UNIVERSITE D'AIX-MARSEILLE

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET
D'INFORMATIQUE DE MARSEILLE

FACULTE DES SCIENCES

LABORATOIRE DES SCIENCES DE L'INFORMATION ET DES SYSTÈMES

THÈSE

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ D'AIX-MARSEILLE

Discipline : Informatique

par

Achref EL MOUELHI

Classes polynomiales pour CSP : de la théorie à la pratique

Soutenue le 03 Décembre 2014

Composition du jury :

Philippe JÉGOU	Université d'Aix-Marseille	Directeur de thèse
Christophe LECOUTRE	Université d'Artois	Rapporteur
Thomas SCHIEX	INRA	Rapporteur
Christine SOLNON	INSA de Lyon	Examinatrice
Cyril TERRIOUX	Université d'Aix-Marseille	Directeur de thèse

Remerciements

Je tiens, tout d'abord, à remercier Monsieur Christophe LECOUTRE et Monsieur Thomas SCHIEX pour avoir eu la patience de relire ce mémoire malgré leur emploi du temps très chargé et Madame Christine SOLNON d'avoir accepté de participer à mon jury.

Je voudrais aussi exprimer toute ma reconnaissance à mon directeur de thèse Philippe JÉGOU pour les conditions de travail que j'ai eues, le temps qu'il m'a consacré et la possibilité qu'il m'a offerte de travailler dans le projet ANR-TUPLES en collaboration avec des chercheurs d'autres laboratoires de recherche.

Je remercie tout autant mon co-directeur de thèse Cyril TERRIOUX pour ses conseils et ses critiques qui m'ont guidés durant mes années de recherche et pour sa disponibilité, il a eu souvent le temps pour répondre à mes questions.

Je remercie également les membres de l'équipe de recherche INCA, les permanents et les doctorants, ainsi que tous les chercheurs et les personnels du laboratoire LSIS particulièrement Véronique BIANCIOTTO, Nadine LATTANZIO et Sandrine DULAC.

Je souhaite vivement remercier tous les membres du projet ANR-TUPLES pour les discussions très intéressantes que j'ai pu avoir avec certains d'entre eux, en particulier Martin COOPER et Bruno ZANUTTINI.

J'adresse mes sincères remerciements à mes parents, Ahmed et Fatma, et mes frères qui ont toujours été là pour moi. Je dois également remercier les familles « EL MOUELHI », « HAJJI » et « MAHJOUBI ».

Un grand merci à ma chère Zineb, pour son aide, sa patience et sa participation à la rédaction de ce manuscrit, et à ma petite princesse Hafsa.

Enfin, je tiens à remercier mes amis Farès, Ali, Issam, Mehdi, Ramzi et Wael pour leurs encouragements.

Table des matières

Table des figures

Liste des tableaux

Liste de notations

- $\mathcal{A} = (v_1, \ldots, v_k)$ est l'affectation des valeurs $(v_1, \ldots, v_k)$ aux variables $\{x_1, ..., x_k\}$. page 21
- (B, T) est une décomposition arborescente de $G = (V, E)$. page 48
  - $T = (I, F)$ un arbre. page 48
  - $B = \{B_i : i \in I\}$ est un ensemble de clusters. page 48
  - w est la largeur arborescente de (B, T). page 48
- BTP est l'ensemble des CSP dont les relations satisfont Broken Triangle Property. page 81
- BTW_k est la l'ensemble des CSP ayant une largeur arborescente bornée par une constante k. page 48
- $\mathcal{C}$ est une classe polynomiale contenant un ensemble de CSP. page 40
  - A_S est l'algorithme de résolution des CSP de $\mathcal{C}$. page 40
  - A_R est l'algorithme de reconnaissance des CSP appartenant à $\mathcal{C}$. page 40
- CCM est l'ensemble des CSP ayant un complément de microstructure triangulé. page 42
- CL l'ensemble des CSP binaires ayant un nombre polynomial de cliques maximales. page 81
- CM est l'ensemble des CSP ayant une microstructure triangulée. page 42
- CSG^0 est l'ensemble de graphe complets. page 43
- CSG^k est une généralisation des graphes triangulés. page 43
- $D(P) = (V, E)$ est le graphe dual de P. page 59
- $DBTP$ est l'ensemble des CSP dont le dual satisfait BTP. page 112
- DCL l'ensemble des CSP dont la DR-microstructure possède un nombre polynomial de cliques maximales. page 83
- $DZOA$ est l'ensemble des CSP dont le dual satisfait ZOA. page 72
- $G(P) = (V, E)$ est le graphe de contraintes de P. page 23
- $G^p(P) = (V^p, E^p)$ est le graphe primal de P. page 24
- $\overline{G} = (V, \overline{E})$ est le complément de $G = (V, E)$. page 26
- $GM(P) = (V, E)$ est le graphe mixte de P. page 67
- $GMM(P) = (V, E)$ est le graphe mixte minimal de P. page 69
- h est la largeur hyperarborescente de HD = (T,χ,λ). page 49
- $H(P) = (V, E)$ est l'hypergraphe de contraintes de P. page 23
- $IFUN$ est l'ensemble des CSP dont les relations sont incrémentalements fonctionnels. page 51
- k-clique est une clique de taille k. page 26
- MC est l'ensemble des CSP dont les relations satisfont max-closed. page 52
- MME est l'ensemble des CSP dont les relations satisfont min-of-max extendable. page 55
- $N_{BT}(P)$ est le nombre de nœuds figurant dans l'arbre de recherche développé par BT. page 80
- $N_{FC}(P)$ est le nombre de nœuds figurant dans l'arbre de recherche développé par FC. page 80
- $N_{nBT}(P)$ est le nombre de nœuds figurant dans l'arbre de recherche développé par nBT.

page 82
- $N_{nFC_i}(P)$ est le nombre de nœuds figurant dans l'arbre de recherche développé par nFC_i. page 82
- $P = (X, D, C)$ est un problème CSP. page 18
  - $X = \{x_1, ..., x_n\}$ est l'ensemble de variables de P. page 18
  - n est le nombre de variables. page 18
  - $D = \{D(x_1), ..., D(x_n)\}$ est un ensemble de domaines associés aux variables de P. page 18
  - d est la cardinalité maximale de tous les domaines. page 18
  - $C = \{c_1, ..., c_e\}$ est un ensemble de e contraintes. page 18
  - e est le nombre de contraintes. page 18
  - $S(c_i)$ est le scope d'une contrainte. page 18
  - a est l'arité d'une contrainte. page 18
  - $R(c_i)$ est la relation d'une contrainte. page 18
  - r est le nombre de tuples d'une contrainte. page 18
  - t_i est un tuple de la relation $R(c_i)$. page 18
- P[$\{x_{i_1}, ..., x_{i_k}\}$] est la restriction du problème P aux variables $\{x_{i_1}, ..., x_{i_k}\}$. page 21
- $P^d = (X^d, D^d, C^d)$ est le CSP dual de P. page 59
- $P^h = (X^h, D^h, C^h)$ est le CSP caché de P. page 63
- $P^m = (X^m, D^m, C^m)$ est le CSP mixte de P. page 66
- $P^M = (X^M, D^M, C^M)$ est le CSP mixte minimal de P. page 69
- $P^p = (X^p, D^p, C^p)$ est le CSP primal de P. page 24
- PM est l'ensemble des CSP ayant une microstructure parfaite. page 44
- R(c_i)[$\{x_{i_1}, ..., x_{i_k}\}$] est la restriction de la relation $R(c_i)$ aux variables $\{x_{i_1}, ..., x_{i_k}\}$. page 21
- RC est l'ensemble des CSP dont les relations sont convexes par rangée. page 52
- RRM est l'ensemble des CSP dont les relations sont renommables monotones à droite. page 53
- (T,χ,λ) est un hyperarbre de $H = (V, E)$. page 49
- $t(P) = (T_{var}(X), T_{dom}(D), T_{cons}(C))$ est la transformation de P par T. page 93
- $t(\mathcal{P})$ est l'ensemble des CSP $\mathcal{P}$ trnsformés par T. page 94
- $t_i[\{x_{i_1}, ..., x_{i_k}\}]$ est la restriction de tuple t_i aux variables $\{x_{i_1}, ..., x_{i_k}\}$. page 21
- TR est l'ensemble des CSP dont toutes les relations sont triangulaires. page 125
- $VC(P) = (V, E)$ est la représentation par variables cachées de P. page 64
- (v'_k, v_i, v_j, v''_k) est un triangle cassé sur la variable x_k. page 96
- ZOA est l'ensemble des CSP dont les relations satisfont 0/1/tous. page 50
- $\mu_{DR}(P) = (V, E)$ est le graphe de microstructure basé sur le codage dual. page 59
- $\mu_{DSR}(P) = (V, E)$ est le graphe de microstructure basé sur un intergraphe. page 62
- $\mu_{HT}(P) = (V, E)$ est le graphe de microstructure basé sur le codage caché. page 64
- $\mu_{ME}(P) = (V, E)$ est le graphe de microstructure basé sur le codage mixte. page 67
- $\mu_{MME}(P) = (V, E)$ est le graphe de microstructure basé sur le codage mixte minimal. page 70
- $\mu(P) = (V, E)$ est le graphe de microstructure de P. page 25
- $\sigma = [v_1, ..., v_n]$ un schéma parfait d'élimination d'un ensemble de sommets. page 43

Chapitre 1

Introduction

L'Intelligence Artificielle se subdivise en plusieurs sous-disciplines traitant chacune des problématiques différentes et dont le but est de réaliser des tâches qui sont accomplies de façon plus satisfaisante par des êtres humains. Parmi ces sous-disciplines, nous pouvons citer notamment : la représentation de connaissances et le raisonnement logique, le traitement du langage naturel, les systèmes multi-agents, la programmation par contraintes,...

En ce qui nous concerne, nous nous intéressons à la programmation par contraintes. Ce domaine, qui permet d'exprimer un problème sous forme d'un ensemble de contraintes, présente depuis plusieurs années un cadre général pour la formalisation et la résolution efficace de nombreux problèmes relevant de l'intelligence artificielle et de la recherche opérationnelle.

Le problème de satisfaction de contraintes (ou bien CSP pour Constraint Satisfaction Problem) désigne le formalisme permettant de modéliser l'ensemble des CSP pouvant être exprimés par des contraintes. Un CSP consiste en un ensemble de variables ayant chacune un ensemble discret de valeurs. Ces variables sont inter-connectées par des contraintes qui définissent l'ensemble des valeurs compatibles ou incompatibles. Quand une contrainte relie seulement deux variables, elle est dite binaire, sinon elle est non-binaire. Un CSP est dit binaire, si toutes ses contraintes sont binaires, sinon il est non-binaire. Résoudre un problème CSP (on dit aussi trouver une solution) équivaut à affecter une valeur à chaque variable sans violer une seule contrainte. Le problème de décision, qui consiste à tester pour un CSP si une solution existe, est NP-complet et nous ne connaissons donc à ce jour aucun algorithme de complexité polynomiale qui résoudrait ce problème dans tous les cas. Pour la résolution pratique de ce problème, de nombreuses approches ont été proposées. Ces approches, qui sont souvent implémentées dans des solveurs, sont basées sur des algorithmes de recherche ayant une complexité temporelle exponentielle. L'amélioration de l'efficacité de ce type d'algorithmes s'appuie généralement sur des techniques de filtrages opérés lors de la recherche (en plus d'autres techniques telles que les heuristiques de choix de variables). Avec l'aide de ces techniques, les solveurs, en dépit de leur complexité théorique exponentielle, s'avèrent très efficaces dans la pratique sur un large éventail de problèmes réels.

Dans une direction orthogonale, d'autres travaux portent sur l'efficacité de la résolution des CSP en définissant des classes polynomiales. Une classe polynomiale est un sous-ensemble d'instances CSP pouvant être reconnues, puis résolues en utilisant des algorithmes de complexité polynomiale. Différents types de classes polynomiales ont été introduites. Malheureusement, la plupart des classes polynomiales se présentent rarement en pratique, ce qui contribue à élargir le fossé entre théorie et pratique. En revanche, comme évoqué ci-dessus, les solveurs, dont la complexité théorique est exponentielle, sont à la base de systèmes pratiques pour la résolution de contraintes, et leurs résultats pratiques sont souvent impressionnants en termes de temps de calcul, alors même qu'ils n'exploitent a priori aucune classe polynomiale.

Dans cette thèse, nous essayons d'amoindrir le fossé existant entre les travaux théoriques sur les classes polynomiales et l'efficacité pratique des méthodes usuelles, cela en tentant de fournir des éléments de réponse à la question portant sur les raisons de l'efficacité observée pour les solveurs. Dans ce but, nous allons étudier des classes polynomiales qui sont implicitement exploitées par les solveurs, c'est-à-dire des classes polynomiales dont les CSP seront résolus en temps polynomial par des algorithmes classiques comme ceux qui sont utilisés dans les solveurs. De façon complémentaire, nous étudierons les algorithmes de base des solveurs pour voir dans quels cas ils peuvent être polynomiaux. Nos travaux couvriront les CSP d'arité quelconque (binaires et non-binaires) pour mieux se rapprocher des applications réelles.

Cette démarche nous a conduit, dans un premier temps, à étudier la classe polynomiale BTP (pour Broken Triangle Property [Cooper et al., 2010]) proposée pour les CSP binaires et qui dispose de certaines propriétés intéressantes. En effet, BTP capte plusieurs autres classes polynomiales telles que les CSP binaires arborescents et les CSP binaires renommables monotone à droite. En plus, BTP est close pour le filtrage de domaines, c'est-à-dire que l'ensemble des CSP qui sont BTP le resteront après application de ce type de filtrage. En outre, les CSP satisfaisant BTP peuvent être résolus par des algorithmes de résolution de type MAC ou RFL. Ces deux algorithmes maintiennent un niveau de cohérence, dit cohérence d'arc, à chaque étape de la résolution. Dans un second temps, nous étudierons les classes polynomiales qui pourraient être résolues en temps polynomial par des algorithmes qui maintiennent, d'une façon plus générale, un niveau de filtrage donné pendant la recherche d'une solution.

Au niveau binaire, plusieurs classes polynomiales pour CSP ont été définies par l'identification des restrictions particulières sur la microstructure [Jégou, 1993a]. Malheureusement, nous ne disposons pas d'un tel outil théorique pour le cas non-binaire puisque la première proposition de microstructure pour les CSP d'arité quelconque s'avère inexploitable car elle s'appuie sur le complément des hypergraphes [Cohen, 2003]. Dans le chapitre 3, nous proposerons des microstructures pour les CSP d'arité quelconque. Ces nouvelles microstructures seront basées sur des codages binaires étudiés précédemment dans la littérature. Comme pour le cas binaire, nos microstructures seront bien des graphes et pas des hypergraphes. Nous montrerons que ces microstructures seront utiles pour étendre certaines classes polynomiales comme ZOA (pour 0-1-tous [Cooper et al., 1994]) définies initialement pour le cas binaire, aux CSP d'arité quelconque.

Dans le chapitre 4, nous proposerons une nouvelle analyse de complexité des algorithmes de résolutions classiques de type BT, FC, RFL, nBT, nFC et nRFL basée sur le nombre de cliques maximales dans les microstructures. Nous montrerons que la taille de l'arbre de recherche développé par ces algorithmes peut être bornée par le nombre de cliques maximales dans la microstructure. Pour le cas non-binaire, nous montrerons qu'en utilisant une des microstructures proposées dans le chapitre 3 nous pourrons avoir des résultats similaires aux cas binaires. En se référant à la théorie des graphes, nous illustrerons certaines classes de graphes ayant un nombre polynomial de cliques maximales.

Dans le chapitre 5, nous introduirons un nouveau concept, basée sur la transformation des CSP, afin de capturer les benchmarks qui sont dans le voisinage des classes polynomiales. Comme les problèmes réels n'appartiennent pas, en général, à des classes polynomiales connues, nous utiliserons le filtrage par cohérence comme moyen de transformation des CSP pour prouver leur appartenance à des classes polynomiales dites cachées. En nous focalisant sur la classe BTP, nous montrerons que notre étude permet d'avoir des résultats très intéressants en pratique. Par la suite, nous prouverons que BTP peut être également utilisée pour réduire la taille des domaines par la fusion des valeurs respectant certaines conditions. Ainsi, nous aurons aussi des résultats importants sur la fusion par BTP et qui paraissent être plus intéressants que les résultats obtenus par l'application des méthodes précédentes (par exemple la substituabilité [Freuder, 1991]).

Dans le chapitre 6, nous proposerons une extension de BTP (notée DBTP) aux CSP d'arité quel-

conques basée sur une des microstructures proposées dans le chapitre 3. Cette nouvelle extension permet d'avoir des résultats similaires à ceux de BTP. En effet, BTP permet dans le cas binaire de capturer les CSP binaires arborescentes et DBTP permet de capturer dans le cas général les CSP β-acycliques. D'un point de vue résolution, DBTP, comme BTP, est une classe close et l'ensemble des CSP y figurant seront résolus par des algorithmes classiques de type MAC ou RFL en temps polynomial. BTP et DBTP sont très semblables en terme de résultats théoriques, mais DBTP ne présente pas une généralisation de DBTP et nous montrerons qu'elles sont différentes dans le cas binaire. Pour mettre en valeur cette extension, nous réaliserons une étude expérimentale pour montrer que DBTP n'est pas une classe artificielle.

Première partie

État de l’art

Chapitre 2

État de l'art

2.1 Introduction

Le problème CSP en particulier et la programmation par contraintes en général, proposent un cadre général pour modéliser et résoudre une grande diversité de problèmes réels (problèmes d'allocation de fréquences, problèmes d'ordonnancement...) et des problèmes académiques (problèmes de pigeons, problèmes de n reines...). Ils disposent d'une littérature relativement large et riche [Jégou, 1991, Dechter, 1992, Tsang, 1993, Rossi et al., 2006], ce qui rend difficile de tout détailler dans ce manuscrit. Pour cette raison, nous rappellerons seulement le nécessaire pour faciliter la compréhension de notre contribution.

Dans ce chapitre, nous expliquerons comment le formalisme CSP permet de modéliser et résoudre efficacement plusieurs types de problèmes rencontrés en intelligence artificielle et plus généralement en informatique. Nous débuterons en définissant formellement le problème CSP et nous rappellerons quelques notations et notions de base (section 2.2). Ensuite, nous parlerons des différentes représentations graphiques des CSP et nous mettrons en particulier l'accent sur la notion de microstructure de CSP binaire (section 2.3). Après, nous illustrerons quelques méthodes utilisées pour réduire la taille des domaines et/ou des relations : ces méthodes s'appuient sur la notion de filtrage par cohérence (section 2.4). Puis, nous présenterons les algorithmes de résolution classiques ainsi que leurs complexités (section 2.5). Nous évoquerons aussi la notion d'heuristiques de choix de variables utilisées pour guider les algorithmes de résolution au cours de la recherche d'une solution (section 2.5.5). Enfin, nous introduirons la notion de classe polynomiale et nous détaillerons les principales classes étudiées dans la littérature (section 2.6).

2.2 Le Problème de Satisfaction de Contraintes

Le *problème de satisfaction de contraintes* (CSP, [Montanari, 1974]) constitue un formalisme important pour exprimer et résoudre efficacement un large éventail de problèmes réels. Formellement, un CSP est défini comme suit :

Définition 2.1. *Un CSP est un triplet $P = (X, D, C)$, où $X = \{x_1, ..., x_n\}$ est un ensemble de **variables**, $D = \{D(x_1), ..., D(x_n)\}$ est un ensemble de **domaines** finis de **valeurs**, un pour chaque variable et $C = \{c_1, ..., c_e\}$ est un ensemble fini de **contraintes**. Chaque contrainte c_i est un couple $(S(c_i), R(c_i))$. $S(c_i) = \{x_{i_1}, ..., x_{i_{a_i}}\} \subseteq X$, appelé la **portée** (le **scope**) de la contrainte, définit l'ensemble des variables sur lesquelles porte la contrainte c_i .*

*Une **relation** $R(c_i)$ définit la compatibilité entre les valeurs appartenant aux domaines des variables figurant dans la porté de la contrainte c_i ($R(c_i) \subseteq D(x_{i_1}) \times ... \times D(x_{i_{a_i}})$). Elle autorise un ensemble de **tuples** (un tuple de la relation $R(c_i)$ sera noté t_i) telle que chaque tuple est égal à une combinaison de valeurs admissible par la contrainte. Nous supposons que toute variable apparaît au moins dans la portée d'une contrainte. $|S(c_i)|$ est l'**arité** de la contrainte notée a_i (c'est-à-dire, le nombre de variables sur lesquelles la contrainte c_i porte). Si la contrainte est d'arité deux, alors elle est dite binaire et elle sera notée c_{ij} avec $S(c_{ij}) = \{x_i, x_j\}$.*

Si toutes les contraintes d'un CSP P sont binaires alors P est dit **binaire**. Sinon (cas général), P est dit **n-aire** (**non-binaire** ou **d'arité quelconque**). Principalement, il existe deux façons pour exprimer les contraintes :

- **contrainte en intension :** c'est une contrainte exprimée par une formule (comme $x_1 + x_2 < x_3$) ou un prédicat comme $eq(x_1, x_2)$ (pour $x_1 = x_2$), ...

- **contrainte en extension :** c'est une contrainte (dite aussi *contrainte table*) exprimée explicitement par une relation indiquant soit l'ensemble des tuples autorisés soit l'ensemble des tuples interdits (dit aussi l'ensemble de nogoods). Par exemple, la relation $R(c_i)$ de la contrainte ternaire c_i autorise l'ensemble des combinaisons suivant {(0,0,1), (1,0,0), (0,1,0)}.

Une contrainte en intension ou en extension peut être exprimée dans certains cas par une contrainte dite *globale* : c'est une contrainte qui fait référence à un motif bien connu comme AllDif [Régin, 1994] (qui exige que les variables de la portée de la contrainte aient des valeurs deux à deux différentes), Element, atMost, NValue, NotAllEqual,...

La taille d'un CSP P (notée $|P|$) dépend de la taille et de la nature de contraintes et de certains paramètres du problème n, d, e, a et r où :
- n est le nombre de variables du CSP,
- d est la cardinalité du plus grand domaine,
- e est le nombre de contraintes,
- a est l'arité maximale de toutes les contraintes,
- r est le nombre de tuples maximal de toutes les relations.

La taille d'un CSP P dont les contraintes sont données en extension est :
$n + e + \sum_{i=1}^{n} |D(x_i)| + \sum_{j=1}^{e} |S(c_j)| + \sum_{k=1}^{e} |R(c_k)|$ avec :
- $|D(x_i)| \leq d$,
- $|S(c_j)| \leq a$ et
- $|R(c_k)| \leq d^a$.

Donc, $|P|$ est de l'ordre de $O(n + e + nd + ae + ed^a) \simeq O(nd + ae + ed^a)$.

Nous pouvons remarquer que la taille d'un CSP dont les contraintes sont données en extension est polynomiale en fonction de l'arité maximale de toutes les contrainte a. Pour le cas binaire, la taille d'un CSP est toujours polynomiale vu qu'elle est de l'ordre de
$O(n + e + nd + 2e + ed^2) \simeq O(nd + 3e + ed^2)$.

Pour les CSP dont les contraintes sont données en intension, le calcul de la taille du CSP, dans certains cas, parait être plus compliqué car nous ne pouvons pas estimer le nombre de tuples de chaque relation vu que les contraintes sont exprimées par des fonctions ou des prédicats. Par contre, nous pouvons l'exprimer à l'aide d'une fonction $f_k()$ que nous utilisons pour noter la taille d'une contrainte c_k.

$|P| = n + e + \sum_{i=1}^{n} |D(x_i)| + \sum_{j=1}^{e} |S(c_j)| + \sum_{k=1}^{e} |f_k()|$ avec $f_k()$ la fonction correspondante au codage de la contrainte c_k.

Nous allons illustrer trois exemples pour mettre en évidence les différentes notations précédentes (les trois exemples ont les mêmes ensembles de variables et de domaines). Les deux premiers exemples seront utilisés tout au long de ce manuscrit.

Exemple 2.1. *Le premier exemple est un CSP non-binaire $P_1 = (X_1, D_1, C_1)$ avec six variables et quatre contraintes en extension.*

- $X_1 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$
- $D_1 = \{D(x_1), D(x_2), D(x_3), D(x_4), D(x_5), D(x_6)\}$
- $D(x_1) = \{a, a'\}$, $D(x_2) = \{b, b'\}$, $D(x_3) = \{c\}$, $D(x_4) = \{d, d'\}$, $D(x_5) = \{e, e'\}$ *et* $D(x_6) = \{f\}$
- $C_1 = \{c_1, c_2, c_3, c_4\}$
  - $S(c_1) = \{x_1, x_2\}$,
  - $S(c_2) = \{x_2, x_3, x_5\}$,
  - $S(c_3) = \{x_3, x_4, x_5\}$ *et*
  - $S(c_4) = \{x_2, x_3, x_6\}$.

$R(c_1)$	
x_1	x_2
a	*b*
a'	*b'*

$R(c_2)$		
x_2	x_3	x_5
b	*c*	*e*

$R(c_3)$		
x_3	x_4	x_5
c	*d*	*e*
c	*d'*	*e'*

$R(c_4)$		
x_2	x_3	x_6
b	*c*	*f*
b'	*c*	*f*

Exemple 2.2. *Le deuxième exemple est un CSP binaire $P_2 = (X_2, D_2, C_2)$ avec six variables et huit contraintes en extension.*

- $X_2 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$
- $D_2 = \{D(x_1), D(x_2), D(x_3), D(x_4), D(x_5), D(x_6)\}$
- $D(x_1) = \{a, a'\}$, $D(x_2) = \{b, b'\}$, $D(x_3) = \{c\}$, $D(x_4) = \{d, d'\}$, $D(x_5) = \{e, e'\}$ *et* $D(x_6) = \{f\}$.
- $C_2 = \{c_{12}, c_{23}, c_{25}, c_{26}, c_{34}, c_{35}, c_{36}, c_{45}\}$

$R(c_{12})$	$R(c_{23})$	$R(c_{25})$	$R(c_{26})$	$R(c_{34})$	$R(c_{35})$	$R(c_{36})$	$R(c_{45})$
x_1 x_2	x_2 x_3	x_2 x_5	x_2 x_6	x_3 x_4	x_3 x_5	x_3 x_6	x_4 x_5
a b	*b c*	*b e*	*b f*	*c d*	*c e*	*c f*	*d e*
a' b'			*b' f*	*c d'*	*c e'*		*d' e'*

Exemple 2.3. *Le troisième exemple est un CSP non-binaire $P_3 = (X_3, D_3, C_3)$ avec six variables et quatre contraintes en intention.*

- $X_3 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$
- $D_3 = \{D(x_1), D(x_2), D(x_3), D(x_4), D(x_5), D(x_6)\}$
- $D(x_1) = D(x_2) = D(x_3) = D(x_4) = D(x_5) = D(x_6) = \{0, 1, ..., 99\}$
- $C_3 = \{c_1, c_2, c_3, c_4\}$
  - $S(c_1) = \{x_1, x_2\}$,
  - $S(c_2) = \{x_2, x_3, x_5\}$,
  - $S(c_3) = \{x_3, x_4, x_5\}$,
  - $S(c_4) = \{x_2, x_3, x_6\}$.,
  - $R(c_1) = neq(x_1, x_2)$,
  - $R(c_2) = neq(x_2, x_3, x_5)$,
  - $R(c_3) = neq(x_3, x_4, x_5)$ *et*
  - $R(c_4) = neq(x_2, x_3, x_6)$.

