica*, (30) :369–384, 1993.

[ZG01] Y. Zibin and Y. Gil. Efficient subtyping tests with pq-encoding. In *Proceedings of OOPSLA'2001*, pages 96–107, 2001.