Dans l'exemple 2.3, le motif neq est utilisé pour que deux variables dans la portée d'une contrainte n'aient pas la même valeur. Nous signalons aussi que toutes les contraintes de cet exemple peuvent être exprimées par des contraintes globales de type AllDif.

Théoriquement, toute contrainte en intention pourrait être transformée en une contrainte en extension. Mais en pratique, ceci est, peut être, difficile à réaliser pour des raisons de mémoire. Il est bien clair que la représentation des contraintes en intension ou par des contraintes globales permet d'éviter de lister toutes les combinaisons des valeurs autorisées, ce qui engendrerait une utilisation prohibitive de la mémoire. Dans l'exemple 2.3, chaque valeur de x_1 est en relation avec 99 valeurs de x_2, pour la représenter en extension, il faudrait donc un espace mémoire très important et aussi du temps pour calculer pour chaque contrainte l'ensemble de valeurs compatibles.

Maintenant, nous illustrons quelques notations importantes pour la suite. Nous commençons par les notations qui concernent les restrictions sur les tuples et les relations.

Notation 2.1 (projection d'un tuple)**.** *Étant donné un tuple t_i :*
$t_i[\{x_{i_1}, ..., x_{i_k}\}] = (v_j \in t_i \mid x_j \in \{x_{i_1}, ..., x_{i_k}\})$ *est la projection du tuple t_i sur les variables de $\{x_{i_1}, ..., x_{i_k}\}$ (on dit aussi la restriction du tuple t_i aux variables $\{x_{i_1}, ..., x_{i_k}\}$).*

Notation 2.2 (projection d'une relation)**.** $R(c_i)[\{x_{i_1}, ..., x_{i_k}\}] = \{t[\{x_{i_1}, ..., x_{i_k}\}] \mid t \in R(c_i)\}$ *est dite la projection de la relation $R(c_i)$ sur les variables de $\{x_{i_1}, ..., x_{i_k}\}$ (on dit aussi la restriction de la relation $R(c_i)$ aux variables $\{x_{i_1}, ..., x_{i_k}\}$). .*

Pour la relation $R(c_3)$, si on note t_3 le tuple (c, d, e). $t_3[\{x_3, x_4\}] = (c, d)$, $t_3[\{x_3, x_5\}] = (c, e)$ et $t_3[\{x_5\}] = (e)$. La restriction de la relation $R(c_3)$ aux variables $\{x_3, x_4\}$ et $\{x_3, x_5\}$ est donnée par les deux tableaux suivants :

$R(c_3)[\{x_3, x_4\}]$	
x_3	x_4
c	d
c	d'

$R(c_3)[\{x_3, x_5\}]$	
x_3	x_5
c	e
c	e'

Notation 2.3. *Étant donné un CSP $P = (X, D, C)$, le CSP $P' = (X', D', C')$ obtenue par la restriction de P aux variables $\{x_{i_1}, ..., x_{i_k}\}$ (qui sera noté $P' = P[\{x_{i_1}, ..., x_{i_k}\}]$) est défini comme suit :*
- $X' = \{x_{i_1}, ..., x_{i_k}\}$,
- $D' = \{D(x_i) \in D \mid x_i \in X'\}$,
- $C' = \{c' | c \in C$ *tel que* $S(c) \cap X' \neq \emptyset, S(c') = S(c) \cap X'$ *et* $R(c') = R(c)[S(c')]\}$

Si on note $P'_1 = (X', C', D')$ la restriction du CSP P_1 aux variables $\{x_3, ..., x_6\}$. $P'_1 = P_1[\{x_3, ..., x_6\}]$ est défini comme suit :
- $X' = \{x_3, x_4, x_5, x_6\}$,
- $D' = \{D(x_3), D(x_4), D(x_5), D(x_6)\}$,
- $C' = \{c_2, c_3, c'_4\}$.

Dans P_1, la contrainte c_4 sera modifiée. $S(c'_4) = \{x_3, x_6\}$ et $R(c'_4) = R(c_4)[\{x_3, x_6\}]$.

Définition 2.2 (jointure)**.** *La jointure entre les relations $R(c_1), R(c_2), ..., R(c_k)$ (notée $R(c_1) \bowtie R(c_2) \bowtie ... \bowtie R(c_k)$) est égale $R(S(c_1) \cup S(c_2) \cup ... \cup S(c_k))$ telle que $\forall t \in R(c_1) \bowtie R(c_2) \bowtie ... \bowtie R(c_k)$, $\forall i, 1 \leq i \leq k$, $t[S(c_i)] \in R(c_i)$.*

Résoudre un problème CSP revient à affecter une valeur à chaque variable du problème sans violer les contraintes. Formellement, ceci est détaillé dans les définitions suivantes :

Définition 2.3 (affectation)**.** *Étant donné un CSP (X, D, C), une **affectation** de valeurs (notée généralement $\mathcal{A}$) à un ensemble de variables $Y (\subseteq X)$ est un ensemble de paires $\mathcal{A} = \{(x_i, v_i) \mid x_i \in Y$ et $v_i \in D(x_i)\}$ (que l'on écrira sous forme de tuple $\mathcal{A} = (v_1, \ldots, v_k)$ quand aucune confusion ne se présente).*

Dans la littérature, nous trouvons différentes classifications d'affectations. En fonction de la taille de Y, nous distinguons deux appellations différentes.

Définition 2.4 (affectation partielle)**.** *Une affectation d'un ensemble de variables $Y \subseteq X$ est dite **affectation partielle**.*

Définition 2.5 (affectation totale)**.** *Une affectation d'un ensemble de variables Y est dite **totale** (ou **complète**) si $Y = X$.*

Étant données les affectations $\mathcal{A}_1 = (a, b, c)$ et $\mathcal{A}_2 = (a, b', c)$ des variables $\{x_1, x_2, x_3\}$ de l'exemple 2.1, $\mathcal{A}_1$ et $\mathcal{A}_2$ sont des affectations partielles tandis que les affectations $\mathcal{A}_3 = (a, b, c, d, e, f)$ et $\mathcal{A}_4 = (a, b', c, d, e, f)$ des variables $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ sont totales.

Avant de continuer avec les affectations, nous rappelons la définition suivante :

Définition 2.6 (contrainte satisfaite ou violée)**.** *Une contrainte c_i est dite **satisfaite** (respectivement **violée**) par rapport à une affectation $\mathcal{A}$ si $\mathcal{A}[S(c_i)] \in R(c_i)$ (resp. $\mathcal{A}[S(c_i)] \notin R(c_i)$). Dans le premier cas, nous disons aussi que l'affectation $\mathcal{A}$ **satisfait** la contrainte c_i, sinon nous disons qu'elle la **viole**.*

Nous pouvons, maintenant, définir les affectations (in)cohérentes :

Définition 2.7 (affectation cohérente)**.** *Une affectation $\mathcal{A}$ de $Y \subseteq X$ est dite **cohérente** (ou **consistante**) si $\forall c_i \in C \mid S(c_i) \subseteq Y$, $\mathcal{A}[S(c_i)] \in R(c_i)$. Dans le cas contraire, elle est dite **incohérente** (ou **inconsistante**).*

Nous constatons que $\mathcal{A}_1$ et $\mathcal{A}_3$ sont cohérentes et que $\mathcal{A}_2$ et $\mathcal{A}_4$ sont incohérentes car l'affectation des valeurs a et b' aux variables x_1 et x_2 n'est pas autorisée par la contrainte $R(c_1)$.

Définition 2.8 (solution)**.** *Une **solution** est une affectation de toutes les variables de X qui satisfait toutes les contraintes de X.*

Revenons aux affectations $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ et $\mathcal{A}_4$, seule $\mathcal{A}_3$ est une solution car elle est totale et cohérente. Par contre, $\mathcal{A}_1$ est partielle et cohérente, $\mathcal{A}_2$ est partielle et incohérente et $\mathcal{A}_4$ est totale et incohérente.

Étant donné un CSP, la question fondamentale est donc, de déterminer s'il existe une solution. Ce problème est bien connu comme étant NP-complet. Dans le cas positif, le problème est dit satisfiable (ou cohérent). Sinon, il est dit insatisfiable (ou incohérent). Il est possible d'identifier des restrictions sur les contraintes ou les relations pour lesquelles la complexité de la résolution d'un CSP est polynomiale (voir la section 2.6 pour plus de détails). Ceci n'est possible que si la taille du CSP en entrée est polynomiale. Nous concluons cette section avec la définition suivante : nous supposons que cette définition est applicable sur tous les CSP considérés dans ce manuscrit.

Définition 2.9 (CSP normalisé [Bessière, 2006])**.** *Un CSP $P = (X, D, C)$ est dit **normalisé** si $\forall c_i, c_j \in C \mid i \neq j, S(c_i) \neq S(c_j)$.*

Nous pouvons remarquer que les CSP de l'exemple 2.1, 2.2 et 2.3 sont normalisés.

2.3 Représentations graphiques

En théorie des graphes, un graphe est un ensemble de sommets interconnectés par un ensemble d'arêtes telles que chaque arête relie exactement deux sommets tandis qu'un hypergraphe est un ensemble de sommets interconnectés par un ensemble d'hyperarêtes, une hyperarête pouvant relier deux sommets ou plus. Dans les deux cas, un (hyper)graphe G est un couple (V, E) avec V (pour **V**ertex en anglais) l'ensemble de sommets et E (pour **E**dge) l'ensemble des (hyper)arêtes.

Dans ce manuscrit, nous considérons seulement les graphes simples, c'est-à-dire les graphes dont chaque couple de sommets est lié par au plus une seule arête.

Les graphes permettent souvent de présenter l'interaction entre certains objets d'un problème donné. Ici, nous emploierons les graphes soit pour montrer l'interaction entre les variables à travers les contraintes du problème, soit pour observer l'interaction entre les valeurs du domaine de chaque variable à travers les relations. Dans le premier cas, nous parlons de la *structure* du problème et le graphe sera appelé graphe de contraintes dans le cas binaire et hypergraphe de contraintes dans le cas n-aire. La représentation des CSP par des graphes a permis de déduire des nouvelles propriétés sur la résolution des CSP.

2.3.1 Structure

L'(hyper)graphe de contraintes d'un CSP permet d'exprimer l'interaction entre les variables à travers les portées de contraintes. Les sommets d'un *(hyper)graphe de contraintes* sont les variables et les (hyper)arêtes sont les portées de contraintes. Formellement, il est défini comme suit :

Définition 2.10 ((hyper)graphe de contraintes)**.** *Étant donné un CSP $P = (X, D, C)$ d'arité quelconque, l'**(hyper)graphe** de contraintes P, noté $G(P)$ (ou $H(P)$), est un couple (V, E) avec :*
- $V = X$,
- $E = \{S(c_i) \mid c_i \in C\}$.

La figure 2.1 correspond au graphe de contraintes de l'exemple 2.2.

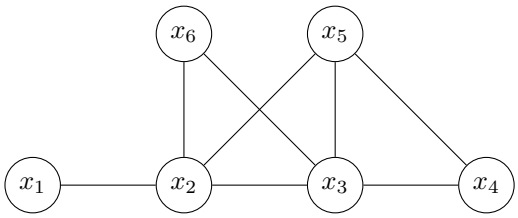

FIGURE 2.1 – Le graphe de contraintes de l'exemple 2.2 et le graphe primal de l'exemple 2.1.

La figure 2.2 présente l'hypergraphe de l'exemple 2.1. Les cercles représentent les hyperarêtes de l'hypergraphe. Dans ce manuscrit, nous utiliserons quelques notions relatives à la théorie des

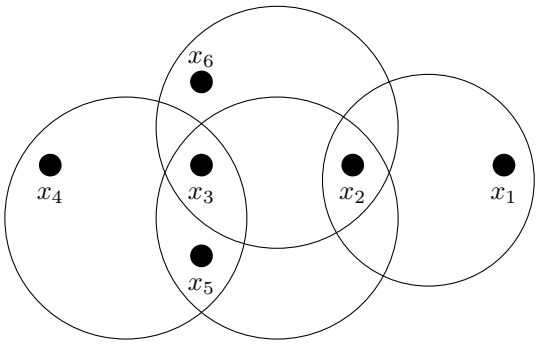

FIGURE 2.2 – Hypergraphe de contraintes de l'exemple 2.1.

graphes, nous en rappelons quelques-unes ici (voir [Berge, 1987a,b] pour plus de détail sur la théorie des graphes et des hypergraphes).

Définition 2.11. *Deux sommets sont voisins s'ils sont interconnectés par une arête.*

Les deux sommets x_1 et x_2 de la figure 2.1 sont voisins.

Définition 2.12. *Le degré d'un sommet correspond au nombre de ses sommets voisins.*

Nous passons maintenant à la notion de *largeur du graphe*, cette notion de a été introduite par Freuder dans [Freuder, 1982]. Étant donné un ordre sur les sommets d'un graphe, la *largeur d'un sommet* est le nombre de sommets adjacents qui le précèdent dans l'ordre. La *largeur de l'ordre* est le maximum de ces largeurs. Enfin, la largeur du graphe est la plus petite des largeurs d'ordre.

Nous finissons ce rappel par la définition des graphes bipartis et plus généralement les graphes n-partis :

Définition 2.13 (graphe biparti)**.** *Un graphe $G = (V, E)$ est dit **biparti** s'il existe V_1 et V_2 tels que $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$ et $\forall x_1, x_2 \in V_1$ (resp. V_2) $\{x_1, x_2\} \notin E$.*

En d'autres termes, un graphe est biparti si nous pouvons partager l'ensemble de ses sommets en deux sous-ensembles disjoints tels qu'il n'y ait pas d'arêtes entre les sommets d'un même sous-ensemble. Chaque sous-ensemble est appelé stable. Nous continuons avec les graphes et nous énonçons la définition du graphe n-parti qui généralise la définition des graphes bipartis.

Définition 2.14 (graphe n-parti)**.** *Un graphe $G = (V, E)$ est dit n-parti si Vpeut être partitionner en n sous-ensembles disjoints tel que les sommets de chaque sous-ensemble ne sont pas interconnectés.*

Dans la littérature, la théorie des graphes s'avère être plus riche que celle des hypergraphes et l'utilisation de ces derniers semble être plus compliquée que les graphes. En effet, il est plus simple de calculer certains paramètres (la largeur, la largeur arborescente, etc) ou de vérifier quelques propriétés (acyclicité, complément, etc) pour un graphe que pour un hypergraphe. Pour cela, la notion de graphe primal [Dechter and Pearl, 1987b] (ou 2-section) a été proposée pour représenter l'interaction entre les variables d'un CSP non-binaire en utilisant les graphes.

Le graphe primal d'un hypergraphe $H(P)$ se construit en transformant toute hyperarête en clique. Formellement, il est défini comme suit :

Définition 2.15 (graphe primal de contraintes)**.** *Étant donné un CSP d'arité quelconque $P = (X, D, C)$ et son hypergraphe de contraintes $H(P) = (V, E)$, le graphe primal de P est le graphe $G^p(P) = (V^p, E^p)$ avec :*

- $V^p = V$,
- $E^p = \{\{x_i, x_j\} \mid x_i, x_j \in X$ *et* $\exists\, c_k \in C \mid \{x_i, x_j\} \subseteq S(c_k)\}$.

Le graphe primal du CSP de l'exemple 2.2 est le graphe de contraintes de l'exemple 2.1 (donné dans la figure 2.1). Après avoir construit ce graphe, nous pouvons définir le CSP binaire associé qui sera appelé le CSP primal.

Définition 2.16 (CSP Primal)**.** *Le CSP **primal** du CSP non-binaire $P = (X, D, C)$ est le CSP binaire $P^p = (X^p, D^p, C^p)$ avec :*

- $X^p = X$,
- $D^p = D$,
- $C^p = \{c^p_{ij} \mid \exists c_k \in C$ *avec* $\{x_i, x_j\} \subseteq S(c_k)$ *tel que* $S(c^p_{ij}) = \{x_i, x_j\}$ *et* $R(c^p_{ij}) = \cap_{c_k \in C \mid \{x_i, x_j\} \subseteq S(c_k)} (R(c_k)[S(c^p_{ij})])\}$.

$R(c_i)$		
x_1	x_2	x_3
a	b	c
a'	b	c'
a'	b'	c

$R(c^p_{12})$	$R(c^p_{23})$	$R(c^p_{13})$
x_1 x_2	x_2 x_3	x_1 x_3
a b	**b c**	a c
a' b	b c'	a' c'
a' b'	b' c	**a' c**

FIGURE 2.3 – Une relation non-binaire $R(c_i)$ et sa transformation en trois relations binaires ($R(c^p_{12})$, $R(c^p_{23})$ et $R(c^p_{13})$).

Malheureusement, la transformation d'un CSP non-binaire en un CSP primal ne conserve pas forcément la satisfiabilité du problème, c'est-à-dire, l'ensemble de solutions de P^p n'est pas toujours égal à celui de P.

En regardant la figure 2.3, nous constatons que les trois relations binaires $R(c^p_{12})$, $R(c^p_{23})$ et $R(c^p_{13})$ ont été obtenues après transformation de la relation n-aire $R(c_i)$ en respectant la définition du CSP primal. Nous pouvons remarquer que (a', b, c) n'est pas une affectation cohérente dans $R(c_i)$ alors que dans les relations binaires il s'agit bien d'une solution.

2.3.2 Microstructure

Nous passons maintenant à une nouvelle représentation graphique de CSP différente de la représentation précédente et qui est appelée *graphe de microstructure* [Jégou, 1993a]. Ce graphe présente l'interaction entre les valeurs en respectant les contraintes et les relations. Intuitivement, les sommets codent les valeurs et les arêtes leurs compatibilités.

Définition 2.17 (microstructure)**.** *Étant donné un CSP binaire $P = (X, D, C)$, la **microstructure** de P est un graphe n-parti $\mu(P) = (V, E)$ avec :*
- $V = \{(x_i, v_i) : x_i \in X, v_i \in D(x_i)\}$,
- $E = \{\ \{(x_i, v_i), (x_j, v_j)\}\ \mid\ i \neq j, c_{ij} \notin C$ *ou* $c_{ij} \in C, (v_i, v_j) \in R(c_{ij})\}$.

Dans le cas où il n'existe pas de contrainte entre deux variables x_i et x_j, cela revient à une contrainte universelle (une contrainte qui autorise toutes les combinaisons de valeurs) entre ces deux variables. Dans ce cas, $S(c_{ij}) = \{x_i, x_j\}$ et $R(c_{ij}) = D(x_i) \times D(x_j)$.

La figure 2.4 présente la microstructure de l'exemple 2.2. Pour des raisons de simplicité, le sommet (x_j, v_j) sera noté v_j dans le graphe de microstructure. Nous remarquons que toutes les valeurs de x_1 sont liées aux valeurs de x_5 car il n'existe pas une contrainte reliant les deux variables. Donc, une contrainte universelle sera considérée. En d'autres termes, la microstructure d'un CSP binaire contient une arête pour toutes les paires de sommets, sauf pour les sommets issus d'un même domaine et pour les sommets correspondant à une combinaison de valeurs qui n'est pas autorisée par une contrainte.

Les définitions suivantes sont importantes avant d'énoncer un résultat sur les microstructures :

Définition 2.18 (graphe complet)**.** *Un graphe $G = (V, E)$ est dit **graphe complet** si $\forall x_i, x_j \in V$ avec $i \neq j$, $\{x_i, x_j\} \in E$.*

Ainsi, un graphe est complet si chaque paire de sommets est reliée par une arête. Le graphe de la figure 2.1 n'est pas complet (x_1 est connecté seulement à x_2). Maintenant, nous pouvons définir la notion de clique (notée souvent Cl) sachant que sa définition découle de la définition du graphe complet.

Définition 2.19 (clique)**.** *Une k-**clique** (ou une clique de taille k) est un ensemble de sommets définissant un sous-graphe complet.*

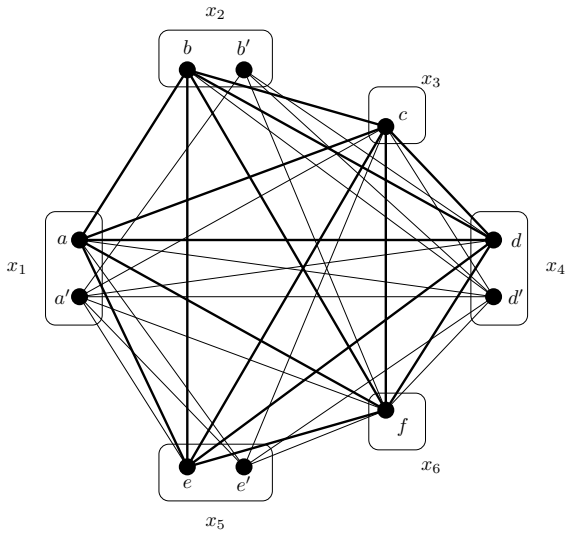

FIGURE 2.4 – Le graphe de microstructure de l'exemple 2.2.

Définition 2.20 (clique maximale)**.** *Une **clique maximale** est une clique qui n'est pas incluse dans une autre clique.*

Le résultat suivant se déduit directement du fait que dans une microstructure, les sommets d'une clique correspondent à des valeurs compatibles issues de domaines différents [Jégou, 1993a].

Proposition 2.1. *Étant donnés un CSP binaire P et sa microstructure $\mu(P)$, une affectation $(v_1, ..., v_n)$ de X est une solution de P ssi $\{(x_1, v_1), ..., (x_n, v_n)\}$ est une n-clique de $\mu(P)$.*

Le graphe de la figure 2.4 a une seule clique maximale de taille 6 : $\{a, b, c, d, e, f\}$. Cette clique correspond évidemment à une solution du problème P_2.

Jégou a montré dans [Jégou, 1993a] que la résolution d'un CSP revient donc à trouver une n-clique dans le graphe de microstructure. On peut facilement constater que la transformation d'un CSP P en sa microstructure $\mu(P)$ peut être réalisée en temps polynomial. On en déduit directement l'existence d'une réduction polynomiale du problème de décision lié à l'existence de solution d'un CSP binaire donnée vers le problème d'existence d'une clique de taille donnée dans un graphe non-orienté (appelé généralement *problème de la clique*).

Cette transformation, qui offre un point de vue nouveau pour l'étude des CSP en fournissant un lien avec des notions issues de la théorie des graphes, a d'abord été exploitée par [Jégou, 1993a] pour définir une nouvelle classe polynomiale pour CSP en se basant sur un résultat important de Gavril [Gavril, 1972]. Une approche identique a été considérée par la suite par Cohen dans [Cohen, 2003] et Salamon et Jeavons dans [Salamon and Jeavons, 2008] (voir section 2.6 pour plus de détails).

Il est clair que la microstructure constitue un outil très utile pour l'étude théorique des CSP. Toutefois, cette notion n'a pu être exploitée que dans la limite des CSP binaires. Le complément de la microstructure pour le cas non-binaire a été introduite par Cohen dans [Cohen, 2003] sans pour autant aboutir à des résultats intéressants comme pour le cas binaire.

Nous rappelons que le complément d'un graphe peut être défini comme suit :

Définition 2.21 (Le complément d'un graphe)**.** *Le* complément d'un graphe $G = (V, E)$ *est un graphe* $\overline{G} = (V, \overline{E})$ *tel que* $\overline{E} = \{\{x_i, x_j\} \mid \{x_i, x_j\} \notin E\}$.

En d'autres termes, le complément d'un graphe est un graphe dans lequel chaque paire de sommets initialement déconnectés seront connectés et inversement. La figure 2.5 représente le complément du graphe de la figure 2.1.

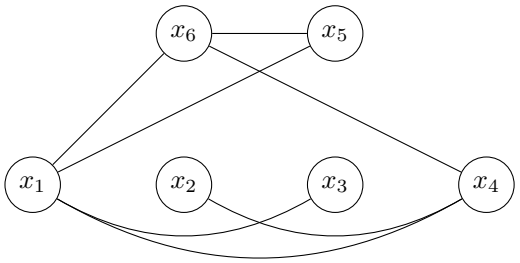

FIGURE 2.5 – Le complément de graphe de contraintes de l'exemple 2.2 présenté à la figure 2.1.

La généralisation de Cohen [Cohen, 2003] qui s'appuie sur le complément d'un hypergraphe est définie de la façon suivante :

Définition 2.22 (Complément de la microstructure)**.** *Étant donné un CSP* $P = (X, D, C)$*, le* ***complément de la microstructure*** *de* P *est un hypergraphe* $\overline{\mathcal{M}}(P) = (V, \overline{E})$ *avec :*

- $V = \{(x_i, v_i) : x_i \in X, v_i \in D(x_i)\}$,
- $\overline{E} = E_1 \cup E_2$ *tels que*
  - $E_1 = \{\ \{(x_i, v_j), (x_i, v_{j'})\}\ \mid\ x_i \in X$ *et* $j \neq j'\}$
  - $E_2 = \{\{(x_{i_1}, v_{i_1}), \ldots (x_{i_k}, v_{i_k})\}\ \mid\ c_i \in C, S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\}$ *et* $(v_{i_1}, \ldots v_{i_k}) \notin R(c_i)\}$.

Nous pouvons facilement constater que pour le cas binaire, la définition de Cohen correspond exactement au complément de la microstructure présentée dans [Jégou, 1993a]. Pour le cas non-binaire, malheureusement, la notion de complément d'un hypergraphe semble ne pas avoir été étudiée dans la littérature, du moins à notre connaissance. En fait, cette notion pose plusieurs questions dont la principale consiste à savoir s'il faudrait considérer toutes les hyperarêtes qui correspondent aux relations universelles, à l'image de la notion de complémentaire de graphe dans le cas binaire. Mais dans ce cas, la taille de l'hypergraphe serait potentiellement exponentielle en fonction de la taille du CSP. Pour cette raison, vraisemblablement, cette définition n'a pas été exploitée même dans [Cohen, 2003] que dans le cas binaire.

Dans la section suivante, nous parlerons des méthode, dites filtrages par cohérence, permettant la réduction des domaines par suppression des valeurs ne pouvant appartenir à une solution.

2.4 Cohérence locale et Filtrage

Étant donné un CSP P, vérifier si P admet une solution est NP-complet. Devant cette complexité, des pistes de recherche ont été développées afin de réduire l'espace à parcourir grâce à la suppression des valeurs ou des tuples ne pouvant participer à aucune solution. Ces méthodes sont appelées *filtrage par cohérence* ou *propagation de contraintes*. En effet, les filtrages par cohérence représentent l'ensemble de techniques permettant de supprimer les valeurs incohérentes et garantir un niveau de cohérence d'un CSP donné. Dans la littérature, il existe plusieurs formes et niveaux de filtrage qui peuvent être appelés avant ou pendant la recherche d'une solution. Tous les

algorithmes de filtrages peuvent être munis d'un ordre sur les variables (ou sur les contraintes). Dans ce cas, leur nom sera suivi par le mot « directionnelle ».

Dans ce qui suit, nous rappellerons le principe de quelques-uns, les inconvénients, les algorithmes qui ont été proposés et leurs complexités.

2.4.1 La cohérence d'arc

La cohérence d'arc (AC pour **A**rc **C**onsistency) présente un niveau de filtrage le plus facile à tester et à intégrer dans les algorithmes de résolution. Elle a été introduite par Mackworth dans [Mackworth, 1977] et elle consiste à supprimer toute valeur n'ayant pas au moins une valeur compatible dans le domaine de chaque variable voisine. La suppression d'une valeur peut engendrer la suppression des tuples dans lesquels cette valeur apparaît.

Définition 2.23 (cohérence d'arc)**.** *Étant donné un binaire $P = (X, D, C)$, une valeur $v_i \in D(x_i)$ est dite arc-cohérente si $\forall x_j \in X$ telle que $c_{ij} \in C, \exists v_j \in D(x_j)$ telle que $(v_i, v_j) \in R(c_{ij})$. Un domaine $D(x_i)$ est dit arc-cohérent si $\forall\ v_i \in D(x_i)$, v_i est arc-cohérente. Un CSP P est dit arc-cohérent si $\forall\ D(x_i) \in D, D(x_i)$ est non-vide et est arc-cohérent.*

Pour $(v_i, v_j) \in R(c_{ij})$, nous disons que v_j est un support de v_i dans $D(x_j)$ et inversement.

L'exemple 2.2 n'est pas arc-cohérent car les domaines $D(x_1)$, $D(x_2)$, $D(x_4)$ et $D(x_5)$ ne sont pas arc-cohérents. Après application de l'arc cohérence, on aura le graphe de microstructure donné par la figure 2.6. La valeur b' et a été supprimée car elle n'a pas de support vis à vis de x_3. Après sa suppression, la valeur a' n'aura plus de support dans x_2 et elle sera également supprimée. e' a été aussi supprimée car elle n'a pas de support vis à vis de x_2. Après sa suppression, la valeur d' n'aura plus de support dans x_2 et elle sera aussi supprimée.

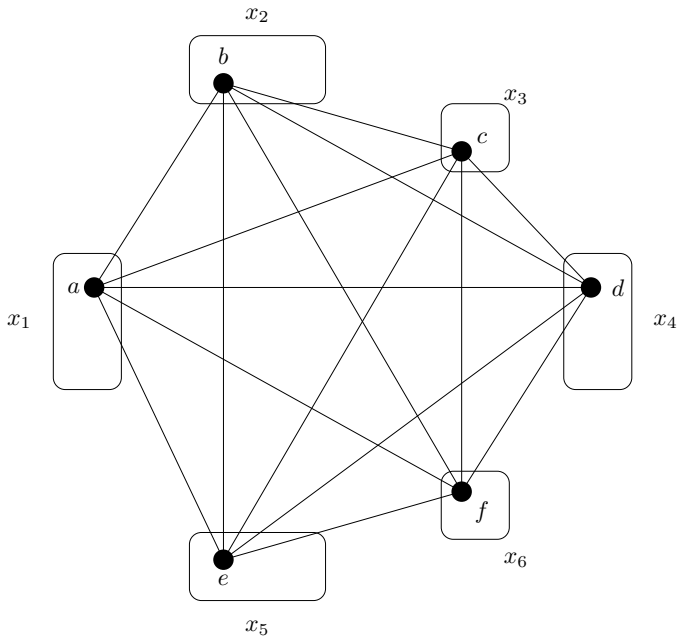

FIGURE 2.6 – Le graphe de microstructure de l'exemple 2.2 après AC.

En pratique, il existe plusieurs algorithmes permettant d'appliquer AC, c'est-à-dire supprimer toute valeur $v_i \in D(x_i)$ qui ne satisfait pas la condition donnée par la définition 2.23. Nous en citons quelques-uns ainsi que leur complexité dans le tableau 2.1 . Pour AC3, il existe plusieurs ex-

Algorithme	Complexité temporelle	Complexité spatiale	Référence
AC1	O(ned^5)	O($e + nd$)	[Mackworth and Freuder, 1985]
AC3	O(ed^3)	O($e + nd$)	[Mackworth, 1977]
AC2001	O(ed^2)	O(ed)	[Bessière et al., 2005]
AC4	O(ed^2)	O(ed^2)	[Mohr and Henderson, 1986]
AC5	O(ed^3)	O($e + nd$)	[Hentenryck et al., 1992]
AC6	O(ed^2)	O(ed)	[Bessière, 1994]
AC7	O(ed^2)	O(ed)	[Bessière et al., 1999]
AC8	O(ed^3)	O(n)	[Chmeiss and Jégou, 1998]

TABLE 2.1 – Quelques algorithmes de cohérence d'arc pour les CSP binaires et leurs complexités.

tensions comme AC3.1 (appelé aussi AC2001) proposé par Bessière, Régin, Yap et Zhang [Bessière et al., 2005], AC3.2 et AC3.3 proposés par Lecoutre et al. dans [Lecoutre et al., 2003].

Théoriquement, AC2001, AC6 et AC7 disposent de la meilleure complexité temporelle (en O(ed^2)) et d'une complexité spatiale en O(ed) mais AC3^{rm} [Lecoutre and Hemery, 2006] et AC3^{rm+bit} [Lecoutre and Vion, 2008] sont les plus efficaces en pratique.

En pratique, il existe quelques implémentations de AC spécifiques à certains types de contraintes comme l'algorithme proposé par Régin [Régin, 1994] pour les contraintes AllDif.

Une définition de la cohérence d'arc a été proposée pour le cas des CSP non-binaires : elle s'appelle la cohérence d'arc généralisée [Mohr and Masini, 1988] (ou GAC pour **G**eneralized **A**rc **C**onsistency en anglais). Avant de l'énoncer, nous rappelons la définition d'un tuple valide :

Définition 2.24 (tuple valide)**.** *Un tuple $t_i \in R(c_i)$ est valide si et seulement si aucune valeur de ce tuple ne sera supprimée du domaine de la variable correspondante [Paparrizou and Stergiou, 2012].*

Définition 2.25 (cohérence d'arc généralisée)**.** *Étant donné un CSP $P = (X, D, C)$ d'arité quelconque, une valeur $v_i \in D(x_i)$ est dite arc-cohérente généralisée si et seulement si pour chaque contrainte c_i telle que $x_i \in S(c_i)$, il existe un tuple valide tuple $t \in R(c_i)$ qui contient une affectation de v_i à x_i. Un domaine $D(x_i)$ est arc-cohérent généralisé si et seulement si toutes ses valeurs sont arc-cohérentes généralisées. Un CSP P est dit arc-cohérent généralisé si et seulement si $\forall\ D(x_i) \in D, D(x_i)$ est non-vide et est arc-cohérent généralisé.*

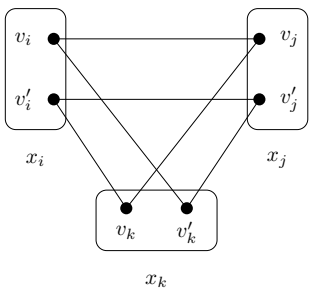

FIGURE 2.7 – Un exemple arc-cohérent mais pas cohérent.

Dans certains cas, la cohérence d'arc ne suffit pas pour supprimer toutes les valeurs incohérentes. Par exemple, dans la figure 2.7, les six valeurs sont arc-cohérentes alors que le CSP est globalement incohérent. Pour cette raison, des niveaux de cohérences plus forts que AC ont été proposés. Ceci sera l'objet des sous-sections suivantes.

2.4.2 La cohérence de chemin

La cohérence de chemin (**P**ath **C**onsistency) a été proposée par Montanari dans [Montanari, 1974]. Elle consiste à supprimer toute affectation cohérente de deux variables différentes ne pouvant être étendue à une troisième. Autrement dit, la cohérence de chemin ne supprime que des paires de valeurs (tuples).

Définition 2.26 (cohérence de chemin)**.** *Un CSP binaire $P = (X, D, C)$ est dit **chemin-cohérent** si pour tout triplet de variables (x_i, x_j, x_k), $\forall v_i \in D(x_i)$ et $v_j \in D(x_j)$ telles que, $(v_i, v_j) \in R(c_{ij})$, $\exists v_k \in D(x_k) \mid (v_i, v_k) \in R(c_{ik})$ et $(v_j, v_k) \in R(c_{jk})$.*

L'exemple 2.2 contient quelques couples de valeurs chemin-incohérents. Par exemple, le couple (e', f) n'a pas de support vis-à-vis de $D(x_2)$, donc l'arête correspondante sera supprimée. Après application de la cohérence de chemin, on aura le graphe donné par la figure 2.8.

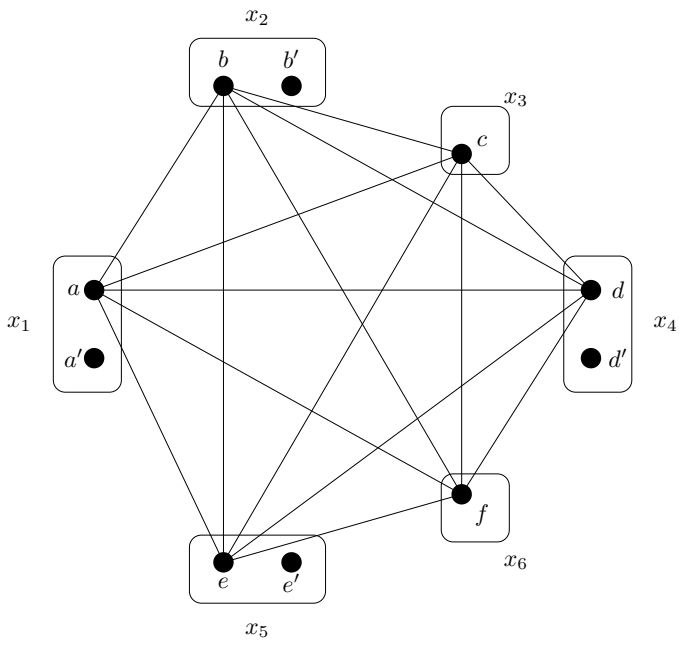

FIGURE 2.8 – Le graphe de microstructure de l'exemple 2.2 après PC.

Comme pour AC, il existe plusieurs algorithmes réalisant le filtrage par PC parmi lesquels nous pouvons citer les suivants :

Les algorithmes PC5 et PC6 sont les mêmes. En fait, ils présentent une généralisation de AC6. Nous utiliserons PC$\{5|6\}$ pour les désigner. PC8 a la meilleure complexité spatiale en O(n^2d) (contre O(n^3d^2) pour PC$\{5|6\}$), mais PC$\{5|6\}$ ont la meilleure complexité en temps O(n^3d^3) (la complexité en temps de PC8 est O(n^3d^4)).

Algorithme	complexité temporelle	complexité spatiale	référence
PC1	O(n^5d^5)	O(n^3d^2)	[Montanari, 1974]
PC2	O(n^3d^5)	O(n^3)	[Mackworth, 1977]
PC2001	O(n^3d^3)	O(n^3d^2)	[Bessière et al., 2005]
PC4	O(n^3d^3)	O(n^3d^3)	[Han and Lee, 1988]
PC$\{5\|6\}$	O(n^3d^3)	O(n^3d^2)	[Singh, 1995, Chmeiss, 1996]
PC8	O(n^3d^4)	O(n^2d)	[Chmeiss and Jégou, 1998]

TABLE 2.2 – Quelques algorithmes de cohérence de chemin et leurs complexités.

Une deuxième forme de cohérence, appelée la cohérence duale (dual consistency) et dont l'idée a été initialement utilisée dans [McGregor, 1979], a été introduite par Lecoutre et al. dans [Lecoutre et al., 2007a,b] pour les CSP binaires et ensuite a été étendue aux CSP d'arité quelconque dans [Lecoutre et al., 2011]. Dans le cas binaire, la cohérence duale est équivalente à la cohérence de chemin. En pratique, il existe plusieurs algorithmes pour appliquer la cohérence d'arc et la cohérence duale comme sCD1, sCD2 et sCD3. Les complexités en temps de ces algorithmes sont respectivement O(n^5d^5), O(n^5d^5) et O(n^3d^4). La complexité en espace est en O(n^2d^2).

Contrairement à la cohérence d'arc, la cohérence de chemin travaille sur les relations, ce qui permet d'avoir des valeurs sans aucun support après application de PC (voir figure 2.8). Ceci est généralement résolu par l'application de AC après PC.

En plus, la cohérence de chemin peut modifier le problème de départ par l'ajout des nouvelles contraintes ce qui donc modifie le graphe de contraintes (voir figure 2.9). Au début, la contrainte qui porte sur x_i et x_k est universelle. Après application de PC, les deux tuples (v_i, v_k) et (v'_i, v'_k) seront supprimés. Par conséquent, une contrainte c_{ik} sera ajoutée.

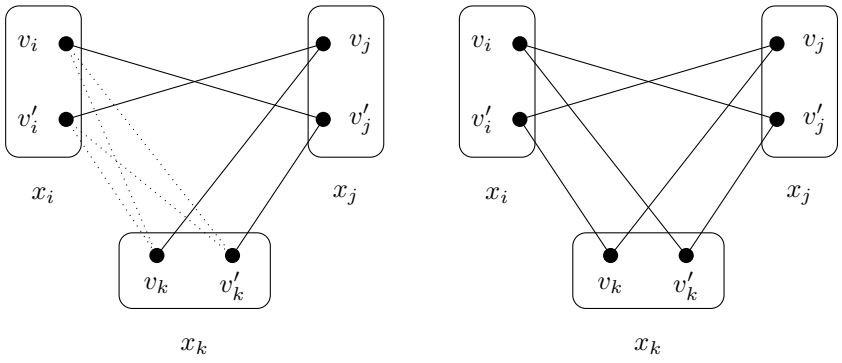

FIGURE 2.9 – Un CSP avec deux contraintes au départ (la relation universelle est représentée en pointillé). Après PC, une nouvelle contrainte sera ajoutée.

Pour éviter la modification du problème par la cohérence de chemin, la *cohérence de chemin restreinte* (notée RPC pour *Restricted Path Consistency*) a été proposée par Berlandier [Berlandier, 1995]. En effet, elle examine la cohérence d'arc de chaque valeur et de plus elle teste pour chaque couple (v_i, v_j), avec $v_i \in D(x_i)$ et $v_j \in D(x_j)$ et v_j le seul support de v_i dans $D(x_j)$, s'il est chemin cohérent.

Définition 2.27 (cohérence de chemin restreinte)**.** *Un CSP binaire $P = (X, D, C)$ satisfait la cohérence de chemin restreinte si et seulement si :*

$\forall x_i \in X$, $D(x_i)$ est un domaine arc-cohérent non-vide, $\forall v_i \in D(x_i)$, $\forall x_j \in X$ telle que v_i a un support unique v_j dans $D(x_j)$, $\forall x_k \in X$ telle que x_k est une variable voisine de x_i et x_j dans le graphe de contraintes, $\exists v_k \in D(x_k)$ telle que $(v_i, v_k) \in R(c_{ik})$ et $(v_j, v_k) \in R(c_{jk})$.

Cette définition a été étendue par la suite par Debruyne et Bessière [Debruyne and Bessière, 1997] pour les valeur v_i ayant k supports dans v_j. Cette nouvelle cohérence a été appelée la *k-cohérence de chemin restreinte* et elle est notée k-RPC.

Définition 2.28 (k-cohérence de chemin restreinte)**.** *Un CSP binaire $P = (X, D, C)$ est dite k-RPC si et seulement si :*
$\forall x_i \in X$, $D(x_i)$ est un domaine arc-cohérent non-vide, $\forall v_i \in D(x_i)$, $\forall x_j \in X$ telle que v_i a k supports v_j dans $D(x_j)$, $\forall x_l \in X$ telle que x_l est une variable voisine de x_i et x_j dans le graphe de contraintes, $\exists v_l \in D(x_l)$ telle que $(v_i, v_l) \in R(c_{il})$ et $(v_j, v_l) \in R(c_{jl})$.

Dans [Debruyne and Bessière, 1997], Debruyne et Bessière ont proposé aussi la cohérence de chemin restreinte maximale (notée maxRPC). En fait, maxRPC peut être considérée comme la borne de k-RPC. Elle consiste à supprimer toutes les valeurs $v_i \in D(x_i)$ ayant au moins un support dans $D(x_j)$ et formant ensemble un couple chemin-cohérent.

Définition 2.29 (cohérence de chemin restreinte maximale)**.** *Un CSP binaire $P = (X, D, C)$ est dite maxRPC si et seulement si :*
$\forall x_i \in X$, $D(x_i)$ est un domaine arc-cohérent non-vide, $\forall v_i \in D(x_i)$, $\forall c_{ij} \in C$, $\exists v_j \in D(x_j)$ telle que $(v_i, v_j) \in R(c_{ij})$ et $\forall x_k \in X$ telle que x_k est une variable voisine de x_i et x_j dans le graphe de contraintes, $\exists v_k \in D(x_k)$ telle que $(v_i, v_k) \in R(c_{ik})$ et $(v_j, v_k) \in R(c_{jk})$.

En pratique, maxRPC1 [Debruyne and Bessière, 1997] et maxRPC2 [Debruyne and Bessière, 1997] ont respectivement une complexité temporelle O($ed^2 + cd^3$) et O($ed^2 + cd^2$) et la même complexité spatiale O($ed + cd$) (avec c le nombre de cliques de taille 3 dans la microstructure).

2.4.3 La k-cohérence et la k-cohérence forte

Si la cohérence d'arc (resp. la cohérence de chemin) teste la cohérence d'une valeur par rapport à deux variables (resp. trois variables), la k-cohérence (ou la k-consistency) [Freuder, 1985] teste la cohérence d'un ensemble de $k-1$ valeurs par rapport à la k-ième variable. C'est-à-dire, elle teste s'il existe une valeur $v_k \in D(x_k)$ compatible avec $(k-1)$ valeur deux-à-deux cohérentes.

Définition 2.30 (k-cohérence)**.** *Étant donnés un CSP $P = (X, D, C)$ et un entier k tel que $1 \leq k < n$. P est dit k-**cohérent** si pour tout ensemble Y de $k-1$ variables et pour toute variable additionnelle x de P, toute affectation partielle cohérente de Y peut être étendue à une affectation cohérente de $Y \cup \{x\}$.*

Nous remarquons que la cohérence d'arc correspond à la 2-cohérence et la cohérence de chemin correspond à la 3-cohérence.

Maintenant, nous pouvons définir la *k-cohérence forte*. Cette notion a aussi été introduite par Freuder dans [Freuder, 1985]. La définition est comme suit :

Définition 2.31 (k-cohérence forte)**.** *Un CSP $P = (X, D, C)$ est dit fortement k-cohérent si pour tout i tel que $1 \leq i \leq k$, P est i-cohérent.*

Nous pouvons par exemple dire qu'un CSP binaire est *fortement chemin-cohérent* (noté SPC pour Strong Path Consistency) s'il est arc et chemin cohérent.

Un algorithme pour appliquer la *k-cohérence* a été proposé par Cooper dans [Cooper, 1989]. Par ailleurs, la complexité en temps pour l'application de la k-cohérence (O($n^k d^k$)) constitue un obstacle pour une exploitation en pratique de ce niveau de cohérence pour $k > 3$. En plus, la k-cohérence modifie le graphe de contraintes en rajoutant des contraintes d'arité inférieure ou égale à $k-1$ pour le cas binaire. Par conséquent, d'autres formes de cohérences (dites cohérences inverses) ont été introduites pour éviter ces inconvénients.

2.4.4 Les cohérences inverses

Dans cette section, nous présentons deux formes de cohérence inverse. La première est la *cohérence de chemin inverse* (appelée PIC pour Path Inverse Consistency) qui a été proposée par Freuder et Elfe dans [Freuder and Elfe, 1996]. Elle a été aussi introduite pour éviter les inconvénients de la cohérence de chemin. Elle présente un niveau de filtrage plus fort que AC et que PC. La définition est la suivante :

Définition 2.32 (cohérence de chemin inverse)**.** *Un CSP binaire $P = (X, D, C)$ est dit **chemin-cohérent inverse** si pour tout triplet de variables (x_i, x_j, x_k), $\forall v_i \in D(x_i)$, $\exists v_j \in D(x_j)$ et $v_k \in D(x_k)$ telles que $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$ et $(v_j, v_k) \in R(c_{jk})$.*

Dans [Freuder and Elfe, 1996], les auteurs ont proposé un algorithme PIC1 pour appliquer la cohérence de chemin inverse de complexité temporelle O(en^2d^4) et de complexité spatiale O(n). Par la suite, Debruyne a proposé PIC2 dans [Debruyne, 2000] qui a une complexité temporelle en O($ed^2 + cd^3$) et une complexité spatiale en O(cd).

Nous passons maintenant à la *cohérence de voisinage inverse* (appelée NIC pour *Neighborhood Inverse Consistency*) qui a été proposée aussi par Freuder et Elfe dans [Freuder and Elfe, 1996] pour les CSP binaires. Elle présente un niveau de filtrage plus fort que AC et PIC. La définition est la suivante :

Définition 2.33 (cohérence de voisinage inverse)**.** *Un CSP binaire $P = (X, D, C)$ est dit **voisin-cohérent inverse** si $\forall x_i \in X$ et $\forall v_i \in D(x_i)$, il existe une affectation cohérente de toutes les variables voisines de x_i compatible avec la valeur v_i.*

Freuder et Elfe ont proposé un algorithme pour appliquer NIC de complexité temporelle O($\Delta^2(n + ed)d^{\Delta+1}$) (avec Δ le degré maximum du graphe, c'est-à-dire, le degré de la variable ayant le plus grand nombre de voisins dans le graphe de contraintes). Dans [Debruyne and Bessière, 2001], Debruyne et Bessière ont montré que NIC n'est pas expérimentalement efficace et est très coûteux quand le graphe de contrainte est dense, ce qui n'est pas surprenant.

2.4.5 La singleton cohérence d'arc

La *singleton cohérence d'arc* (**S**ingleton **A**rc **C**onsistency [Debruyne and Bessière, 1997]) présente un niveau de filtrage plus fort que la cohérence d'arc. Elle teste la cohérence d'arc après avoir affectée chacune des valeurs v_i à une variable x_i.

Définition 2.34 (singleton cohérence d'arc)**.** *Un CSP binaire $P = (X, D, C)$ est dite **singleton arc-cohérent** si $\forall x_i \in X$, $\forall v_i \in D(x_i)$, le sous-problème obtenu après affectation de v_i à x_i , est pas arc-cohérent.*

En pratique, il existe plusieurs algorithmes pour appliquer SAC :
- SAC1 proposé par Debruyne et Bessière [Debruyne and Bessière, 1997],
- SAC2 proposé par Barták et Erben [Barták and Erben, 2004],
- SAC-Opt proposé par Debruyne et Bessière [Bessière and Debruyne, 2004],

– SAC-SDS proposé par Debruyne et Bessière [Bessière and Debruyne, 2005],
– SAC3 proposé par Bessière, Cardon, Debruyne et Lecoutre [Bessiere et al., 2011],
– SAC3-SDS proposé par Bessière, Cardon, Debruyne et Lecoutre [Bessiere et al., 2011].

La meilleure complexité en temps pour SAC est celle de SAC-Opt et SAC3 (O(ned^3)) [Bessière and Debruyne, 2005] alors qu'en pratique SAC-SDS, SAC3 et SAC3-SDS sont les plus efficaces. SAC a été par la suite généralisée aux CSP non-binaires et elle est notée SGAC (pour **S**ingleton **G**eneralized **A**rc **C**onsistency). Sa complexité en temps est O(ne^2d^{2a}).

2.4.6 Relation entre les cohérences des CSP binaires

Le schéma de la figure 2.10 définit le lien entre les différentes formes de cohérences présentées précédemment pour le cas des CSP binaires [Debruyne and Bessière, 2001] : une flèche de CO_1 vers CO_2 signifie que CO_1 est plus fort que CO_2. Les pointillés désignent l'incomparabilité de deux cohérences.

Nous disons que la cohérence CO_1 est *plus forte que* la cohérence CO_2 (notée $CO_2 \leq CO_1$) si CO_1 supprime au moins les mêmes valeurs que CO_2. Nous disons que la cohérence CO_1 est *strictement plus forte que* la cohérence CO_2 (notée $CO_2 < CO_1$) si $CO_2 \leq CO_1$ et s'il existe au moins un CSP P où CO_1 supprime plus de valeurs que CO_2. Nous pouvons facilement remarquer que ces relations sont transitives. Enfin, nous disons que CO_1 et CO_2 sont *incomparables* s'il existe au moins un CSP pour lequel $CO_2 \leq CO_1$ et une autre pour lequel $CO_1 \leq CO_2$.

Nous pouvons constater que SPC, NIC et SAC présentent un niveau de filtrage très fort même si SPC a été démontré plus fort que SAC (et évidemment AC) mais elle est incomparable avec NIC. maxRPC est évidemment plus fort que k-RPC, RPC, PIC et AC. AC constitue donc le niveau de filtrage le plus simple et basique et possède la meilleure complexité en temps et en espace.

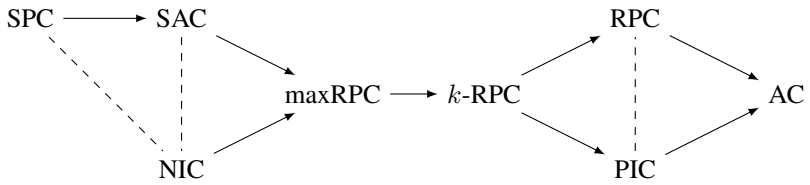

FIGURE 2.10 – Relation entre les cohérences.

2.4.7 L'hyper-k-cohérence

Dans cette section, nous présenterons quelques formes de cohérences qui travaillent sur les contraintes. Nous commençons par l'hyper-2-cohérence (hyper-2-consistency) qui est appelée aussi inter-cohérence (PairWise Consistency). Le principe de l'inter-cohérence a été défini pour les bases de données relationnelles et par la suite a été introduit pour les CSP dans [Janssen et al., 1989]. Elle consiste à vérifier que chaque tuple de la relation associée à la contrainte c_i a un support dans la relation associée à la contrainte c_j. En d'autres mots, il s'agit d'appliquer le principe de la cohérence d'arc sur les relations. En effet, nous testons pour chaque tuple t_i de la relation $R(c_i)$ s'il existe un tuple t_j de chaque relation $R(c_j)$ tels que t_i et t_j sont compatibles.

Définition 2.35 (Inter-cohérence [Janssen et al., 1989])**.** *Un CSP $P = (X, D, C)$ est* ***inter-cohérente ou pairwise-cohérent*** *ssi $\forall\ 1 \leq i \leq e$, $R(c_i) \neq \emptyset$ et $\forall\ 1 \leq i, j \leq e$, $R(c_i)[S(c_i) \cap S(c_j)] = R(c_j)[S(c_i) \cap S(c_j)]$.*

Ce principe peut être appliqué sur k contraintes pour obtenir l'hyper-k-cohérence (hyper-k-consistency ou aussi k-wise consistency). L'hyper-k-cohérence a été introduite par Jégou dans [Jégou, 1993b, 1991]. C'est une généralisation de la k-cohérence aux CSP n-aires. La définition est la suivante :

Définition 2.36 (hyper-k-cohérence)**.** *Un CSP $P = (X, D, C)$ est **hyper-k-cohérent** ssi* $\forall c_i$, $R(c_i) \neq \emptyset$ *et* $\bowtie_{i=1}^{k-1} R(c_i)[(\cup_{i=1}^{k-1} S(c_i)) \cap S(c_k)] \subseteq R(c_k)[(\cup_{i=1}^{k-1} S(c_i)) \cap S(c_k)]$.

La complexité de l'application de l'hyper-k-cohérence (O(r^k)) présente un inconvénient majeur pour qu'elle soit exploitée en pratique. Comme la k-cohérence, l'hyper-k-cohérence peut aussi modifier l'hypergraphe de contraintes de départ. L'hyper-k-cohérence forte ou la k-wise consistance forte a été introduite dans [Jégou, 1993a]. La définition est comme suit :

Définition 2.37 (hyper k-cohérence forte)**.** *Un CSP $P = (X, D, C)$ est dit fortement hyper-$(k+1)$-cohérente si pour tout $1 \leq i \leq k$, P est hyper-i-cohérent.*

2.5 Méthodes de résolutions

Le problème CSP est connu pour être NP-complet. Comme nous n'avons pas de méthode universelle pour une résolution efficace de tout CSP, alors des méthodes différentes ont été proposées parmi lesquelles nous citons :

- **Les méthodes complètes** [Darwiche and Pipatsrisawat, 2009] : elles explorent l'espace de recherche de manière exhaustive afin de pouvoir décider de la satisfiabilité d'un CSP.
- **Les méthodes incomplètes** [Kautz et al., 2009] : elles explorent seulement une partie de l'espace de recherche afin d'atteindre une solution.

Dans ce manuscrit, nous nous intéressons seulement aux méthodes complètes. Une idée triviale consiste à générer toutes les combinaisons de valeurs possibles et vérifier si toutes les contraintes sont satisfaites. Une telle approche est appelée *générer et tester* (pour *generate and test*). Il est clair que cette méthode est fortement combinatoire et que sa complexité est de l'ordre de O($e.d^n$), donc inexpoitable en pratique. Il est donc nécessaire d'améliorer cette méthode afin de rendre meilleure cette complexité.

Dans les prochaines sous-sections, nous illustrerons certains algorithmes de résolution de CSP basés sur le principe « générer et tester » et évitant ses défauts.

2.5.1 Backtrack

La méthode standard de résolution de CSP est le *Backtrack* (noté BT). L'algorithme backtrack [Golomb and Baumert, 1965], dont l'idée est inspirée de « generate and test » explore l'espace de recherche d'une manière exhaustive même s'il teste la satisfaction des contraintes après chaque instanciation. Les différentes étapes de la résolution sont décrites ci-dessous.

- Il commence par une affectation vide et dans le cas général, étant donnée une affectation partielle cohérente courante $(v_1, v_2, \ldots, v_i)$, il choisit une nouvelle variable x_{i+1} (ligne 6 de l'algorithme 1) et cherche à affecter des valeurs de $D(x_{i+1})$ à x_{i+1} (ligne 7 de l'algorithme 1).
- Le seul contrôle effectué consiste alors à vérifier que l'affectation résultante $(v_1, v_2, \ldots, v_i, v_{i+1})$ est cohérente (ligne 1 de l'algorithme 1).
  - Dans l'affirmative, il continue avec la prochaine variable si $i < n$ (ligne 4 de l'algorithme 1)
  - si $i = n$; il s'agit d'une solution (ligne 3 de l'algorithme 1).

- Sinon (si $(v_1, v_2, \ldots, v_i, v_{i+1})$ n'est pas compatible), BT affecte une autre valeur de $D(x_{i+1})$ à x_{i+1} et il la teste. S'il n'y a plus de valeur inexplorée, il s'agit d'un échec et BT effectue un *backtrack* (retour arrière chronologique vers la variable précédente pour la réaffecter).

Il est facile de voir que la recherche effectuée par BT correspond à un parcours en profondeur d'abord d'un arbre sémantique appelé *arbre de recherche* dont la racine est un tuple vide, tandis que les branches sont des i-uplets qui représentent les affectations des variables le long du chemin correspondant dans l'arbre. Les nœuds de cet arbre qui correspondent à des affectations partielles cohérentes sont appelés *nœuds cohérents*, tandis que les autres nœuds sont appelés *nœuds incohérents*.

La complexité en temps de BT est O(ed^n) sachant que le nombre de nœuds dans l'arbre de recherche est au plus $\Sigma_{0 \leq i \leq n} d^i = \frac{d^{n+1}-1}{d-1}$, par conséquent, il est en O (d^n). Nous supposons aussi que le test de contraintes peut se réaliser en temps constant.

Algorithme 1 : BT($P = (X, D, C)$: CSP ; $t = (v_1, v_2, \ldots v_i)$: tuple) : Booléen

```
1  si t = (v_1, v_2, ... v_i) est une affectation cohérente partielle alors
2      si (i = n) alors
3          retourner vrai /* t = (v_1, v_2, ... v_i) est une solution */
4      sinon
5          Coherence ← faux
6          D'(x_{i+1}) ← D(x_{i+1})
7          Choisir une variable x_{i+1} à instancier
8          tant que D'(x_{i+1}) ≠ ∅ et Coherence = faux faire
9              Choisir v_{i+1} ∈ D(x_{i+1})
10             D'(x_{i+1}) ← D'(x_{i+1}) \ {v_{i+1}}
11             t' ← (v_1, v_2, ... v_i, v_{i+1})
12             Coherence ← BT(P' = (X, D', C),t')
13         retourner Coherence
```

BT peut être considéré comme un algorithme générique mais il souffre aussi de certains défauts. Par exemple, il effectue des retours arrières chronologiques alors que l'échec ne provient pas forcément de ce niveau. Des algorithmes basés sur BT et utilisés en pratique réalisent un certain travail supplémentaire à chaque nœud de l'arbre de recherche. Il existe principalement trois moyens pour l'amélioration de backtrack dont la tâche correspond à la :

- modification de l'algorithme backtrack de façon à ce qu'il puisse effectuer des retours arrières non chronologiques ou l'enregistrement d'*informations* pour éviter de ré-explorer des parties déjà visitées,
- réduction de l'espace de recherche avant et pendant la recherche d'une solution par filtrage par cohérence,
- proposition des heuristiques pour l'ordre d'instanciations des variables, des heuristiques aussi pour le choix de la contrainte et d'autres pour les valeurs.
- utilisation de *redémarrage* : Une stratégie de redémarrage consiste à redémarrer entièrement la résolution d'un CSP après un temps fixé en mémorisant certaines informations sur l'exécution actuelle. Après chaque redémarrage, l'algorithme est relancé en exploitant les informations des exécutions précédentes pour ne pas redévelopper un sous-arbre déjà exploré.

Nous notons par nBT la version n-aire de BT et qui correspond exactement au même algorithme que BT.

2.5.2 Backtrack avec retour arrière non chronologique

En cas d'échec sur une variable x_i, BT effectue un retour arrière chronologique sur la variable x_{i-1}. Cependant, il est possible que cette variable ne soit pas la cause de l'échec surtout quand on n'a pas de contrainte qui porte sur x_i et x_{i-1}. Dans ce cas, le retour arrière chronologique ne résoudra pas le problème vu que x_i n'est pas la source de l'échec. Pour éviter ce problème, certains algorithmes, dits intelligents, qui effectue des retours en arrière non chronologiques ont été proposés. Parmi ces algorithmes, nous pouvons citer le *Backjumping* (noté BJ et appelé aussi *le backtrack intelligent*)[Stallman and Sussman, 1977], le *retour arrière dirigé par les conflits* (noté CBJ) [Prosser, 1993]. Ces algorithmes diffèrent dans leur façon d'interpréter l'échec et donc sur le choix de la variable vers laquelle ils effectueront un retour arrière.

Par exemple, BJ fait un retour arrière vers la variable x_j la plus récemment affectée est qui est en conflit avec la variable courante x_i si toutes les valeurs de x_i sont incohérentes avec l'affectation courante de x_j. Mais si toutes les autres valeurs de x_j ont été testées, alors BJ va revenir à la variable qui la précède dans l'ordre d'affectation. Donc, BJ n'effectue des retours arrières que lorsqu'il s'agit d'une affectation cohérente qui ne peut être étendue d'une façon cohérente à la variable suivante.

Par contre, CBJ utilise un *ensemble de conflits* pour chaque variable instanciée qui contient toutes les variables instanciées avant la variable courante x_i qui sont en conflit avec au moins une valeur de x_i. Cet ensemble sera utilisé pour effectuer les retours arrières. En cas d'échec, CBJ effectue un retour arrière vers la dernière variable affectée de l'ensemble des conflits de la variable courante.

BJ et CBJ ont une complexité supérieure à celle de BT vu que l'analyse d'échecs demande un coût supplémentaire. Dans la littérature, nous trouvons d'autres algorithmes qui effectuent des retours en arrière non chronologiques comme le *backtracking dynamique* (noté DBT)[Ginsberg, 1993] et le Graph-based Backjumping (noté GBJ [Dechter, 1990]).

2.5.3 Backtrack avec enregistrement de nogood

L'arbre de recherche de l'algorithme Backtrack peut contenir de la redondance vu qu'il explore des parties de l'arbre de recherche qu'il a déjà visitées. Pour éviter ces redondances, certains algorithmes proposent d'enregistrer certaines informations comme les affectations des variables cohérentes (appelées aussi *good*) ou incohérentes (comme les *nogood*). Par exemple, le *Nogood Recording* (noté NR), proposé par Schiex et Verfaillie dans [Schiex and Verfaillie, 1993], s'appuie sur la mémorisation des nogoods [Doyle, 1979] chaque fois qu'un conflit se produit durant la recherche d'une solution. Il utilise ces nogoods pour éviter l'exploration des parties inutiles de l'arbre de recherche. BT et NR ont la même complexité en temps si on considère le coût d'enregistrement de nogoods.

Dans [Lecoutre et al., 2007c], Lecoutre et al. ont proposé un algorithme qui combine les redémarrages et l'enregistrement des nogoods. En effet, ce nouvel algorithme consiste à mémoriser seulement les nogoods qui précèdent les redémarrages.

2.5.4 Backtrack avec filtrage

Dans cette partie, nous présenterons quelques algorithmes qui appliquent le filtrage pendant la recherche pour garantir la cohérence de la prochaine instanciation. Le premier algorithme est le *Forward Checking* (noté FC) [Haralick and Elliot, 1980]. Après chaque instanciation, FC supprime les valeurs des domaines de variables en relation avec la dernière variable instanciée incompatibles avec la dernière affectation réalisée (de la ligne 8 à la ligne 12 de l'algorithme 2). Ceci garantit la cohérence de la prochaine affectation avec les affectations précédentes. Théoriquement, FC et BT

ont la même complexité temporelle (O(ed^n)), alors qu'en pratique FC obtient dans la plupart des cas de meilleurs résultats que BT.

Algorithme 2 : FC($P = (X, D, C)$: CSP ; $t = (v_1, v_2, \dots v_i)$: tuple) : Booléen

```
1  si (i = n − 1) alors
2      Retourner vrai /* t = (v_1, v_2, ... v_i) peut être étendue à une solution*/
3  sinon
4      Choisir une variable x_{i+1} à instancier
5      pour v_{i+1} ∈ D(x_{i+1}) faire
6          t' ← (v_1, v_2, ... v_i, v_{i+1})
7          Coherence ← Vrai
8          pour x_j ∈ X et c_{ji+1} ∈ C telle que i + 1 < j ≤ n faire
9              D'(x_j) ← {v_j ∈ D(x_j) : (v_j, v_{i+1}) ∈ R(c_{ji+1})}
10             si (D'(x_j) = ∅) alors
11                 Coherence ← Faux
12                 Quitter-la-Boucle-Pour
13         si (Coherence) alors
14             P' ← (X, D', C)
15             Coherence ← FC(P', t')
16     Retourner faux
```

Dans le cas binaire, FC applique la cohérence d'arc sur toute contrainte dont la variable courante figure dans sa portée. Dans le cas n-aire, il existe plusieurs façon pour définir l'ensemble de contraintes sur lequel la cohérence d'arc doit être vérifiée. Ceci justifie l'existence de la classe nFC$_i$ ($i = 0, 1 \dots 5$) qui recouvre l'application partielle ou totale de la cohérence d'arc généralisée sur un sous-ensemble de contraintes impliquant à la fois les variables affectées et les variables futures. Dans chaque cas, le filtrage est réalisé après chaque affectation de variable. Ainsi, la complexité des algorithmes de type nFC$_i$ dépend du coût du filtrage. Pour nFC$_5$ qui réalise le filtrage le plus puissant, la complexité est en O ($n^e ard$).

Le deuxième algorithme est le *real full lookahead* (noté RFL) [Nadel, 1988]. RFL applique la cohérence d'arc sur le sous-problème courant en supposant que les domaines des variables affectées sont réduits à la valeur affectée. Comme pour FC, ceci garantira la cohérence de l'affectation courante. Si on considère la meilleure complexité de AC qui est O(ed^2), la complexité de RFL sera de l'ordre de O(ed^{n+1}). nRFL est la version n-aire de RFL mais en utilisant la cohérence d'arc généralisée.

Le dernier est l'algorithme de maintien de la cohérence d'arc (noté MAC) [Sabin and Freuder, 1994]. Cet algorithme est légérement différent des algorithmes précédents vu qu'il développe un arbre binaire de recherche (contrairement à tous les algorithmes précédents qui développent au plus d branches au niveau de chaque nœud.). En effet, supposant que l'affectation de v_{i+1} à x_{i+1} produit un échec, après retour à l'affectation courante $(v_1, v_2, \dots v_i)$, et avant affectation d'une nouvelle valeur à x_{i+1}, la valeur v_{i+1} sera supprimée du domaine $D(x_{i+1})$, et un filtrage par AC sera réalisé. Tous les domaines des variables futures pourront être influencés par ce filtrage. Ce processus est appelé *réfutation* de la valeur v_{i+1} et peut être compris comme une extension de l'affectation courante $(v_1, v_2, \dots, v_i)$ en utilisant le négatif du nœud $x_{i+1} \leftarrow \neg v_{i+1}$. Le nombre de nœuds de l'arbre de recherche développé par MAC est au plus $2 \times \Sigma_{0 \leq i \leq n-1} d^i = 2\frac{d^n - 1}{d-1} \in$ O(d^{n-1}).

En terme de complexité, MAC et RFL ont la même complexité.

Dans la littérature, nous trouvons des algorithmes qui maintiennent :

– la cohérence d'arc et la cohérence de chemin directionnelle (noté MAC+) [Chmeiss and Sais, 2000],
– la cohérence de chemin inverse (appelé maintaining PIC) [Debruyne, 2000],
– la singleton cohérence d'arc (noté MSAC) [Lecoutre and Prosser, 2006],
– la cohérence d'arc généralisee (noté MGAC) [Cheng and Yap, 2006].

En pratique, il existe des implémentations *hybrides* qui combinent deux algorithmes. Par exemple,
– MAC-CBJ [Prosser, 1995],
– FC-CBJ [Prosser, 1993],
– FC-NR [Schiex and Verfaillie, 1993],
– FC-DBT [Verfaillie and Schiex, 1994],
– MAC-DBT [Jussien et al., 2000].

Dans la plupart des cas, les algorithmes de résolutions sont munis d'une heuristique pour avoir un ordre sur les variables qui semble être le plus adéquat pour trouver vite une solution.

2.5.5 Heuristiques d'ordonnancement

Les *heuristiques d'ordonnancement* (ou de sélection) de variables permettent de définir un ordre d'instanciation de variables. Elles sont utilisées pour guider les algorithmes de résolutions et donc accélérer l'obtention de la solution. Elles s'appuient sur le principe *First-Fail* énoncé dans [Haralick and Elliot, 1980] : « pour réussir, on essaie tout d'abord là où on va probablement échouer » (To succed, try first where you are almost likely to fail). Les heuristiques peuvent être intégrées dans les algorithmes de résolutions, cités dans la section précédente, et elles auront une influence sur la taille de l'arbre de recherche [Dechter and Meiri, 1989]. Il existe trois types d'heuristiques : les premières sont statiques, les deuxièmes sont dynamiques et les dernières sont adaptatives.

2.5.5.1 Les heuristiques statiques

Une heuristique est dite *statique* (notée SVO pour **S**tatic **V**ariable **O**rdering) si l'ordre sur les variables est calculé avant la recherche et qu'il ne varie pas tout au long de la résolution. Ce genre d'heuristiques est souvent basé sur quelques caractéristiques du graphe des contraintes (largeur, degré,...). Elles sont principalement utilisées par les algorithmes qui n'utilisent pas du filtrage.

Une première heuristique statique triviale consiste à considérer l'ordre lexicographique des variables. Pour les exemples 2.1, 2.2 et 2.3, les variables seront ordonnées de la façon suivante $(x_1, x_2, x_3, x_4, x_5, x_6)$. Une deuxième heuristique consiste à sélectionner la variable ayant la plus grande largeur de graphe (Maximum Width Ordering). Trouver cet ordre peut se réaliser en temps linéaire (O$(n + e)$). Pour l'exemple 2.2, l'ordre suivant $(x_3, x_2, x_6, x_5, x_4, x_1)$ nous donne la plus petite largeur de graphe à savoir 2. Une troisième heuristique, notée généralement *deg*, ordonne les variables selon leur degré décroissant (Maximum Degree Ordering) dans le graphe de contraintes [Dechter and Meiri, 1989]. Pour l'exemple 2.2, un des ordres (il n'est pas unique) à considérer est : $(x_2, x_3, x_5, x_6, x_4, x_1)$. Ces deux dernières heuristiques choisissent en premier les variables les plus contraintes, ce qui tend à réduire le nombre de retour en arrière pendant la recherche.

L'avantage de ces heuristiques est qu'elles ne sont pas difficiles à implémenter et qu'elles ont une complexité polynomiale. Par contre, elles ne prennent pas en considération les données courantes du problème.

2.5.5.2 Les heuristiques dynamiques

Pour les heuristiques *dynamiques* (notées DVO pour Dynamic Variable Ordering), la prochaine variable à instancier est choisie en fonction de certaines données courantes du problème (c'est-à-

dire au niveau d'un nœud de l'arbre de recherche) qui peuvent varier tout au long de la recherche. Ces heuristiques sont généralement intégrées dans les algorithmes utilisant du filtrage.

La première heuristique est appelée *min domain* (notée souvent *dom*). Pour cette heuristique, l'ordre sur les variables est relatif à la cardinalité des domaines : la variable ayant le plus petit domaine sera prioritaire [Haralick and Elliot, 1980]. Cette heuristique permet de minimiser le facteur de branchement dans l'arbre de recherche. Malheureusement, cette heuristique ne peut pas décider quand les variables ont toutes la même taille de domaine.

Une deuxième heuristique notée *ddeg* (ou *futdeg*) [Bessière and Régin, 1996] consiste à ordonner les variables selon leur degré dynamique. Donc, la variable ayant le plus grand nombre de variables voisines non instanciées sera sélectionnée. Cette heuristique permet de choisir la variable la plus contrainte dans le reste du problème. Mais elle ne peut pas départager les variables, par exemple, quand le graphe de contraintes est complet, car dans ce cas toutes les variables auront le même degré. Ceci conduit à déduire que les heuristiques dynamiques ne sont pas efficaces quand une sorte de symétrie existe dans un CSP. Par la suite, des heuristiques comme dom + deg [Frost and Dechter, 1995] (respectivement dom + $ddeg$ [Smith, 1999]) sont utilisées pour choisir la variable ayant le plus petit domaine courant et deg (resp. $ddeg$) pour briser les égalités. Ces heuristiques permettent aussi de minimiser le facteur de branchement. Certaines autres heuristiques comme dom/deg [Bessière and Régin, 1996] (respectivement $dom/ddeg$ [Bessière and Régin, 1996]) favorise les variables minimisant le rapport entre la taille du domaine courant et le degré (resp. le degré dynamique). Ces heuristiques permettent de découvrir les échecs le plus tôt possible. Plus récemment, une nouvelle heuristique (noté $wdeg$) basée sur le *degré pondéré* (dit aussi poids) des variables a été proposée dans [Boussemart et al., 2004]. En effet, cette heuristique affecte des poids aux contraintes relatifs aux échecs rencontrés tout au long de la recherche. Précisément, le poids d'une contrainte est égal au nombre des échecs dans lesquels elle est impliquée, c'est-à-dire, le nombre de fois où cette contrainte est violée. Par la suite, le poids d'une variable x_i est la somme des poids des contraintes c_j liant x_i et au moins une variable future. $dom/wdeg$ sera donc l'heuristique qui sélectionne la variable qui minimise ce rapport (taille du domaine / degré pondéré). Actuellement, les heuristiques basées sur le degré pondéré des variables sont les meilleures pour des raisons de coût et de facilité d'implémentation. De plus, ces heuristiques prennent en compte l'état courant de la recherche en plus de certaines informations sur les échecs rencontrés avant (ce principe est connu sous le nom last conflict).

2.5.5.3 Les heuristiques adaptatives

Les heuristiques adaptatives combinent des données dynamiques et statiques du problème. L'heuristique impact, par exemple, qui a été initialement introduite dans [Geelen, 1992] et par la suite réétudiée dans [Refalo, 2004]. Cette notion est basée sur l'effet d'une affectation pour réduire l'espace de recherche. Elle peut être utilisée pour ordonner les variables et les valeurs. La variable ayant le plus grand impact sera choisie en premier. Pour cette variable, on choisit la valeur ayant le plus petit impact.

2.6 Classes polynomiales

Bien que le problème CSP soit NP-complet, il n'en demeure pas moins qu'il existe des classes de CSP qui peuvent être à la fois *reconnues* et *résolues* en temps polynomial. Cela leur confère le statut de « classes polynomiales ».

Définition 2.38 (classe polynomiale)**.** *Une classe **polynomiale** pour CSP (dite aussi **traitable** ou tractable en anglais) est un ensemble (fini ou infini) $\mathcal{C}$ de CSP pour lesquels il existe deux algorithmes de complexité polynomiale A_R et A_S telle que pour tout CSP P, A_R peut décider si $P \in \mathcal{C}$ et si $P \in \mathcal{C}$, A_S résout P.*

Dans ce manuscrit, nous considérons la définition proposée dans [Gottlob et al., 2000]. Mais dans la littérature, certains auteurs considèrent l'existence d'un algorithme de résolution A_S suffisante pour que la classe soit polynomiale.

L'étude des classes polynomiales, pour les problèmes de satisfaction de contraintes, constitue depuis longtemps un domaine de recherche important qui s'avère aujourd'hui très actif. Généralement, une classe polynomiale est définie par une propriété déduite par l'identification de certaines restrictions sur quelques données du problème. Ceci justifie l'existence des classes polynomiales différentes. En effet, il existe plusieurs types de classes polynomiales :

- classes liées aux domaines,
- classes liées à la microstructure,
- classes structurelles,
- classes relationnelles,
- classes hybrides.

Théoriquement, des relations d'inclusions, d'équivalences ou d'intersections relient quelques classes polynomiales. Dans les paragraphes suivants, nous rappellerons les définitions des classes polynomiales les plus connues et qui seront évoquées dans les prochains chapitres.

2.6.1 Classes polynomiales liées aux domaines des variables

Dans cette section, nous parlerons des classes polynomiales liées aux domaines des variables et définies comme suit :

Définition 2.39 (classes polynomiales liées aux domaines)**.** *Les classes polynomiales liées aux domaines rassemblent les CSP ayant des restrictions sur les domaines et pouvant être reconnues et résolues en temps polynomial indépendamment de leurs contraintes et de leurs relations.*

Ici, nous rappelons une classe polynomiale liée à la *taille* des domaines [Dechter, 1992]. En fait, dans [Dechter, 1992], Dechter a défini une condition sur les domaines des variables d'un CSP d'arité quelconque pour que ce dernier soit globalement cohérent. En effet, un CSP est *globalement cohérent* quand toute affectation d'un sous-ensemble de variables peut être étendue à toutes les variables sans faire de retour arrière et en considérant tout ordre possible sur les variables [Dechter, 1992].

Théorème 2.1. *Étant donné un CSP P d'arité a ayant au maximum d valeurs par domaine, si P est fortement $(d(a-1)+1)$-cohérent, alors ce CSP est globalement cohérent.*

Pour le cas binaire, nous pouvons déduire le corollaire suivant :

Corollaire 2.1. *Étant donné un CSP binaire P et ayant maximum d valeurs par domaine, si P est fortement $(d+1)$-cohérent, alors le CSP est globalement cohérent.*

Les deux corollaires suivants sont déduits du théorème 2.1. Un CSP est dit *monovalent* si tous les domaines sont de taille un et il est dit *bivalent* si tous les domaines sont au plus de taille deux.

Corollaire 2.2. *Étant donné un CSP binaire monovalent P, si P est arc-cohérent, alors il est globalement cohérent.*

Corollaire 2.3. *Étant donné un CSP binaire bivalent P, si P est fortement chemin-cohérent, alors il est globalement cohérent.*

À l'exception des cas monovalent et bivalent, cette classe n'a pas été exploitée en pratique à cause de la complexité de l'application de la k-cohérence à savoir $O(n^k.d^k)$.

2.6.2 Classes polynomiales liées à la microstructure

Plusieurs classes polynomiales pour CSP binaires ont été proposées en travaillant sur le graphe de microstructure. Ces classes peuvent être définies comme suit :

Définition 2.40 (classes polynomiales liées à la microstructure)**.** *Les classes polynomiales liées à la microstructure présentent l'ensemble des CSP ayant un graphe de microstructure particulier permettant de les résoudre en temps polynomial indépendamment des autres paramètres du problème.*

Comme la microstructure est un graphe, alors les restrictions pour ces classes polynomiales font forcément référence à la théorie des graphes.

2.6.2.1 Microstructure triangulée

Dans cette section, nous consacrerons notre étude aux les graphes triangulés. Nous commençons tout d'abord par rappeler la définition des graphes triangulés qui est fondée sur la notion de cycle sans corde.

Définition 2.41 (un cycle)**.** *Un **cycle** est une chaîne de sommets dans un graphe dont le premier et le dernier sommet sont identiques. Il est dit sans corde si tout couple de sommets non consécutifs ne sont pas liés par une arête. La longueur d'un cycle est le nombre de sommets qui le constituent.*

Définition 2.42 (triangulé)**.** *Un graphe est dit **triangulé** s'il ne possède pas un cycle sans corde de longueur supérieure ou égale à quatre.*

Le graphe de contraintes de la figure 2.1 (page 23) est un graphe triangulé : il contient deux cycles de longueur quatre $(x_2, x_3, x_4, x_5, x_2)$ et $(x_2, x_5, x_3, x_6, x_2)$ mais avec corde $\{x_3, x_5\}$ et $\{x_2, x_3\}$.

Les graphes triangulés ont certaines propriétés intéressantes parmi lesquelles :

- la reconnaissance en temps polynomial (O($|V|^3|E|$)) [Chandrasekharan and Iyengar, 1988],
- un nombre polynomial (O($|V|$)) de cliques maximales [Fulkerson and Gross, 1965],
- un algorithme de complexité linéaire (O($|V| + |E|$)) pour la recherche des cliques maximales [Gavril, 1972].

En exploitant le résultat important de Gavril [Gavril, 1972] sur la recherche des cliques maximales, Jégou a montré, dans [Jégou, 1993a], que si le graphe de microstructure d'un CSP P est *triangulé*, alors ce CSP sera résolu en temps polynomial.

Notation 2.4. *La classe des CSP dont le graphe de microstructure est triangulé est notée CM (pour **C**hordal **M**icrostructure) [Cohen, 2003].*

Donc, un CSP dont la microstructure est un graphe triangulé peut être résolu en temps linéaire sachant que dans le cas général trouver les cliques maximales d'un graphe quelconque est un problème NP-difficile.

Par la suite, Cohen a utilisé la même approche dans [Cohen, 2003] pour montrer tout d'abord que la cohérence d'arc suffit pour résoudre un CSP dont la microstructure est triangulée. Ensuite, il a montré que les CSP, dont le complément de la microstructure est triangulé, définissent une classe polynomiale, résultat qui se déduit immédiatement de [Gavril, 1972] et de [Jégou, 1993a].

Notation 2.5. *La classe des CSP dont le complément du graphe de microstructure est triangulé est notée CCM (pour **C**hordal **C**omplement **M**icrostructure) [Cohen, 2003].*

Certes, les graphes triangulés disposent de propriétés intéressantes. Malheureusement, les graphes ne sont pas tous triangulés. Pour cette raison, plusieurs algorithmes de triangulation ont été proposés. En effet, la *triangulation* d'un graphe $G = (V, E)$ consiste à ajouter un ensemble d'arêtes E' à G de façon que le nouveau graphe $G' = (V, E \cup E')$ soit triangulé. Plus précisément, on rajoute des arêtes entre quelques sommets non-voisins qui participent à la formation d'un cycle sans corde de longueur supérieure ou égale à quatre. En pratique, il existe plusieurs algorithmes pour la triangulation des graphes comme MCS [1] [Berry et al., 2004], MNS [2], LexBFS [3] [Corneil, 2004], LexDFS [4]...

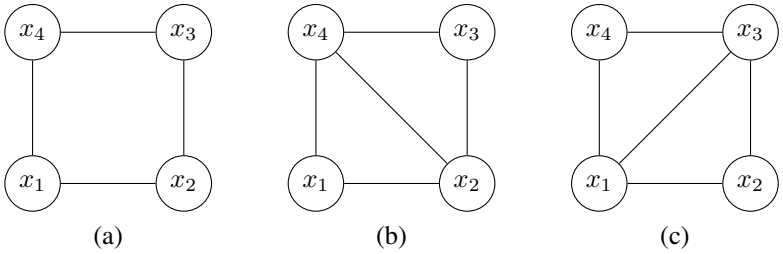

FIGURE 2.11 – Un graphe non-triangulé (a) et deux triangulations possibles (b) et (c).

La figure 2.11 (a) présente un graphe non-triangulé alors que (b) et (c) présentent deux triangulations (ce ne sont pas les seules) de (a). Ceci prouve que la triangulation d'un graphe n'est pas unique.

Après avoir triangulé le graphe de microstructure d'un CSP P, nous pouvons utiliser la méthode de *décomposition de domaines* pour le résoudre. Les différentes étapes de cette méthode sont décrites ci-dessous :

- construire la microstructure $\mu(P) = (V, E)$,
- trianguler $\mu(P)$ ($\mu'(P)$ le nouveau graphe de microstructure triangulé),
- trouver toutes les cliques maximales de $\mu'(P)$,
- pour toute clique Cl_k, tester si Cl_k couvre toutes les variables de X, alors le nouveau problème $P(Cl_k) = (X, D', C)$ (avec $P(Cl_k)$ le sous-problème relatif à la clique Cl_k, c'est-à-dire $\forall x_i, D'(x_i) = \{v_i \in D(x_i) \mid (v_i, x_i) \in Cl_k\}$) admet une solution. Sinon, la clique ne correspond pas à une solution (car elle contient une arête incohérente ajoutée par la procédure de la triangulation).

La complexité de cette méthode, au pire des cas, est égale à la complexité du problème initial. En effet, elle est de l'ordre de O($e.p.\delta^n$) : avec p le nombre des problèmes induits et δ la taille maximale des nouveaux domaines.

Après avoir décomposé le problème, le nombre total de solutions sera égal au nombre des solutions des sous-problèmes induits.

Théorème 2.2. *Soit $\mu'(P)$ le graphe triangulé de $\mu(P)$ et Cl = $\{Cl_1, ..., Cl_p\}$ l'ensemble des cliques maximales de $\mu'(P)$ alors :*
Solutions(P) = $\cup_{1 \leq i \leq n} Solutions(P(Cl_i))$.

1. MCS = Maximum Cardinality Search
2. MNS = Maximal Neighbourhood Search
3. LexBFS =Lexicographic Breadth FirstSearch
4. LexDFS =Lexicographic Depth First Search

2.6.2.2 Généralisation des graphes triangulés

Dans [Chmeiss and Jégou, 1997], Chmeiss et Jégou ont introduit les graphes CSG^k qui généralisent les graphes triangulés et certaines de leurs propriétés (notamment celle concernant le nombre polynomial de cliques maximales...). Les définitions et les notations suivantes permettent de bien définir cette nouvelle classe de graphes.

Définition 2.43. *Un sommet est dit simplicial si l'ensemble de ses voisins forme une clique.*

Notation 2.6. *Étant donné un graphe $G = (V, E)$, $G_{\{v_i,...,v_n\}}$ désigne un sous-graphe de G formé par l'ensemble de sommets $\{v_i, ..., v_n\}$.*

La figure 2.12 illustre un sous-graphe $G_{\{x_2,...,x_5\}}$ du graphe de la figure 2.1 formé par quatre sommets x_2, x_3, x_4 et x_5.

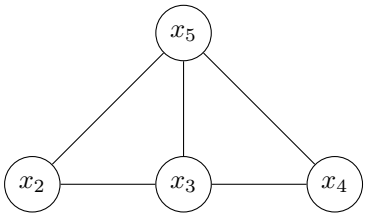

FIGURE 2.12 – $G_{\{x_2,...,x_5\}}$ un sous-graphe du graphe de contraintes de l'exemple 2.2.

Définition 2.44. *Un ordre $\sigma = [v_1, ..., v_n]$ est un schéma parfait d'élimination si chaque v_i est simplicial pour le sous-graphe induit par $G_{\{v_i,...,v_n\}}$ c'est-à-dire l'ensemble des sommets voisins à v_i et qui le précèdent dans l'ordre forment un graphe complet.*

Définition 2.45. *Les graphes CSG^k sont définis selon la valeur de k :*
- *pour $k = 0$, les graphes CSG^0 forment la classe des graphes complets.*
- *pour $k > 0$, les graphes CSG^k sont la classe des graphes $G = (V, E)$ tel qu'il existe un ordre $\sigma = [v_1, ..., v_n]$ sur V tel que pour $i = 1, ..., n$ le graphe $G(N^+(v_i))$ est CSG^{k-1} avec $N^+(v_i) = \{v_j \in V \mid \{v_i, v_j\} \in E, i < j\}$.*

Nous pouvons signaler que CSG^1 définit l'ensemble des graphes triangulés.

Chmeiss et Jégou ont prouvé que les graphes CSG^k ont au plus $|V|^k$ cliques maximales, et ils ont proposé un algorithme en O($|V|^{2(k-1)}(|V| + |E|)$) pour la recherche des cliques maximales dans un graphe CSG^k. Dans le même esprit des algorithmes de triangulation des graphes non-triangulés, un algorithme de k-triangulation a été proposé et qui consiste à rajouter des arêtes à un graphe quelconque afin de le rendre CSG^k. Ensuite, l'algorithme TR^k-Decomposition permet la résolution d'un CSP *(P)* par la méthode de décomposition des domaines après avoir appliquée la k-triangulation sur son graphe de microstructure.

En pratique, les algorithmes qui se basent sur la décomposition de la microstructure du CSP sont inefficaces vu le coût prohibitif de manipulation de la microstructure. Mais, pour des petites valeurs de k, ils ont eu des résultats interéssants. Pour $k = 2$ par exemple, l'algorithme Deuxtriang calcule la 2-triangulation d'un graphe G (en O($|V|^4$)) et Cliquesmax cherche toutes les cliques maximales dans ce graphe 2-triangulé (en O ($n.e$)). D'autres résultats pratiques ont montré que TR^2-Decomposition est plus efficace que MAC et FC dans le cas d'exécution parallèle des sous-problèmes générés [Ndiaye, 2005].

2.6.2.3 Microstructure parfaite

Plus récemment, dans la continuité de ces travaux, Salamon et Jeavons [Salamon and Jeavons, 2008] ont généralisé le résultat de [Jégou, 1993a], aux *graphes parfaits* (les graphes triangulés sont des graphes parfaits) en s'appuyant sur des résultats issus de la Conjecture de Berge sur les graphes parfaits [Chudnovsky et al., 2006, 2005]. Nous rappelons qu'un graphe est dit parfait s'il ne comporte ni un cycle ni le complément d'un cycle de longueur impaire et supérieure ou égale à cinq.

Notation 2.7. *La classe des CSP dont le graphe de microstructure est parfait est notée PM (pour **P**erfect **M**icrostructure).*

2.6.3 Classes polynomiales structurelles

Nous passons maintenant à des classes polynomiales différentes dites *classes polynomiales structurelles* qui sont définies au niveau de l'interaction entre les variables.

Définition 2.46 (classes polynomiales structurelles)**.** *Une classe polynomiale **structurelle** est un ensemble de CSP ayant des restrictions sur les portées de leurs contraintes (ou (hyper)graphe de contraintes) indépendamment de leurs relations.*

Nous nous intéressons ici aux CSP acycliques [Freuder, 1982], aux CSP ayant une largeur de (hyper)graphe bornée [Freuder, 1982, Jégou, 1993a] et aux classes de CSP ayant une largeur arborescente bornée par une constante [Gottlob et al., 2000].

2.6.3.1 Largeur de graphe

Cette notion, comme nous l'avons rappelé précédemment, a été introduite par Freuder dans [Freuder, 1982]. Elle a permis de définir une condition suffisante pour qu'un CSP puisse être résolue en temps polynomiale. La condition suivante est la suivante :

Théorème 2.3. *Si le graphe de contraintes est de largeur k et si le CSP est fortement $(k+1)$-cohérent, alors une solution peut être trouvée sans retour arrière.*

Pour un CSP satisfaisant ces conditions, l'algorithme backtrack peut trouver une solution sans retour arrière en considérant l'ordre sur les variables qui a donné la largeur k à ce CSP. À partir de ce théorème, nous pouvons déduire les deux corollaires suivants en considérant $k = 1$ et $k = 2$.

Corollaire 2.4. *Pour les CSP dont le graphe de contraintes est acyclique (la largeur est égale à 1, on note par $TREE$ l'ensemble de ces CSP), la cohérence d'arc suffit pour garantir qu'une solution peut-être trouvée sans retour arrière.*

Corollaire 2.5. *Étant donné un CSP P, si la largeur de son graphe de contraintes est égale à 2 et si le CSP vérifie la cohérence de chemin, alors une solution peut être trouvée sans retour arrière.*

D'après les deux corollaires précédents, on constate qu'en calculant la largeur du graphe de contrainte et en vérifiant le niveau de cohérence du CSP, ce CSP peut être résolu en temps polynomial. Cette méthode aussi n'a pas été exploitée en pratique à cause de la complexité de la k-cohérence O($n^k.d^k$) sauf dans le cas où le graphe de contrainte est arborescent.

2.6.3.2 CSP non-binaire acyclique

Pour le cas binaire, Freuder a montré dans [Freuder, 1982] que si un CSP binaire est *acyclique* (dit aussi *arborescent*) alors la cohérence d'arc suffira pour trouver une solution sans backtrack. Pour le cas n-aire, il existe plusieurs formes d'acyclicités [Beeri et al., 1983], nous en rappelons quelques-unes ainsi que leurs relations. Nous commençons tout d'abord par les α-acycliques. Dans la littérature, nous trouvons plusieurs définitions équivalentes, ici nous en énonçons seulement les deux suivantes :

Définition 2.47 (hypergraphe conforme [Beeri et al., 1983])**.** *Un hypergraphe est dit conforme si toutes les cliques maximales de sa 2-section correspondent à une hyperarête dans l'hypergraphe original.*

L'hypergraphe de la figure 2.2 est conforme car les 4 cliques maximales de la 2-section correspondent exactement aux hyperarêtes de l'hypergraphe.

Théorème 2.4 (hypergraphe α-acyclique [Beeri et al., 1983])**.** *Un hypergraphe est α-acyclique s'il est conforme et si son graphe primal est triangulé.*

L'exemple 2.1 a un hypergraphe α-acyclique car il satisfait les deux conditions du théorème 2.4.

Une deuxième caractérisation des hypergraphes α-acycliques, appelée « *running intersection property* » a été proposée dans [Beeri et al., 1983] et elle est définie comme suit :

Définition 2.48. *$H = (V, E)$ est un hypergraphe **α-acyclique**, si et seulement si, il existe un ordre sur les contraintes $(c_1, ..., c_e)$ tel que*

$$\forall k, 1 < k \leq e, \exists j < k, (S(c_k) \cap \bigcup_{i=1}^{k-1} S(c_i)) \subseteq S(c_j).$$

Cette propriété garantit l'existence d'un ordre pour lequel si $S(c_k)$ appartient à un chemin entre $S(c_i)$ et $S(c_j)$, alors $S(c_i) \cap S(c_j) \subseteq S(c_k)$. Nous passons maintenant aux hypergraphes β-acycliques qui sont définis de la façon suivante :

Définition 2.49 (hypergraphe β-acyclique [Graham, 1979])**.** *Un hypergraphe $H = (V, E)$ est β-acyclique s'il ne contient pas un cycle de Graham. Une séquence $(E_1, ..., E_m, E_{m+1})$ avec $m \geq 3$ telle que $(E_1, ..., E_m)$ sont distincts et $E_1 = E_{m+1}$ est un cycle de Graham si chaque $\Delta_i = E_i \cap E_{i+1}$ $(1 \leq i \leq m)$ est non-vide, et $\forall i \neq j$, Δ_i Δ_j sont incomparables (c'est-à-dire $\Delta_i \not\subseteq \Delta_j$ et $\Delta_j \not\subseteq \Delta_i$).*

Les hypergraphes β-acycliques sont appelés aussi *super-balanced* hypergraph et *totally-balanced* hypergraph [Lehel, 1985]. L'hypergraphe de l'exemple 2.1 est β-acyclique car il ne peut pas contenir un cycle de Graham. En effet, le tableau 2.3 explique pour toutes les combinaisons pourquoi un cycle de Graham ne peut exister.

Définition 2.50 (hypergraphe γ-acyclique)**.** *Un hypergraphe $H = (V, E)$ est γ-acyclique ssi son **graphe d'intersection** (line graph) ne contient pas de cycle. Le graphe d'intersection d'un hypergraphe $H = (V, E)$ est un graphe $B = (I, J)$ avec I l'ensemble des intersections non-vides des hyperarêtes de H et J et l'ensemble des arêtes qui existent chaque fois que l'intersection entre deux sommets x et y de I n'est pas vide.*

L'hypergraphe de l'exemple 2.1 n'est pas γ-acyclique car il existe un cycle de longueur trois dans le graphe d'intersection (voir figure 2.13).

La définition des hypergraphes α-acycliques et γ-acycliques provient de la théorie des graphes alors que la définition de β-acyclique provient de la théorie des bases de données relationnelles.

La relation entre les différentes formes d'acyclicité est donnée par le théorème 2.5.

Séquence (Ordre sur les portées de contraintes)	Explication
$\{S(c_1), S(c_2), S(c_3), S(c_4), S(c_1)\}$	$S(c_3) \cap S(c_4) \subset S(c_2) \cap S(c_3)$
$\{S(c_1), S(c_2), S(c_4), S(c_3), S(c_1)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_1), S(c_3), S(c_2), S(c_4), S(c_1)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_1), S(c_3), S(c_4), S(c_2), S(c_1)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_1), S(c_4), S(c_3), S(c_2), S(c_1)\}$	$S(c_3) \cap S(c_4) \subset S(c_2) \cap S(c_3)$
$\{S(c_1), S(c_4), S(c_2), S(c_3), S(c_1)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_2), S(c_1), S(c_3), S(c_4), S(c_2)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_2), S(c_1), S(c_4), S(c_3), S(c_2)\}$	$S(c_3) \cap S(c_4) \subset S(c_2) \cap S(c_3)$
$\{S(c_2), S(c_3), S(c_1), S(c_4), S(c_2)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_2), S(c_3), S(c_4), S(c_1), S(c_2)\}$	$S(c_3) \cap S(c_4) \subset S(c_2) \cap S(c_3)$
$\{S(c_2), S(c_4), S(c_3), S(c_1), S(c_2)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_2), S(c_4), S(c_1), S(c_3), S(c_2)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_3), S(c_1), S(c_2), S(c_4), S(c_3)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_3), S(c_1), S(c_4), S(c_2), S(c_3)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_3), S(c_2), S(c_1), S(c_4), S(c_3)\}$	$S(c_1) \cap S(c_4) = S(c_1) \cap S(c_2)$
$\{S(c_3), S(c_2), S(c_4), S(c_1), S(c_3)\}$	$S(c_1) \cap S(c_4) \subset S(c_4) \cap S(c_2)$
$\{S(c_3), S(c_4), S(c_2), S(c_1), S(c_3)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_3), S(c_4), S(c_1), S(c_2), S(c_3)\}$	$S(c_1) \cap S(c_4) = S(c_1) \cap S(c_2)$
$\{S(c_4), S(c_1), S(c_2), S(c_3), S(c_4)\}$	$S(c_1) \cap S(c_4) = S(c_1) \cap S(c_2)$
$\{S(c_4), S(c_1), S(c_3), S(c_2), S(c_4)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_4), S(c_2), S(c_1), S(c_3), S(c_4)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_4), S(c_2), S(c_3), S(c_1), S(c_4)\}$	$S(c_1) \cap S(c_3) = \emptyset$
$\{S(c_4), S(c_3), S(c_2), S(c_1), S(c_4)\}$	$S(c_3) \cap S(c_4) \subset S(c_2) \cap S(c_3)$
$\{S(c_4), S(c_3), S(c_1), S(c_2), S(c_4)\}$	$S(c_1) \cap S(c_3) = \emptyset$

TABLE 2.3 – Preuve d'inexistence de cycle de Graham

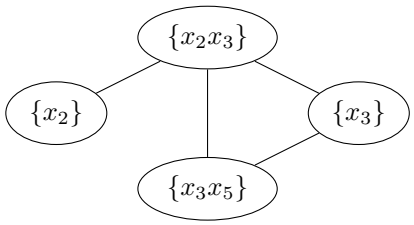

FIGURE 2.13 – Le graphe d'intersection de l'exemple 2.1.

Théorème 2.5 ([Fagin, 1983]). *γ-acyclique $\subsetneq$ β-acyclique $\subsetneq$ α-acyclique.*

À l'issu d'un résultat important dans [Beeri et al., 1983] sur la polynomialité des hypergraphes acycliques , il a été déduit que l'ensemble des CSP ayant un hypergraphe α-acyclique définissent une classe polynomiale et que l'inter-cohérence (ou l'hyper-2-cohérence) suffira pour trouver une solution sans backtrack.

En pratique, il existe une méthode, dite coupe-cycle (Cycle CutSet) [Dechter and Pearl, 1987a], qui consiste à trouver un ensemble de sommets, dit aussi coupe-cycle, dans le graphe (ou l'hypergraphe) de contraintes dont la suppression induit un graphe (ou un hypergraphe) acyclique. Si toutes les variables qui appartiennent à un ensemble coupe-cycle d'un graphe de contraintes sont affectées en satisfaisant les contraintes associées, nous propageons ces affectations par la cohérence d'arc. Si le CSP obtenu après filtrage par AC ne possède aucun domaine vide, alors le CSP est satisfiable. Étant donné un CSP d'arité quelconque, la complexité de cette méthode est O($e.d^{k+2}$) avec k la taille de l'ensemble coupe cycle.

2.6.3.3 Généralisation de théorème de Freuder sur la largeur de graphe

Dans [Jégou, 1993a], Jégou a généralisé le théorème de Freuder sur la largeur des graphes de contraintes aux CSP n-aires en se basant sur l'hypergraphe de contraintes. La largeur d'un hypergraphe de contraintes a été définie comme suit :

Définition 2.51 (largeur d'un hypergraphe). *Étant donnés un hypergraphe $H = (V, E)$ et O l'ensemble de tous les ordres possibles sur les hyperarêtes (dans le cas des CSP, ce sont les portées des contraintes) de E. Pour un ordre $\tau \in O$,*

- *la largeur d'une hyperarête E_i est le nombre d'intersections maximales avec les hyperarêtes qui la précèdent dans l'ordre : $|\{E_i \cap E_j / j < i \wedge \nexists k < i / E_i \cap E_j \subsetneq E_i \cap E_k\}|$,*
- *la largeur d'ordre τ est le maximum des largeurs des hyperarêtes,*
- *la largeur d'un hypergraphe est le minimum des largeurs de tous les ordres.*

Pour l'exemple 2.1, la largeur de son hypergraphe est égale à 1 en considérant l'ordre lexicographique croissant.

Comme pour le cas binaire, les CSP non-binaires ayant une largeur d'hypergraphe bornée définissent une classe polynomiale.

Théorème 2.6. *Étant donnée un CSP n-aire P, si la largeur de son hypergraphe de contraintes est k et s'il est hyper-$(k+1)$-cohérent, alors il est cohérent.*

Pour un hypergraphe $H = (V, E)$ et un ordre sur E, nous rappelons que trouver la largeur de cet ordre peut se réaliser en temps polynomial. Par contre, trouver l'ordre qui permet de calculer le minimum des largeurs est NP-complet.

2.6.3.4 Largeur arborescente bornée

La notion de *décomposition arborescente* a été introduite par Robertson et Seymour dans [Robertson and Seymour, 1986]. Elle consiste à décomposer le graphe de contraintes en un ensemble de sous-graphes connectés dans un arbre.

Formellement, la décomposition arborescente est définie comme suit :

Définition 2.52 (décomposition arborescente)**.** *Étant donné un graphe $G = (V, E)$, une **décomposition arborescente** de G est une paire (B, T) avec $T = (I, F)$ un arbre et $B = \{B_i : i \in I\}$, une famille de sous-ensembles de X appelés* clusters*, telle que chaque cluster B_i est un nœud de T et vérifie :*

- $\cup_{i \in I} B_i = V$,
- $\forall \{x_i, x_j\} \in E, \exists i \in I$ *tel que* $\{x_i, x_j\} \subseteq B_i$*, et*
- *si x_i appartient à deux clusters B_l et B_k, alors x_i appartient à tous les clusters du chemin reliant B_l à B_k.*

La largeur arborescente de la décomposition arborescente (B, T) est égale à la taille du plus grand grand cluster moins un ($max_{i \in I}|B_i| - 1$). La largeur d'arbre (ou bien tree-width), notée w, est la plus petite largeur de toutes les décompositions arborescentes de G.

Définition 2.53 (largeur arborescente bornée)**.** *Soit k un entier positif, nous notons BTW_k l'ensemble des CSP ayant une largeur arborescente bornée par k.*

Étant donnés un CSP P et son graphe de contraintes G, déterminer la largeur arborescente de G est un problème NP-difficile. Par ailleurs, si la largeur arborescente est bornée, alors P peut être résolu en temps polynomial (O($e.d^{w^+ + 1}$), où $w^+ + 1$ est la taille du plus grand sous-ensemble de B).

Plusieurs méthodes structurelles ont employé la décomposition arborescente pour atteindre une complexité polynomiale. Parmi ces méthodes nous pouvons citer le regroupement en arbre et le regroupement cyclique. Dans la suite, nous rappellerons les différentes étapes de ces méthodes ainsi que la complexité de chacune.

Le *regroupement en arbre* (Tree-Clustering) est une des méthodes basée sur la décomposition arborescente. Elle a été proposée par Dechter et Pearl dans [Dechter and Pearl, 1989] et elle consiste à effectuer certaines transformations à un CSP afin d'avoir un graphe acyclique dit *arbre de jointure* (join tree [Beeri et al., 1983]).

Le Tree-Clustering est inopérant vu que son temps d'exécution est très important et l'espace mémoire utilisé pour le stockage des solutions des sous-problèmes (relatifs aux clusters) est prohibitif. En effet, il a une complexité temporelle exponentielle en fonction de la largeur arborescente w de la décomposition considérée ($O(|P|.d^{w+1})$) et une complexité spatiale en O($n.a.d^s$) avec s la taille de la plus grande intersection entre deux clusters). Ceci qui explique son inefficacité en pratique bien qu'il possède la meilleure complexité théorique [Jégou and Terrioux, 2003].

En pratique, l'algorithme BTD (**B**acktracking with **T**ree **D**ecomposition [Jégou and Terrioux, 2003]), qui est une approche énumérative, exploite une décomposition arborescente du graphe de contraintes. L'algorithme utilise les notions de « good » et de « nogood » pour éviter les redondances en parcourant l'espace de recherche. BTD a des bons résultats en pratique pour des valeurs bornées de *w*. Il existe plusieurs extensions pour cet algorithme tel que FC-BTD [Jégou and Terrioux, 2003] et RFL-BTD [Jégou and Terrioux, 2003] et MAC-BTD [Jégou and Terrioux, 2014].

Dans [Darwiche, 2001], Darwiche a proposé une méthode similaire à BTD, appelée *Recursive Conditioning* et notée RC) qui utilise un espace de recherche linéaire. RC s'appuie sur la notion de *dtree* qui correspond à un graphe binaire complet. En effet, dtree construit récursivement à partir d'un séparateur (cutset) qui décompose le graphe de contraintes en un ensemble de sous-graphes connexes. Dans [Darwiche and Hopkins, 2001], les auteurs ont montré l'existence d'une équivalence entre dtree et la décomposition arborescente.

Dans [Dechter and Mateescu, 2004], Dechter et Mateescu ont proposé la méthode *AND/OR Search* qui a aussi une complexité spatiale linéaire. Cette méthode est basée sur le calcul d'un *pseudo-arbre* du graphe de contraintes et la construction d'un arbre de recherche AND/OR associé. La complexité AND/OR Search est exponentielle en fonction de la hauteur du pseudo-arbre.

Dans [Jégou, 1990], la méthode *regroupement cyclique* (Cyclic Clustering) a été introduite par Jégou et qui fonctionne de la manière suivante : il cherche un sous-graphe $G' = (E', V')$ triangulé maximal dans le graphe de contrainte $G = (E, V)$. Puis, il cherche les cliques maximales de *G'* et il résout les sous-problèmes associés à ces cliques. Après, il détermine l'ensemble coupe-cycle. Enfin, il lance la méthode coupe cycle sur le nouveau CSP [Jégou and Terrioux, 2004].

2.6.3.5 Largeur hyperarborescente bornée

Dans [Gottlob et al., 2000], Gottlob et al. ont proposé une nouvelle approche, dite *décomposition hyperarborescente*. Si les clusters dans la méthode de la décomposition sont représentés par les variables, dans la décomposition hyperarborescente ils seront représentés par les contraintes. En fait, il s'agit d'une généralisation de la décomposition arborescente. La définition s'appuie sur la notion d'*hyperarbre*.

Définition 2.54 (hyperarbre)**.** *Étant donné un hypergraphe $H = (X, C)$, un hyperarbre de H est un triplet (T,χ,λ) où $T = (S, A)$ est un arbre enraciné qui associe à chaque sommet $s \in S$ deux ensembles $\chi(s) \subseteq X$ et $\lambda(s) \subseteq C$. Si $T' = (S', A')$ est un sous-arbre de T, alors $\chi(T') = \cup_{s' \in S'} \chi(s')$. On note $sommets(T)$, l'ensemble S des sommets de T, et $racine(T)$ la racine de T. En outre, quel que soit $s \in S$, T_s est le sous-arbre de T enraciné en s.*

Définition 2.55 (décomposition hyperarborescente)**.** *Étant donné un hypergraphe $H = (V, E)$, une décomposition hyperarborescente de H est un hyperarbre HD=(T,χ,λ) qui vérifie les conditions suivantes :*

1. $\forall e \in E, \exists s \in$ *sommets (T) tel que* $e \subseteq \chi(s)$
2. $\forall x \in X$, *l'ensemble* $\{s \in$ *sommets(T) tel que* $x \in \chi(s)\}$ *induit un sous-arbre (connexe) de T,*
3. $\forall s \in$ *sommets(T),* $\chi(s) \subseteq \cup_{c \in \lambda(s)} c$,
4. $\forall s \in$ *sommets(T),* $\cup_{c \in \lambda(s)} c \cap \chi(T_s) \subseteq \chi(s)$.

La largeur h d'une décomposition hyperarborescente HD = (T,χ,λ) est égale au $max_{s \in sommets(T)} |\lambda(s)|$. La largeur hyperarborescente h^* de H est la largeur minimum sur toutes ses décompositions hyperarborescentes. Nous rappelons que $h \leq w$ ce qui permet de déduire que la décomposition hyperarborescente est théoriquement meilleure que la décomposition arborescente [Gottlob et al., 2000].

Étant donné un CSP $P = (X, D, C)$, les méthodes de résolution basées sur la décomposition hyperarborescente comportent principalement trois étapes :

1. déterminer une décomposition hyperarborescente de P,
2. résoudre chaque cluster en faisant la jointure des relations associées aux différentes contraintes d'un cluster,
3. résoudre le CSP acyclique obtenu.

La complexité temporelle de la première étape est O(e^{2h^*}) si on considère l'algorithme opt-k-decomp proposé dans [Gottlob et al., 1999]. La complexité temporelle de la deuxième étape est O($S.r^h$) et la complexité spatiale dans le pire des cas est O(r^h).

2.6.4 Classes polynomiales relationnelles

Dans cette sous-section, nous illustrerons certaines classes polynomiales dites *relationnelles* :

Définition 2.56 (classes polynomiales relationnelles)**.** *Une classe polynomiale* ***relationnelle*** *(appelée aussi Language-Based) est un ensemble de CSP identifiés par certaines restrictions sur les relations indépendamment de la structure.*

C'est par exemple le cas des relations « 0-1-tous » (ZOA) [Cooper et al., 1994], « max-closed » [Jeavons and Cooper, 1995], « convexes par rangées » [van Beek and Dechter, 1995] et les relations « renommables monotones à droite » [Cooper et al., 2010]. Nous détaillons maintenant ces classes que nous utiliserons dans les prochains chapitres.

2.6.4.1 Les relations 0-1-tous

Cooper et al. ont prouvé que tout CSP binaire dont les relations des contraintes vérifient la propriété 0-1-tous (notée ZUT ou ZOA en anglais pour Zero/One/All) peut être résolu en temps polynomial [Cooper et al., 1994]. Nous utilisons les deux notations suivantes pour définir ZOA.

Notation 2.8. *Étant donnée une contrainte* c_{ij}*,* $D(x_j)[x_i] = \{v_j \in D(x_j) \mid \exists v_i \in D(x_i), (v_i, v_j) \in R(c_{ij})\}$.

Notation 2.9. *Étant donnée une contrainte* c_{ij}*,* $R(c_{ji}) = \{(v_j, v_i) : (v_i, v_j) \in R(c_{ij})\}$.

Définition 2.57 (0-1-tous [Cooper et al., 1994])**.** *Étant donné un CSP binaire* $P = (X, D, C)$*, une relation* $R(c_{ij})$ *est dite* ***0-1-tous*** *directionnelle si pour chaque valeur* $v_i \in D(x_i)$*,* $R(c_{ij})$ *vérifie l'une des conditions suivantes :*

- *(ZERO) pour chaque valeur* $v_j \in D(x_j)[x_i], (v_i, v_j) \notin R(c_{ij})$*,*
- *(UN) il y a une valeur unique* $v_j \in D(x_j)[x_i]$ *telle que* $(v_i, v_j) \in R(c_{ij})$*,*
- *(TOUS) pour chaque valeur* $v_j \in D(x_j)[x_i], (v_i, v_j) \in R(c_{ij})$*.*

La relation $R(c_{ij})$ *est dite 0-1-tous si* $R(c_{ij})$ *et* $R(c_{ji})$ *sont 0-1-tous directionnelles.* P *est dit 0-1-tous si* $\forall i \neq j$*,* $R(c_{ij})$ *est 0-1-tous.*

Notation 2.10. *Nous noterons* ZOA *l'ensemble des CSP vérifiant la propriété 0-1-tous.*

Dans le but de caractériser les relations ZOA, nous rappelons les trois configurations suivantes :

Définition 2.58 (complète)**.** *Une relation* $R(c_{ij})$ *est dite* complète *si elle est égale à* $D(x_i)[x_j] \times D(x_j)[x_i]$.

Définition 2.59 (bijection [David, 1993])**.** *Une relation* $R(c_{ij})$ *est dite une* bijection *si chaque valeur de* $D(x_i)[x_j]$ *est compatible avec une seule valeur de* $D(x_j)[x_i]$ *(et inversement).*

Définition 2.60 (double éventail)**.** *Une relation* $R(c_{ij})$ *est dite* double éventail *(*two-fan*) si une seule valeur de* $D(x_i)[x_j]$ *est connecté à toutes les autres valeurs de* $D(x_j)[x_i]$ *(et inversement).*

Si pour une relation double éventail $R(c_{ij})$ on a en plus $D(x_i)[x_j] = D(x_i)$ et $D(x_j)[x_i] = D(x_j)$, alors $R(c_{ij})$ est dite double éventail fort.

La figure 2.14 présente un exemple de ces trois configurations :

Un résultat important de [Cooper et al., 1994] montre que toute relation ZOA est soit complète, soit une bijection ou soit un double éventail. Ensuite, Cooper et al. ont montré que si une relation ne satisfait pas ZOA, alors elle contient une de trois configurations suivantes (voir figure 2.15).

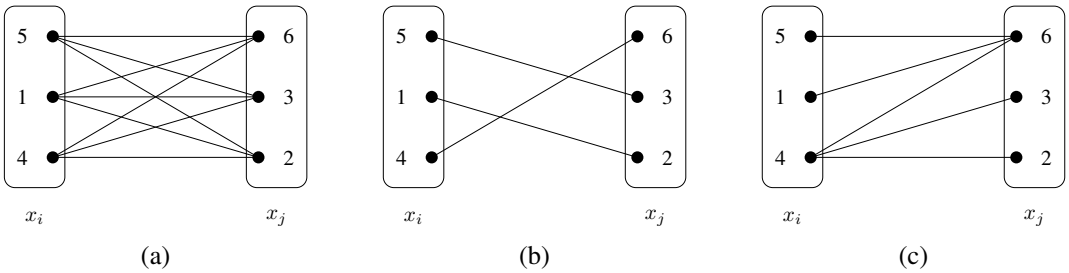

FIGURE 2.14 – *Trois configurations de la propriété ZOA : (a) une relation complète (b) une bijection et (c) un double éventail*

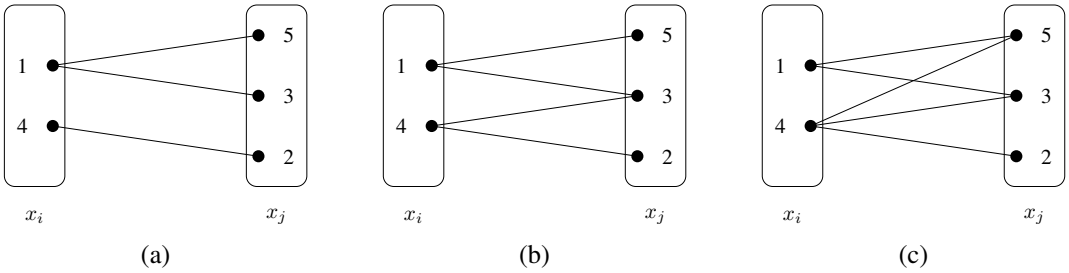

FIGURE 2.15 – *Les trois configurations possibles d'une relation qui n'est pas ZOA.*

Dans [Zhang et al., 1999], un algorithme de complexité temporelle O($e(d + n)$) a été proposé pour la résolution des CSP dont les relations des contraintes satisfont ZOA.

2.6.4.2 Les CSP fonctionnels

Les CSP fonctionnels ont été proposés par David dans [David, 1993]. Il s'agit d'une sous-classe de ZOA.

Définition 2.61 (Fonctionnel [David, 1993])**.** *Un CSP binaire P est dit fonctionnel si toutes les relations satisfont la première ou la deuxième condition de ZOA (ZERO et UN).*

Notation 2.11. *Nous noterons FUN l'ensemble des CSP binaires fonctionnels.*

Par définition, la classe des CSP binaires fonctionnels est une sous-classe de ZOA.

Proposition 2.2. $FUN \subsetneq ZOA$.

Cette classe a été généralisée par la suite par Cooper et al. dans [Cohen et al., 2011] pour les CSP d'arité quelconque.

Définition 2.62 (CSP incrémentalement fonctionnel [Cohen et al., 2011])**.** *Un CSP $P = (X, D, C)$ est dit **incrémentalement fonctionnel** (**incrementally functional** en anglais) s'il existe un ordre $<$ sur les variables, tel que pour $1 \leq i < n$, chaque solution de $P[\{x_1, \ldots, x_i\}]$ peut s'étendre au plus à une solution de $P[\{x_1, \ldots, x_{i+1}\}]$.*

Notation 2.12. *Nous noterons $IFUN$ l'ensemble des CSP incrémentalement fonctionnels.*

2.6.4.3 Max-Closed

Dans [Jeavons and Cooper, 1995], Jeavons et Cooper ont montré que s'il existe un ordre sur les valeurs pour lequel toutes les relations d'un CSP sont **M**ax-**C**losed, alors le CSP peut être résolu en temps polynomial.

Définition 2.63 (Max-closed [Jeavons and Cooper, 1995])**.**
*Étant donné un CSP $P = (X, D, C)$, P est dit **max-closed** s'il existe un ordre sur les valeurs tel que pour chaque contrainte c_i d'arité a_i,*
$\forall (v_1, v_2, \ldots, v_{a_i}), (v'_1, v'_2, \ldots, v'_{a_i}) \in R(c_i)$, $(max(v_1, v'_1), max(v_2, v'_2), \ldots, max(v_{a_i}, v'_{a_i})) \in R(c_i)$.

Pour la figure 2.16, nous considérons l'ordre naturel sur les entiers. Pour la figure 2.16 (a), on a (1,3) et (4,2) $\in R(c_{ij})$ mais (max(1,4), max(3,2))=(4,3) $\notin R(c_{ij})$. Par conséquent, la relation n'est pas max-closed. Pour la figure 2.16 (b), la relation est max-closed car (1,3) et (2,4) $\in R(c_{ij})$ et (max(1,2), max(3,4))=(2,4) $\in R(c_{ij})$, (1,3) et (2,3) $\in R(c_{ij})$ et (max(1,2), max(3,3))=(2,3) $\in R(c_{ij})$ et enfin (2,3) et (2,4) $\in R(c_{ij})$ et (max(2,2), max(3,4))=(2,4) $\in R(c_{ij})$.

Notation 2.13. *Nous noterons MC l'ensemble des CSP max-closed.*

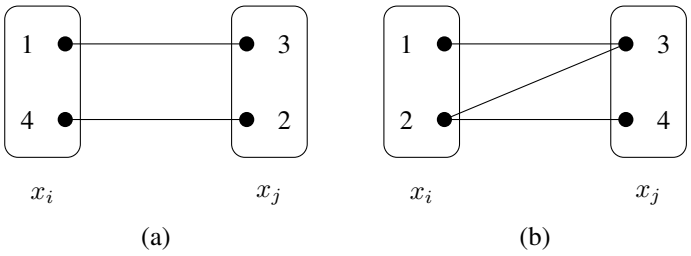

FIGURE 2.16 – *La propriété max-closed.*

De la même façon, les relations min-closed ont été définies. La classe des CSP satisfaisant max-closed définissent une classe polynomiale pour laquelle l'inter-cohérence est une procédure de décision [Jeavons and Cooper, 1995]. Dans [Cooper et al., 2010], il a été cité que tout CSP satisfaisant max-closed peut être résolu en appliquant la cohérence d'arc généralisée [Mohr and Masini, 1988] et en choisissant le plus grand élément de chaque variable.

2.6.4.4 Les relations convexes par rangée

Cette classe polynomiale (Convexe par rangée) a été introduite par Van Beek et Dechter dans [van Beek and Dechter, 1995] pour les CSP binaires, ensuite, une généralisation a été proposée pour les CSP n-aires [van Beek and Dechter, 1995].

Définition 2.64 (convexe par rangée)**.** *Étant donné un CSP binaire $P = (X, D, C)$, P est dit **convexe par rangée** (**row-convex** en anglais) par rapport à un ordre $<$ sur les variables et à un ordre sur les valeurs, si, pour chaque contrainte c_{ij} de C avec $x_i < x_j$, $\forall v_i \in D(x_i)$, $\{v_j \in D(x_j) | (v_i, v_j) \in R(c_{ij})\} = [a_j..b_j]$ pour $a_j,\ b_j \in D(x_j)$ où $[a_j..b_j]$ représente les valeurs de $D(x_j)$ entre a_j et b_j par rapport à l'ordre sur les valeurs.*

Notation 2.14. *Nous noterons RC l'ensemble des CSP convexes par rangée.*

En considérant la matrice d'interaction des valeurs dont les zéros (resp. les uns) expriment l'incompatibilité (resp. la compatibilité) entre deux valeurs , vérifier la propriété convexe par rangée revient à tester l'inexistence d'un zéro entre deux un pour un ordre fixé sur les valeurs.

$$M_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array} \quad M_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$

En considérant tout ordre possible sur les valeurs, la matrice M_1 représente une relation non convexe par rangée et M_2 représente une relation convexe par rangée.

Le concept de relation convexe par rangée généralise le concept des relations *fonctionnelles* et définit une classe polynomiale pour laquelle la cohérence de chemin est une procédure de décision. Par la suite, Zhang et Yap ont proposé une nouvelle propriété, dite *tree convexe* [Zhang and Yap, 2003], qui présente une extension du concept convexe par rangée [Zhang and Bao, 2009]. Comme pour RC, la cohérence de chemin est une procédure de décision pour la classe des CSP « tree convexe ». Pour certaines de ses sous-classes, la cohérence d'arc suffit pour garantir la cohérence globale du CSP [Zhang and Freuder, 2008].

Dans [Jeavons and Cooper, 1995], Jeavons et Cooper ont montré que tout CSP binaire satisfaisant à la fois min-closed et max-closed est aussi convexe par rangée. La classe des CSP convexes par rangée capture également la classe des CSP satisfaisant ZOA.

Proposition 2.3. $ZOA \subsetneq RC$.

2.6.4.5 Les relations renommables monotones à droite

La classe des relations *renommables monotones à droite* a été introduite par Cooper et al. dans [Cooper et al., 2010]. Pour cette classe, la cohérence d'arc est une procédure de décision.

Définition 2.65 (relation renommable monotones à droite [Cooper et al., 2010])**.** *Un CSP binaire $P = (X, D, C)$ est dit renommable monotone à droite **renamable right monotone** par rapport à un ordre $<$ sur les variables, si, pour $2 \leq j \leq n$, chaque domaine $D(x_j)$ peut-être ordonné par $\lessdot_j$ de telle sorte que pour chaque contrainte c_{ij} de C avec $x_i < x_j$, $\forall v_i \in D(x_i), v_j, v'_j \in D(x_j)$, si $(v_i, v_j) \in R(c_{ij})$ et $v_j \lessdot_j v'_j$ alors $(v_i, v'_j) \in R(c_{ij})$.*

Notation 2.15. *Nous noterons RRM l'ensemble de ces CSP.*

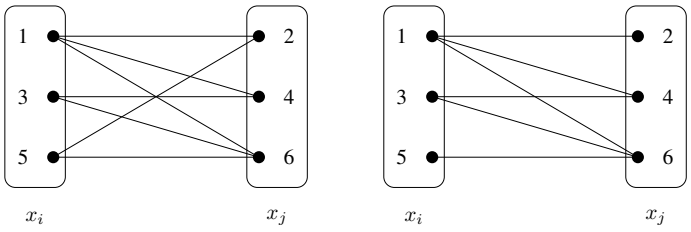

FIGURE 2.17 – *La propriété RRM.*

La figure 2.17(a) n'est pas monotone à droite car la valeur 5 de la variable *i* est compatible avec la valeur 2 de la variable *j* mais elle n'est pas compatible avec la valeur 4 de la même variable.

Pour la figure 2.17(b), 1 est compatible avec 2, 4 et 6, 3 sont compatibles avec 4 et 6 et enfin 5 est compatible avec 6. Par conséquent, la relation est monotone à droite.

2.6.5 Classes polynomiales hybrides

Nous finissons cette section avec les classes polynomiales dites *hybrides*. En fait, les classes hybrides sont les classes les plus récentes.

Définition 2.66 (classes polynomiales hybrides)**.** *Les classes polynomiales hybrides captent à la fois des CSP appartenant à des classes polynomiales structurelles et des CSP appartenant à des classes polynomiales relationnelles.*

Le terme hybride pour les classes polynomiales a été introduit par Cohen dans [Cohen, 2003] puis repris par Cooper et al. dans [Cooper et al., 2010]. D'ailleurs c'est dans [Cooper et al., 2010] que la propriété BTP (pour **B**roken **T**riangle **P**roperty) a été introduite.

2.6.5.1 La propriété des triangles cassés (*BTP*)

La propriété BTP a été introduite dans [Cooper et al., 2010] pour les CSP binaires et a défini un pont vers certaines autres classes polynomiales. La classe polynomiale correspondante aux instances satisfaisant BTP capte à la fois des classes polynomiales *structurelles* et des classes polynomiales *relationnelles*. Formellement, BTP est définie comme suit :

Définition 2.67 (Broken-Triangle Property [Cooper et al., 2010])**.** *Un CSP binaire P satisfait la **Broken Triangle Property (BTP)** par rapport à un ordre sur les variables $<$ si, pour tout triplet de variables (x_i, x_j, x_k) tel que $x_i < x_j < x_k$, si $(v_i, v_j) \in R(c_{ij})$, $(v_i, v'_k) \in R(c_{ik})$ et $(v_j, v''_k) \in R(c_{jk})$, alors soit $(v_i, v''_k) \in R(c_{ik})$, soit $(v_j, v'_k) \in R(c_{jk})$.*

Cette définition peut se représenter graphiquement par la microstructure du CSP comme indiqué dans la figure 2.18.

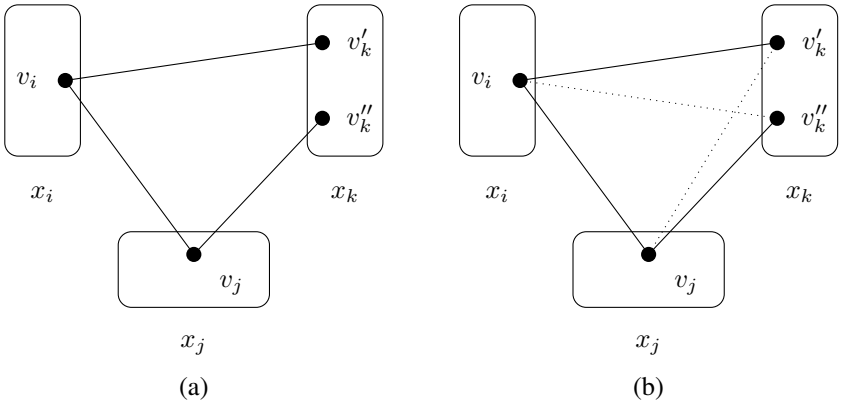

FIGURE 2.18 – (a) un motif non-BTP, (b) un motif BTP.

Pour la figure 2.18 (a), ce motif n'est pas BTP car il n'existe pas d'arête entre $\{v_j, v'_k\}$ ni entre $\{v_i, v''_k\}$ par rapport l'ordre $x_i < x_j < x_k$. Pour la figure 2.18 (b), ce motif est BTP car nous avons une arête entre $\{v_j, v'_k\}$ et entre $\{v_i, v''_k\}$ (une de ces deux arête suffira pour que ce motif soit BTP). Étant donnée une instance CSP binaire P, tester l'existence d'un ordre pour BTP peut se faire en temps polynomial.

Théorème 2.7. *Le problème de trouver un ordre sur les variables pour lequel une instance CSP binaire satisfait BTP (ou déterminer qu'un tel ordre ne peut exister) peut être résolu en temps polynomial (en $O(ned^3)$)*

Cooper et al. ont prouvé, que les CSP satisfaisant BTP, définissent une nouvelle classe polynomiale. Ces CSP peuvent être reconnus (en O(ned^3)) et résolus (en O(ed^2)) en temps polynomial.

Un des avantages de BTP est que cette propriété est préservée même après suppression de valeurs par application de filtrage par cohérence.

Définition 2.68. *Une classe $\mathcal{C}$ d'instances de CSP est dite **close** par rapport à un filtrage de cohérence ϕ si elle est fermée pour ϕ, c'est-à-dire, si le problème obtenu après l'application de ϕ à toute instance de $\mathcal{C}$ appartient à la classe $\mathcal{C}$. Une propriété est dite **close** si elle définit une classe d'instances closes.*

La propriété close a été définie pour expliquer comment BTP sera préservée même après réduction des domaines par application de filtrage. Ceci a permis de montrer des résultats intéressants sur la résolution des instances BTP par MAC en temps polynomial.

La propriété BTP est tellement forte que les auteurs ont pensé à l'alléger. Cette nouvelle version est dite min-of-max extendable et elle est définie par rapport à un ordre sur les valeurs.

Définition 2.69 (min-of-max extendable [Cooper et al., 2010])**.** *Une instance CSP binaire P satisfait la **Min-of-Max Extendable (MME)** par rapport à un ordre sur les variables $<$ si, pour tout triplet de variables (x_i, x_j, x_k) tel que $x_i < x_j < x_k$, si $(v_i, v_j) \in R(c_{ij})$ alors ,(v_i, v_j, v_k) est une affectation cohérente, avec $v_k = min(max(R(c_{ik})[v_i], max(R(c_{jk})[v_j])$.*

Par symétrie, nous pouvons définir max-of-min extendable avec
$v_k = max(min(R(c_{ik})[v_i], min(R(c_{jk})[v_j])$. Le lemme suivant définit la relation entre BTP et MME.

Lemme 2.1. *Une instance CSP binaire satisfait BTP par rapport à un ordre $<$ sur les variables, si et seulement si, il est MME par rapport à $<$ en considérant tout ordre possible sur les valeurs.*

Contrairement à BTP, la propriété MME est non close. Par exemple, pour une instance CSP satisfaisant MME, on peut perdre la propriété en appliquant la cohérence d'arc.

La classe des instances BTP (et MME) capte certaines autres classes polynomiales comme celle des instances binaires acycliques (TREE) et la classe renommable monotone à droite (RRM). Elle est aussi en relation avec certaines autres classes comme convexe par rangée.

Proposition 2.4. $RRM \subsetneq BTP$, $TREE \subsetneq BTP$.

Proposition 2.5. *Si une instance CSP binaire est convexe par rangée et est directionnel chemin-cohérente, alors elle est min-of-max entendable (et max-of-min extendable).*

2.7 Conclusion

A travers ce chapitre, nous avons introduit le cadre général de la thèse portant sur les classes polynomiales pour le problème CSP. Nous avons commencé par énoncer les définitions et les notations qui entourent le problème CSP. Nous avons aussi illustré les méthodes permettant de réduire les domaines. Ensuite, nous avons parlé des algorithmes de résolutions à savoir l'algorithme Backtracking et les algorithmes utilisant le filtrage avant ou pendant la recherche ainsi que les heuristiques utilisées pour guider et accélérer la recherche d'une solution.

Par la suite, nous avons rappelé certaines classes polynomiales identifiées par certaines restrictions sur les domaines, les contraintes ou les relations d'une instance CSP.

Nous avons montré aussi que les travaux théoriques sur les classes polynomiales sont assez développés. Mais, la plupart de ces travaux n'ont pas montré, à ce jour, l'intérêt pratique de ces

classes sauf la décomposition arborescente qui a été exploitée par de nombreux travaux pour les réseaux ayant une petite largeur arborescente. Sinon, les classes polynomiales n'ont jamais été exploitées, au moins d'une façon directe malgré l'efficacité des solveurs CSP en pratique.

Dans le prochain chapitre, nous proposerons des définitions de microstructure pour les CSP non-binaires afin d'étendre certaines classes polynomiales définies pour le cas restreint des CSP binaires aux CSP d'arité quelconques.

Deuxième partie

Microstructures des CSP n-aires et extension des classes polynomiales

Chapitre 3

Microstructures pour CSP n-aires

3.1 Introduction

Comme nous l'avons indiqué dans l'état de l'art, plusieurs travaux ont mis en évidence l'intérêt que recèle l'étude des *microstructures* pour définir de nouvelles *classes polynomiales* qui sont parfois fondées sur la théorie (algorithmique) des graphes [Jégou, 1993a, Cohen, 2003, Salamon and Jeavons, 2008]. En effet, il a été montré, pour le cas d'un CSP binaire, que si le graphe de microstructure est triangulé ou parfait ou si le complément de la microstructure est triangulé alors le CSP peut être résolu en temps polynomial. Il est à rappeler que la première microstructure pour CSP n-aire a été proposée par Cohen dans [Cohen, 2003] et qui est basée sur le complément d'un hypergraphe. Cette microstructure semble ne pas avoir été exploitée en théorie et en pratique vu que le recours au complément d'un hypergraphe pose manifestement des problèmes d'ordre combinatoire comme nous l'avons expliqué dans le chapitre précédent.

Dans ce chapitre, nous étudierons des microstructures pour les CSP d'arité quelconque pour vérifier si certains résultats sur les classes polynomiales pour les CSP binaires peuvent être étendus aux CSP d'arité quelconque. Pour le faire, nous proposerons une approche différente de celle de [Cohen, 2003], puisqu'elle s'appuie tout simplement sur la notion de graphe. A cette fin, nous présenterons ici trois types possibles de microstructures basées respectivement sur les différents codages binaires des CSP non-binaires : la représentation duale [Dechter and Pearl, 1987b], la transformation dite par variables cachées [Rossi et al., 1990] et le codage mixte [Stergiou and Walsh, 1999]. Nous étudierons les propriétés de base de ces représentations, en suggérant différentes pistes pour leur exploitation théorique, notamment en vue de l'extension de classes polynomiales au cas des CSP d'arité quelconque. Une partie de ce travail a été publiée dans [El Mouelhi et al., 2013b].

3.2 Microstructures basées sur les codages binaires des CSP non-binaires

Dans cette section, nous présentons des microstructures pour les CSP d'arité quelconque en se basant sur certaines transformations, appelées *codages* permettant la conversion d'un CSP non-binaire en un CSP binaire en préservant l'ensemble de solutions du CSP non-binaire. Il s'agit du codage dual [Dechter and Pearl, 1987b], du codage par variables cachées [Rossi et al., 1990, Bacchus and van Beek, 1998] et du codage mixte [Stergiou and Walsh, 1999].

3.2.1 Microstructure basée sur le codage dual

Le codage dual, dans le domaine des CSP, a été employé pour la première fois dans [Dechter and Pearl, 1987b]. L'idée de ce codage est inspirée ce qui est connu dans la théorie des (hyper)graphes sous le vocable *Line Graph* ou aussi *Qual Graphs* [Bernstein and Goodman, 1981] dans la théorie des bases de données relationnelles. Dans la communauté CSP, il est appelé *Graphe Dual* et aussi *Intergraphe* dans [Jégou, 1991, 1993b] et il est basé sur la transformation d'un hypergraphe en graphe. Dans le codage dual, les variables correspondent aux contraintes du problème originel et elles sont généralement dites *variables duales*. Les nouvelles contraintes binaires relient deux variables duales si elles partagent au moins une variable, c'est-à-dire, si l'intersection des portées des contraintes originelles correspondantes n'est pas vide. Le CSP dual correspondant à ce codage binaire est défini comme suit :

Définition 3.1 (CSP Dual)**.** *Le CSP* ***dual*** *du CSP $P = (X, D, C)$ est le CSP binaire $P^d = (X^d, D^d, C^d)$ où chaque contrainte c_i de C correspond à la variable x_i^d de X^d dont le domaine $D(x_i^d)$ est défini par l'ensemble des tuples t_i de $R(c_i)$, et une contrainte c_{ij}^d de C^d relie deux variables x_i^d et x_j^d de X^d si les contraintes associées c_i et c_j de C partagent au moins une variable. La relation $R(c_{ij}^d)$ est alors définie par les tuples $(t_i, t_j) \in D(x_i^d) \times D(x_j^d)$ tels que $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$.*

Pour des raisons de simplicité, nous utiliserons, dans le reste de ce manuscrit, la contrainte originelle c_i pour désigner la variable duale x_i^d correspondante dans P^d. Cette transformation permet de définir une représentation binaire équivalente au problème non-binaire de départ, au sens où il existe une bijection entre les ensembles de solutions du CSP de départ et de celui de la représentation duale [Jégou, 1991]. Graphiquement, les sommets du graphe dual sont les contraintes originelles et les arêtes correspondent à l'intersection non vide de leurs portées. Formellement, la représentation duale d'un CSP est donnée par le graphe défini comme suit :

Définition 3.2 (graphe dual)**.** *Étant donné un CSP $P = (X, D, C)$, le graphe* ***dual*** *de P est un graphe $D(P) = (V, E)$ avec :*
- $V = \{c_i \in C\}$,
- $E = \{\{c_i, c_j\} \mid i \neq j$ *et* $S(c_i) \cap S(c_j) \neq \emptyset\}$

En remplaçant c_i par x_i^d et c_j par x_j^d, nous pouvons remarquer que le graphe dual correspond au graphe de contraintes du CSP P^d. La figure 3.1 représente le graphe dual de l'exemple 2.1 avec les arêtes étiquetées par les intersections des portées de contraintes.

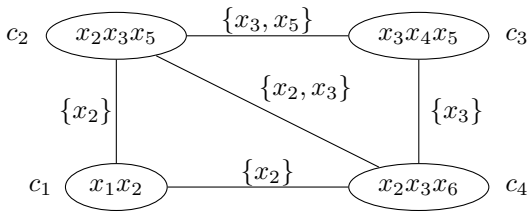

FIGURE 3.1 – Le graphe dual de l'exemple 2.1.

La définition de la microstructure associée à ce codage, notée *DR-microstructure*, correspond à la microstructure de ce CSP binaire équivalent :

Définition 3.3 (DR-microstructure)**.** *Étant donné un CSP $P = (X, D, C)$, la* microstructure basée sur la représentation duale *de P, dite* ***DR-microstructure****, est un graphe non-orienté $\mu_{DR}(P) = (V, E)$ avec :*

- $V = \{(c_i, t_i) : c_i \in C, t_i \in R(c_i)\}$,
- $E = \{\ \{(c_i, t_i), (c_j, t_j)\}\ \mid\ (c_i, t_i) \in V$ *et* $(c_j, t_j) \in V$ *avec* $i \neq j, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]\}$

Nous pouvons remarquer que dans le cas où $S(c_i) \cap S(c_j) = \emptyset$, chaque tuple de la relation $R(c_i)$ sera connecté à tous les tuples de la relation $R(c_j)$ (c'est-à-dire, $\forall t_i \in R(c_i)$ et $\forall t_j \in R(c_j)$, on a $\{(c_i, t_i), (c_j, t_j)\} \in E$.

Comme pour la microstructure des CSP binaires, il existe un lien direct entre les affectations cohérentes et les cliques et évidemment entre les solutions et les cliques maximales.

Théorème 3.1. *Un CSP P a une solution, si et seulement si, $\mu_{DR}(P)$ a une clique de taille e.*

Preuve : Par construction, $\mu_{DR}(P)$ est un graphe e-parti, et toute clique contient au plus un sommet (c_i, t_i) pour chaque contrainte $c_i \in C$. Donc, une e-clique de $\mu_{DR}(P)$ correspond exactement à une clique avec un seul sommet (c_i, t_i) de chaque contrainte $c_i \in C$. De plus, chaque couple de sommets (c_i, t_i) et (c_j, t_j) reliés par une arête satisfait $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$. Donc, tous les sommets (c_i, t_i) d'une clique sont deux à deux adjacents et donc compatibles. Aussi, une e-clique de $\mu_{DR}(P)$ correspond exactement à e tuples t_i autorisés par toutes les contraintes, ce qui est équivalent à une solution de P. □

La DR-microstructure de l'exemple 2.1 est présentée par la figure 3.2. Nous avons 4 contraintes, donc $e = 4$. Conformément au théorème 3.1, une solution de P est une clique de taille 4, ce qui est le cas de $\{ab, bce, bcf, cde\}$. Dans les exemples, nous noterons directement par t_i le sommet (c_i, t_i).

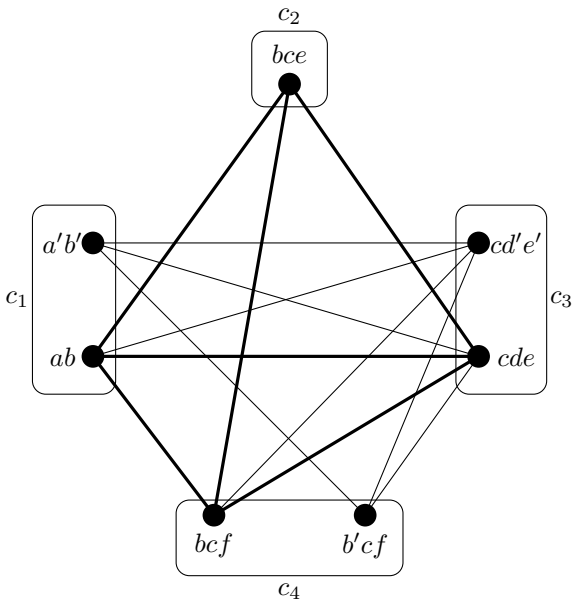

FIGURE 3.2 – La microstructure basée sur le codage dual de l'exemple 2.1.

En supposant que les relations des CSP sont données sous forme de tables, la taille de la DR-microstructure est bornée par un polynôme fonction de la taille du CSP, c'est-à-dire $|E| \leq |V|^2$ avec $|V| = \Sigma_{c_i \in C}|R(c_i)|$. De plus, il est clair que le calcul de la DR-microstructure, pour un CSP dont les relations sont exprimées en extension, peut se réaliser en temps polynomial (O($er + e^2r^2$) avec r le nombre de tuples maximal de toutes les contraintes) :

– er pour construire tous les sommets,
– e^2r^2 pour construire toutes les arêtes.

Nous supposons que l'ajout d'un sommet ou d'une arête dans la DR-microstructure peut se faire en temps constant (O(1)). Pour un CSP dont les relations sont données en intension, il faut prévoir une complexité plus élevée (typiquement exponentielle en l'arité) puisqu'il faudrait calculer tous les tuples autorisés de chaque relation avant la construction des sommets correspondants.

Étant donné un CSP P et son graphe dual G, il est bien connu que dans G certaines arêtes redondantes peuvent être éliminées sans toucher à l'équivalence entre l'ensemble des solutions du problème de départ et l'ensemble des solution du nouveau CSP correspondant au graphe dual modifié [Janssen et al., 1989, Jégou, 1991]. Effectivement, une arête peut être supprimée du graphe dual si elle figure dans un cycle et si l'ensemble de variables originelles qui l'étiquette apparaît dans l'étiquette de chaque arête de ce cycle. Certains des nouveaux graphes obtenus après suppression des arêtes redondantes sont minimaux pour l'inclusion et pour le nombre des arêtes restantes. Ces graphes seront appelés *qual subgraphs* ou *intergraphes* alors que les minimaux seront appelés *minimal gual graphs* ou *intergraphes minimaux*.

L'exemple de la figure 3.1 a deux intergraphes minimaux (voir figure 3.3). Nous pouvons facilement remarquer que dans le graphe dual, les contraintes formant le cycle (c_1, c_2, c_4, c_1) partagent la variable x_2, donc elle pourrait être supprimée. Deux possibilités se présentent : soit supprimer l'arête qui relie c_1 à c_2, soit supprimer l'arête qui relie c_1 à c_4. Pareillement pour le cycle (c_2, c_3, c_4, c_2) et la variable x_3. Donc, ceci nous mène aux deux intergraphes minimaux de la figure 3.3.

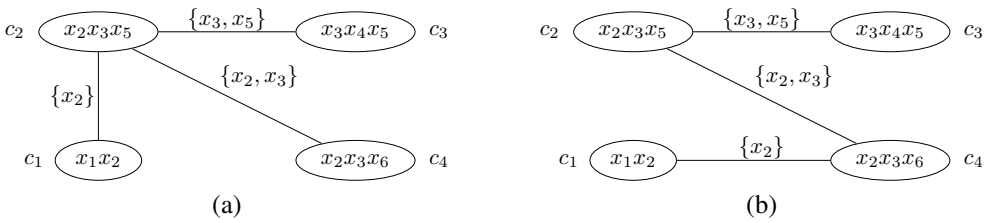

FIGURE 3.3 – Les deux intergraphes minimaux de l'exemple 2.1.

L'exemple précédent possède certains intergraphes non-minimaux. Dans la figure 3.4, nous en illustrons deux.

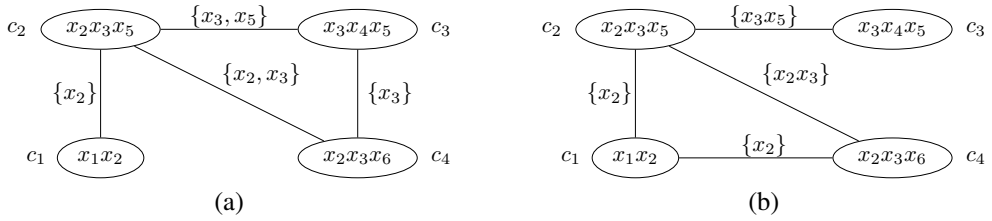

FIGURE 3.4 – Deux intergraphes non-minimaux de l'exemple 2.1.

Dans [Jégou, 1991], il est montré que pour un CSP d'arité quelconque, il existe un ensemble des CSP binaires équivalents construits sur la base de l'ensemble des intergraphes. Sur cette base, nous pouvons ainsi définir un ensemble de microstructures différentes, voisines de la DR-microstructure.

En considérant cet ensemble de graphes partiels, nous pouvons étendre la définition précédente de DR-microstructure et qui sera notée DSR-microstructure (pour **D**ual **S**ubgraph **R**epresentation) :

Définition 3.4 (DSR-microstructure)**.** *Étant donnés un CSP $P = (X, D, C)$ (d'arité quelconque) et un de ses intergraphes (C, F), la* Microstructure basée sur la représentation des graphes partiels duaux *de P est un graphe non-orienté $\mu_{DSR}(P, (C, F)) = (V, E)$ avec :*

- $V = \{(c_i, t_i) : c_i \in C, t_i \in R(c_i)\}$,
- $E = E_1 \cup E_2$ *tels que*
  - $E_1 = \{\ \{(c_i, t_i), (c_j, t_j)\}\ \mid\ \{c_i, c_j\} \in F, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]\}$
  - $E_2 = \{\ \{(c_i, t_i), (c_j, t_j)\}\ \mid\ \{c_i, c_j\} \notin F\}$.

La figure 3.5 présente un des intergraphes minimaux de l'exemple 2.2 illustré précédemment (a) et la microstructure correspondante (b). En gras, les arêtes qui ont été rajoutées par rapport à la DR-microstructure.

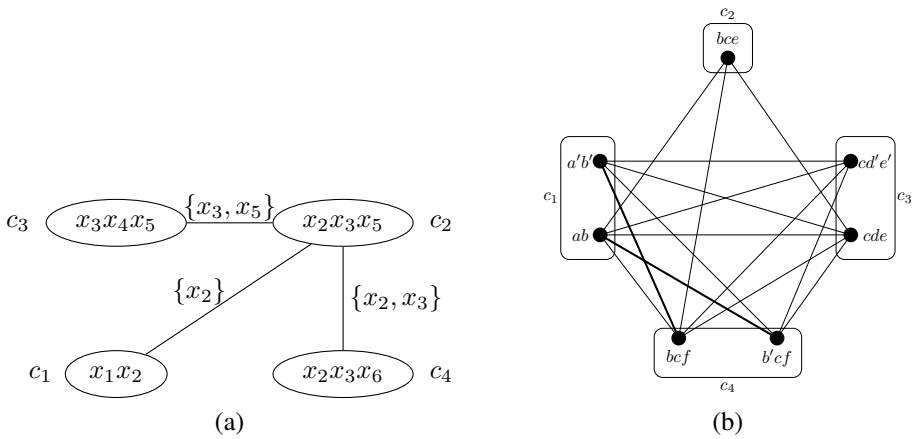

FIGURE 3.5 – L'intergraphe minimal (a) et la microstructure correspondante (b).

Avec cette représentation, nous disposons des mêmes propriétés en termes de taille de la DSR-microstructure puisqu'elle demeure bornée par le même polynôme que la DR-microstructure, et le calcul de cette microstructure sera également réalisable en temps polynomial. En fait, on aura la même complexité que pour le calcul de la DR-microstructure à laquelle s'ajoute le coût du calcul du nouvel intergraphe.

Nous pouvons facilement constater que les deux microstructures DR-microstructure et DSR-microstructure ont le même ensemble de sommets. Pour les arêtes, il est bien clair que l'ensemble des arêtes de la DR-microstructure est inclus dans l'ensemble des arêtes de la DSR-microstructure. En effet, toute arête du graphe dual supprimée sera remplacé par une contrainte universelle dans la microstructure correspondant au nouvel intergraphe. Ceci nous conduit au théorème suivant :

Théorème 3.2. *Étant donné un CSP P, quelque soit (C, F) intergraphe de P, $\mu_{DR}(P)$ est un graphe partiel de $\mu_{DSR}(P, (C, F))$.*

Preuve : Il est clair que toute arête de $\mu_{DR}(P)$ est aussi une arête de $\mu_{DSR}(P, (C, F))$ car :
(1) pour les arêtes qui existent dans le graphe dual et qui n'ont pas été supprimées dans le graphe partiel dual, nous conservons le même ensemble d'arêtes dans $\mu_{DSR}(P, (C, F))$ que celui de $\mu_{DR}(P)$.
(2) pour les arêtes qui existent dans le graphe dual et qui ont été supprimées dans le graphe partiel dual, nous considérons une relation universelle dans $\mu_{DSR}(P, (C, F))$ qui contient nécessairement

les arêtes de la $\mu_{DR}(P)$. □

Nous pouvons également déduire que le résultat sur les cliques est toujours vrai :

Théorème 3.3. *Un CSP P possède une solution, si et seulement si, pour tout intergraphe (C, F) de P, $\mu_{DSR}(P, (C, F))$ possède une clique de taille e.*

Preuve : ($\Rightarrow$)
Soit $(v_1, v_2, ..., v_n)$ une solution de P telle que les e contraintes sont satisfaites. Alors, il existe e tuples $(t_1, t_2, ..., t_e)$ de $(c_1, c_2, ..., c_e)$ qui correspondent à $(v_1, v_2, ..., v_n)$. Donc, $\forall i, j$ tels que $i \neq j$, on a : $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$. Si $\{S(c_i), S(c_j)\}$ est une arête dans l'intergraphe alors $\{(c_i, t_i), (c_j, t_j)\}$ est une arête dans la DSR-microstructure, sinon (si $\{S(c_i), S(c_j)\}$ n'est pas une arête dans l'intergraphe, alors on aura une arête aussi $\{(c_i, t_i), (c_j, t_j)\}$ dans la DSR-microstructure car dans ce cas tous les tuples de $S(c_i)$ sont compatibles avec tous les tuples de $S(c_j)$.
($\Leftarrow$)
Étant donnée une e-clique $Cl = \{(c_1, t_1), ..., (c_e, t_e)\}$ de la DSR-microstructure. Comme il n'y a pas d'arête dans la DSR-microstructure de la forme $\{(c_i, t_j), (c_i, t'_j)\}$ (avec $t_j \neq t'_j$), alors on a toujours un seul tuple par contrainte qui participe dans Cl. Et comme toute variable x est dans la portée d'une contrainte, alors on a une affectation au moins de chaque variable. Donc, il reste à montrer qu'une variable n'a qu'une affectation. Soit une variable x qui appartient à la fois à $S(c_i)$ et à $S(c_j)$ (avec $i \neq j$) et telle que $t_i[\{x\}] \neq t_j[\{x\}]$ (a), on a donc deux cas de figures :

- l'arête $\{S(c_i), S(c_j)\}$ appartient à l'intergraphe associé à la DSR-microstructure, donc l'arête $\{(c_i, t_i), (c_j, t_j)\}$ est dans la DSR-microstructure puisque $(t_1, ..., t_i, ..., t_j, ..., t_e)$ est une clique, d'après (a) ceci est impossible.
- l'arête $\{S(c_i), S(c_j)\}$ n'appartient pas à l'intergraphe associé à la DSR-microstructure. Puisque $x \in S(c_i) \cap S(c_j)$, il existe un chemin dans l'intergraphe $c_i = c_1, c_2, ..., c_k = c_j$ tel que $S(c_i) \cap S(c_j) \subseteq S(c_\alpha) \cap S(c_{\alpha+1})$ pour tout $1 \leq \alpha < k$. Ceci implique que $x \in S(c_1), S(c_2), ..., S(c_k)$. Soit $t_1, t_2, ..., t_k$ les k tuples de la clique, on a $t_\alpha[S(c_\alpha) \cap S(c_{\alpha+1})] = t_{\alpha+1}[S(c_\alpha) \cap S(c_{\alpha+1})]$. Donc $t_\alpha[\{x\}] = t_{\alpha+1}[\{x\}]$, d'après (a) ceci est aussi impossible.

Donc, toute e-clique dans la DSR-microstructure correspond à une solution de CSP. □

3.2.2 Microstructure basée sur le codage par variables cachées

Le deuxième codage est basé sur la notion de variables cachées (*hidden variable en anglais*) ou aussi factor graphs [Kschischang et al., 1998]. Il est inspiré par Peirce [Peirce et al., 1933] (cité dans [Rossi et al., 1990]). Dans cette transformation, l'ensemble de variables contient les variables originelles de X plus l'ensemble des variables cachées C. Les nouvelles contraintes binaires vont relier une contrainte originelle (variable duale) à une variable originelle si la variable originelle appartient à la portée de la contrainte originelle. Le CSP binaire associé à ce codage est défini comme suit :

Définition 3.5 (CSP Caché)**.** *Le CSP **caché (Hidden CSP)** du CSP $P = (X, D, C)$ est le CSP binaire $P^h = (X^h, D^h, C^h)$ avec :*

- $X^h = X_1^h \cup X_2^h$ *:*
  - $X_1^h = \{x_j \in X\}$
  - $X_2^h = \{c_i \in C\}$
- $D^h = D \cup \{R(c_i) \mid c_i \in C\}$,
- $C^h = \{c_i^h \mid S(c_i^h) = \{x_k^h, x_j^h\}$ *avec* $x_k^h = x_k$ *et* $x_j^h = c_j$ *telles que* $x_k \in S(c_j)$ *et* $R(c_i^h) = \{(t_j, v_k) \mid t_k \in R(c_j), v_k \in D(x_k)$ *et* $t_j[x_k] = v_k\}$.

Dans la suite, nous utiliserons la variable originelle x_j pour désigner la variable x_j^h appartenant à X_1^h et la contrainte originelle c_i pour désigner la variable cachée x_i^h appartenant à X_2^h.

Le graphe correspondant à ce codage est un graphe biparti dont le premier stable contient les variables originelles et le deuxième stable contient les variables cachées. Nous connectons une contrainte c_i à toute variable appartenant à sa portée $S(c_i)$.

Définition 3.6 (représentation par variables cachées)**.** *Étant donné un CSP $P = (X, D, C)$ d'arité quelconque, la* ***représentation par variables cachées*** *de P est un graphe non-orienté $VC(P) = (V, E)$ avec :*
- $V = V_1 \cup V_2$ *tels que :*
  - $V_1 = \{x_j \in X\}$
  - $V_2 = \{c_i \in C\}$,
- $E = \{\{c_i, x_j\} \mid c_i \in C$ *et* $x_j \in X$ *telle que* $x_j \in S(c_i)\}$.

La figure 3.6 montre la représentation par variables cachées de l'exemple 2.1.

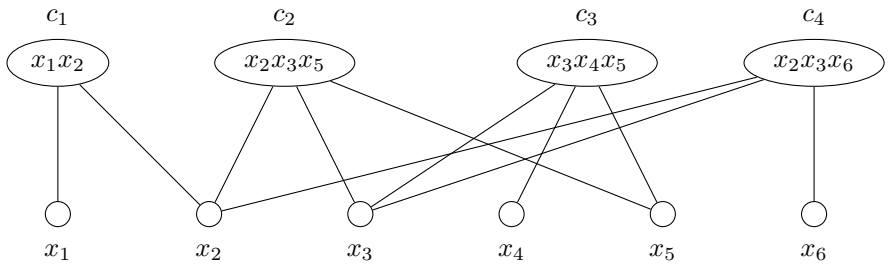

FIGURE 3.6 – Le graphe de codage par variables cachées de l'exemple 2.1.

Dans la littérature, nous trouvons d'autres appellations de ce graphe comme graphe de contraintes biparti [Jégou, 1991]. Ce codage est clairement différent du précédent vu qu'il s'appuie sur les graphes bipartis.

La microstructure sera notée HT-microstructure (pour **H**idden **T**ransformation) :

Définition 3.7 (HT-microstructure)**.** *Étant donné un CSP $P = (X, D, C)$ (d'arité quelconque), la* microstructure cachée *de P est un graphe non-orienté $\mu_{HT}(P) = (V, E)$ avec :*
- $V = S_1 \cup S_2$ *tels que :*
  - $S_1 = \{(x_i, v_i) : x_i \in X, v_i \in D(x_i)\}$,
  - $S_2 = \{(c_i, t_i) : c_i \in C, t_i \in R(c_i)\}$,
- $E = \{\ \{(c_i, t_i), (x_j, v_j)\}\ \mid$ *soit* $x_j \in S(c_i)$ *et* $v_j = t_i[x_j]$, *soit* $x_j \notin S(c_i)\}$.

Contrairement à la DR-microstructure, la HT-microstructure ne correspond pas à la microstructure du graphe de contraintes biparti à cause des contraintes universelles imposées quand $x_j \notin S(c_i)$. La HT-microstructure de l'exemple 2.1 est donnée par la figure 3.7.

Nous pouvons constater que la HT-microstructure est aussi un graphe biparti (tout comme le graphe de contraintes biparti) car nous avons d'un côté les valeurs des domaines, et de l'autre, les tuples des relations. Avant de parler d'une solution dans la HT-microstructure, nous devons rappeler la définition suivante :

Définition 3.8. *Une biclique $K_{a,b}$ est un sous-graphe biparti complet avec a sommets pour une partie et b sommets pour l'autre.*

Dans la HT-microstructure, une solution correspond à une biclique un peu particulière comme le montrera le théorème 3.4 qui se déduit directement des deux lemmes suivants.

Lemme 3.1. *Dans une HT-microstructure, une biclique $K_{n,e}$ avec e tuples (avec $e > 0$) appartenant à des relations deux à deux différentes ne peut pas contenir deux valeurs différentes d'un même domaine.*

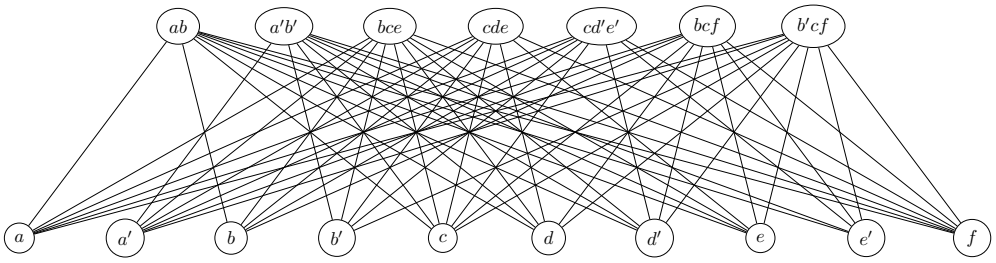

FIGURE 3.7 – Microstructure cachée de l'exemple 2.1.

Preuve : Supposons qu'une biclique $K_{n,e}$ avec e tuples appartenant à des relations deux à deux différentes contienne deux valeurs différentes v_j et $v'_j \in D(x_j)$. Il existe alors au moins une contrainte c_i telle que $x_j \in S(c_i)$ (car une variable doit appartenir au moins à la portée d'une contrainte). Soit $t_i \in R(c_i)$ le tuple de c_i dans la biclique $K_{n,e}$. Donc,

- soit $t_i[x_j] = v_j$
- soit $t_i[x_j] = v'_j$
- soit $t_i[x_j] = v''_j$ avec $v''_j \neq v_j$ et $v''_j \neq v'_j$.

Dans les trois cas, t_i sera lié seulement à une seule valeur de ces trois. Or, ceci contredit que v_j et v'_j sont dans une biclique car il est impossible d'avoir deux tuples d'une même relation. □

Lemme 3.2. *Dans une HT-microstructure, une biclique $K_{n,e}$ avec n valeurs issues de domaines différents ne peut pas contenir deux tuples issus d'une même relation.*

Preuve : Supposons qu'une biclique $K_{n,e}$ avec n valeurs appartenant à des variables deux à deux différentes contienne deux tuples t_i et t'_i d'une relation d'une même contrainte c_i. Donc, il existe au moins une variable x_j telle que $t_i[x_j] \neq t'_i[x_j]$. Si $v_j = t_i[x_j]$ et $v'_j = t'_i[x_j]$ appartiennent toutes les deux à la biclique $K_{n,e}$, on a une contradiction car nous ne pouvons pas avoir deux valeurs d'une même variable. □

En utilisant ces deux lemmes, nous pouvons déduire que toute biclique $K_{n,e}$ avec n valeurs et e tuples tels que chaque couple de valeurs appartient à un couple de variables différentes et chaque couple de tuples appartient à un couple de relations différentes, correspond à une affectation de toutes les variables qui satisfait toutes les contraintes. Nous pouvons dans ce cas énoncer le théorème suivant :

Théorème 3.4. *Étant donnés un CSP $P = (X, D, C)$ et sa HT-microstructure $\mu_{HT}(P)$, P possède une solution ssi $\mu_{HT}(P)$ a une biclique $K_{n,e}$ avec n valeurs et e tuples tels que tous les tuples appartiennent à des relations deux à deux différentes et toutes les valeurs appartiennent à des domaines deux à deux différents.*

En revenant à l'exemple précédent, nous pouvons observer qu'une biclique ne correspond pas forcément à une solution. Bien que $\{b, c, d, d', e, f, ab, a'b', bce, bcf\}$ et $\{d, d', c, e, e', f, ab, a'b', bcf, b'cf\}$ soient des bicliques $K_{6,4}$, elles ne constituent pas pour autant des solutions. Par contre, $\{a, b, c, d, e, f, ab, bce, bcf, cde\}$ est une biclique $K_{6,4}$ et est aussi une solution de P.

Donc, l'ensemble de solutions n'est pas équivalent à l'ensemble des bicliques $K_{n,e}$. Cet ensemble est équivalent seulement à l'ensemble des bicliques $K_{n,e}$ avec n valeurs et e tuples telles qu'aucun couple de valeurs (resp. de tuples) n'appartient à un même domaine (resp. à une même relation).

Il faut signaler aussi que, dans la HT-microstructure, nous pouvons trouver une biclique $K_{n',e'}$ avec $n' \geq n$ et $e' \geq e$. L'exemple de la figure 3.8 contient une biclique $K_{6,3}$ $(a, a', a'', b, b', b'', cd, c'd', c''d'')$ alors que $n = 4$ et $e = 2$.

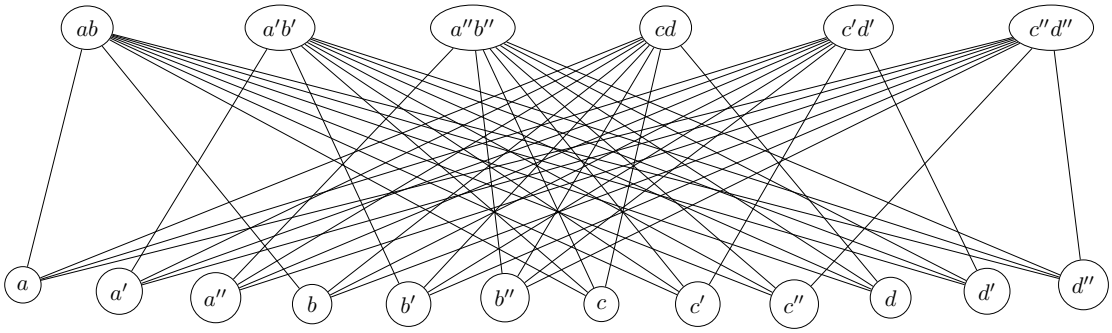

FIGURE 3.8 – Une microstructure cachée ayant une biclique $\{ab, a'b', a''b'', a, a', a'', b, b', b''\}$ de taille supérieure à n et e.

Comme pour la DR-microstructure, la taille de la HT-microstructure est bornée polynomialement par la taille du CSP :
- $|V| = \Sigma_{x_i \in X} |D(x_i)| + \Sigma_{c_i \in C} |R(c_i)|$ et
- $|E| \leq \Sigma_{x_i \in X} |D(x_i)| \times \Sigma_{c_i \in C} |R(c_i)|$.

De plus, étant donné un CSP, calculer sa HT-microstructure peut aussi se réaliser en temps polynomial si la taille du CSP est polynomiale. En effet, la complexité de calcul de cette microstructure pour un CSP dont les contraintes sont en extension est O($nd + er + nder$) :
- nd pour calculer les sommets correspondants aux valeurs,
- er pour calculer les sommets associés aux tuples et
- $nder$ pour les arêtes.

Il existe une autre façon de représenter la microstructure à partir du codage caché et qui est lié à une autre manière de compléter la microstructure que nous développerons dans le paragraphe suivant.

3.2.3 Microstructure basée sur le codage mixte

Le dernier graphe est basé sur le codage mixte (également appelé *double* dans [Stergiou and Walsh, 1999, Smith et al., 2000]). Le codage mixte d'un CSP non-binaire, combine à la fois le codage dual et le codage par variables cachées.

Le CSP mixte associé à ce codage binaire est différent de celui de [Fargier et al., 1996] et il est défini de la façon suivante :

Définition 3.9 (CSP mixte)**.** *Le CSP **mixte (mixed CSP)** du CSP $P = (X, D, C)$ est le CSP binaire $P^m = (X^m, D^m, C^m)$ avec :*
- $X^m = X^h$,
- $D^m = D^h$,
- $C^m = C^d \cup C^h \cup C^u$ *avec :*
  - $C^d = \{c^d_{ij} \mid S(c^d_{ij}) = \{x^d_i, x^d_j\}$ *avec* $x^d_i = x_i$ *et* $x^d_j = x_j\}$
  - $C^h = \{c^h_i \mid S(c^h_i) = \{x^h_k, x^h_j\}$ *avec* $x^h_k = x_k$ *et* $x^h_j = c_j$ *telles que* $x_k \in S(c_j)$ *et* $R(c^h_i) = \{(t_j, v_k) \mid t_k \in R(c_j), v_k \in D(x_k)$ *et* $t_j[x_k] = v_k\}$.
  - $C^u = \{c^u_{ij} \mid c^u_{ij} = (S(c^u_{ij}), R(c^u_{ij}))$

– $S(c^u_{ij}) = \{x_i, x_j\}$ *pour* $i \neq j$,
– $R(c^u_{ij}) = \{(v_i, v_j)$ *pour tout* $i \neq j$ *et* $\forall v_i \in D(x_i)$ *et* $v_j \in D(x_j)\}$.

Les relations associées aux contraintes des deux premiers ensembles de contraintes ($C^d \cup C^h$) correspondent respectivement aux relations associées aux contraintes duales et cachées. Les relations associées à C^u sont des relations universelles.

Dans le graphe mixte, les variables duales seront inter-connectées (comme dans le graphe dual) et aussi connectées aux variables originelles (comme dans le codage par variables cachées). Nous conservons les mêmes relations de compatibilité entre les sommets comme dans les représentations précédentes.

Définition 3.10 (graphe mixte)**.** *Étant donné un CSP $P = (X, D, C)$ d'arité quelconque, le codage* ***mixte*** *de P est un graphe non-orienté $GM(P) = (V, E)$ avec :*

– $V = V_1 \cup V_2$ *tels que :*
  • $V_1 = \{x_j \in X\}$
  • $V_2 = \{c_i \in C\}$,
– $E = E_1 \cup E_2$ *tels que :*
  • $E_1 = \{\{c_i, c_j\} \mid i \neq j$ *et* $S(c_i) \cap S(c_j) \neq \emptyset\}$,
  • $E_2 = \{\{c_i, x_j\} \mid c_i \in C$ *et* $x_j \in X$ *telles que* $x_j \in S(c_i)\}$.

La figure 3.11 est le graphe mixte de l'exemple 2.1.

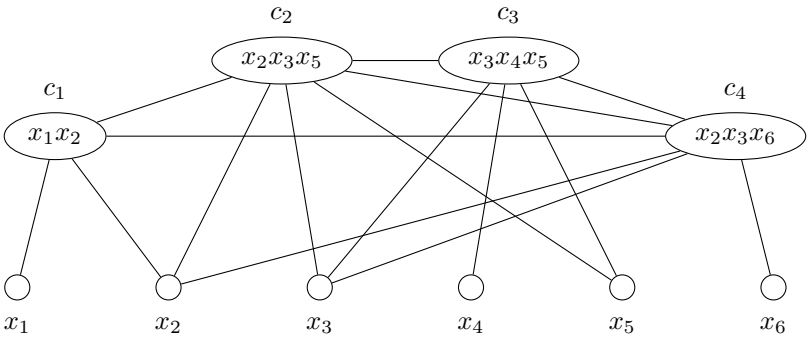

FIGURE 3.9 – Le graphe mixte de l'exemple 2.1.

Dans la microstructure associée au codage mixte, nous préservons aussi le même ensemble de sommets que celui de la HT-microstructure.

Pour les arêtes, nous en avons de trois types :

– des arêtes qui relient deux sommets correspondant aux valeurs originelles : deux sommets sont connectés si leurs valeurs originelles n'appartiennent pas au même domaine.
– des arêtes qui relient deux sommets correspondant aux tuples originels ($t_i \in R(c_i)$ et $t_j \in R(c_j)$) : ces deux sommets sont connectés s'ils satisfont la condition $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]\}$.
– des arêtes qui relient deux sommets dont le premier correspond à une valeur originelle $v_j \in D(x_j)$ et le deuxième correspond à un tuple originel $t_i \in R(c_i)$: les deux sommets seront connectés s'ils satisfont la condition $x_j \in S(c_i)$ et $v_j = t_i[x_j]$ ou bien $x_j \notin S(c_i)\}$.

La définition de la microstructure associée à ce codage (ME-microstructure pour **M**ixed **E**ncoding) est la suivante :

Définition 3.11 (ME-microstructure)**.** *Étant donné un CSP* $P = (X, D, C)$*, la* Microstructure basée sur le codage mixte *de* P *est un graphe non-orienté* $\mu_{ME}(P) = (V, E)$ *avec :*

- $V = S_1 \cup S_2$ *tels que*
  - $S_1 = \{(c_i, t_i) : c_i \in C, t_i \in R(c_i)\}$,
  - $S_2 = \{(x_j, v_j) : x_j \in X, v_j \in D(x_j)\}$,
- $E = E_1 \cup E_2 \cup E_3$ *tels que*
  - $E_1 = \{\ \{(c_i, t_i), (c_j, t_j)\}\ \mid\ i \neq j, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]\}$
  - $E_2 = \{\ \{(c_i, t_i), (x_j, v_j)\}\ \mid$ *soit* $x_j \in S(c_i)$ *et* $v_j = t_i[x_j]$ *soit* $x_j \notin S(c_i)\}$
  - $E_3 = \{\ \{(x_i, v_i),\ (x_j, v_j)\} \mid\ x_i \neq x_j\}$.

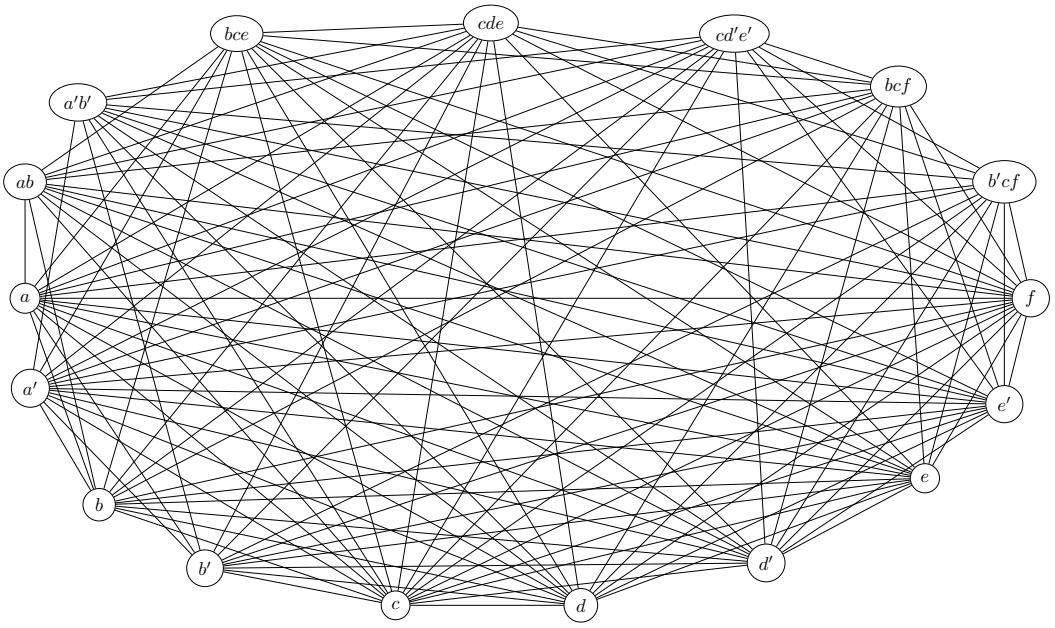

FIGURE 3.10 – La microstructure basée sur le codage mixte de l'exemple 2.1.

La microstructure basée sur le codage mixte du CSP de l'exemple 1 est donnée dans la figure 3.10. Nous pouvons constater que dans ce codage, nous gardons le même ensemble de sommets que celui de la HT-microstructure. Pour les arêtes, nous aurons celles de la DR-microstructure et de la HT-microstructure auxquelles s'ajoutent les arêtes reliant les valeurs issues des domaines différents, et correspondant à un sous-graphe n-parti complet. Une première observation sur ce graphe de microstructure est donnée par le lemme suivant :

Lemme 3.3. *Dans une ME-microstructure, une clique de* $n + e$ *sommets ne peut contenir ni deux valeurs d'une même variable, ni deux tuples d'une même relation.*

Preuve : Soient v_i et v'_i deux valeurs du domaine d'une variable x_i. Par définition, les sommets correspondants à v_i et v'_i ne peuvent être adjacents et donc, ne peuvent figurer ensemble dans une même clique. Pareillement pour les tuples. □

En s'appuyant sur ce lemme, nous pouvons illustrer la relation entre cliques et solutions de CSP :

Théorème 3.5. *Un CSP* P *possède une solution ssi* $\mu_{ME}(P)$ *possède une clique de taille* $n + e$*.*

Preuve : Dans la ME-microstructure, en se référant au lemme 3.3, une clique de $n + e$ sommets contient exactement un sommet par variable et par contrainte. Donc, elle correspond à une affectation de n variables qui satisfait les e contraintes, et il s'agit donc d'une solution. □

Comme pour les autres microstructures, la taille de la ME-microstructure est bornée par un polynôme fonction de la taille du CSP.

– $|V| = \Sigma_{x_i \in X}|D(x_i)| + \Sigma_{c_i \in C}|R(c_i)|$ et
– $|E| \leq \Sigma_{x_i \in X}|D(x_i)| \times \Sigma_{c_i \in C}|R(c_i)| + (\Sigma_{x_i \in X}|D(x_i)|)^2 + (\Sigma_{c_i \in C}|R(c_i)|)^2$.

De plus, calculer la ME-microstructure d'un CSP est réalisable en temps polynomial. Le temps de calcul sera de l'ordre de O($nd + er + e^2r^2 + nder + n^2d^2$) avec :

– nd pour calculer les sommets correspondants aux valeurs,
– er pour calculer les sommets associés aux tuples,
– e^2r^2 pour les arêtes entre les tuples,
– $nder$ pour les arêtes entre les tuples et les valeurs originelles et
– n^2d^2 pour les arêtes entre les valeurs originelles.

Le sous-graphe de ME-microstructure, qui concerne les valeurs est un graphe complet. Dans ce qui suit, nous allons essayer de minimiser le nombre d'arêtes de ce sous-graphe et le nouveau graphe de microsctructure sera appelé MME-microstructure (pour **M**inimal **M**ixed **E**ncoding). Nous commencerons par la définition de graphe mixte minimal ensuite nous énoncerons la définition de ce CSP binaire mixte minimal :

Définition 3.12 (graphe mixte minimal)**.** *Étant donné un CSP $P = (X, D, C)$ d'arité quelconque, la représentation **mixte minimale** de P est un graphe non-orienté $GMM(P) = (V, E)$ avec :*

– $V = \{x_i^m \in X^m\}$ *:*
– $E = E_1 \cup E_2 \cup E_3$
  - $E_1 = \{\{c_i, c_j\} \mid i \neq j$ *et* $S(c_i) \cap S(c_j) \neq \emptyset\}$,
  - $E_2 = \{\{c_i, x_j\} \mid c_i \in C$ *et* $x_j \in X$ *telles que* $x_j \in S(c_i)\}$,
  - $E_3 = \{\{x_i, x_j\} \mid x_i, x_j \in X$ *telles que* $\exists c_k \in C \mid \{x_i, x_j\} \subseteq S(c_k)\}$.

Ainsi, il s'agit de combiner le graphe primal, le graphe dual et le graphe de variables cachées.

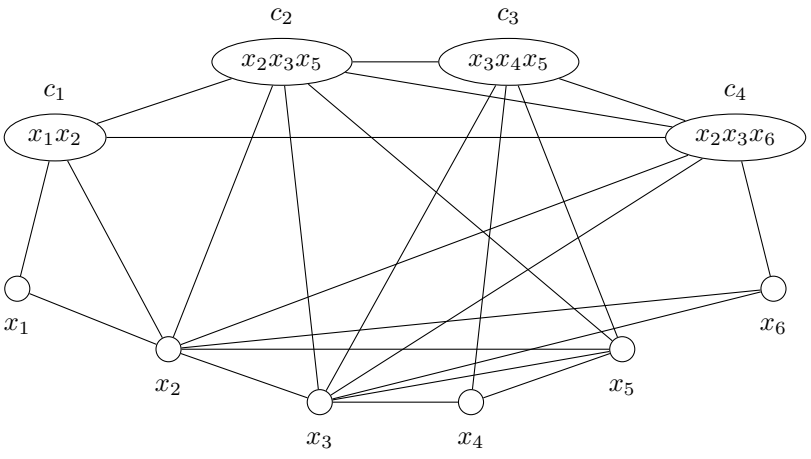

FIGURE 3.11 – Le graphe mixte minimal de l'exemple 2.1.

La figure 3.11 est la représentation mixte minimale de l'exemple 2.1.

Définition 3.13 (CSP mixte minimal)**.** *Le CSP **mixte minimal** du CSP $P = (X, D, C)$ est le CSP binaire $P^M = (X^M, D^M, C^M)$ avec :*
- $X^M = X^m$,
- $D^M = D^m$,
- $C^M = C^d \cup C^h \cup C^p$ *avec* $C^p = \{c^p_{ij} \mid \exists c_k \in C$ *tel que* $\{x_i, x_j\} \subseteq S(c_k)\}$ *et* $R(c^p_{ij}) = \cap_{c_k \in C | \{x_i,x_j\} \subseteq S(c_k)} R(c_k)[\{x_i, x_j\}]$

Nous définissons maintenant la microstructure associée.

Définition 3.14 (MME-microstructure)**.** *Étant donné un CSP $P = (X, D, C)$ d'arité quelconque, la* Microstructure basée sur le codage mixte minimal *de $P^M = (X^M, D^M, C^M)$ est un graphe non-orienté $\mu_{ME}(P) = (V, E)$ avec :*
- $V = S_1 \cup S_2$ *tels que*
  - $S_1 = \{(c_i, t_i) : c_i \in C, t_i \in R(c_i)\}$,
  - $S_2 = \{(x_j, v_j) : x_j \in X, v_j \in D(x_j)\}$,
- $E = E_1 \cup E_2 \cup E_3$ *tels que*
  - $E_1 = \{ \{(c_i, t_i), (c_j, t_j)\} \mid i \neq j, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]\}$
  - $E_2 = \{ \{(c_i, t_i), (x_j, v_j)\} \mid$ *soit* $x_j \in S(c_i)$ *et* $v_j = t_i[x_j]$, *soit* $x_j \notin S(c_i)\}$
  - $E_3 = \{ \{(x_i, v_i),\ (x_j, v_j)\} \mid\ x_i \neq x_j$ *et* $\forall\ c_k \mid\ \{x_i, x_j\} \subseteq S(c_k)$ *et* $(v_i, v_j) \in R(c_k)[\{x_i, x_j\}]\}$.

Nous pouvons observer que, dans le cas binaire, l'ensemble E_3 correspond à la microstructure classique définie dans [Jégou, 1993a] pour les CSP binaires. Nous pouvons facilement constater que la taille de cette MME-microstructure est inférieure à la taille de la ME-microstructure. Par contre, il faut compter un coût supplémentaire pour le calcul du test d'existence d'arêtes entre chaque couple de valeurs.

La figure 3.12 correspond à la MME-microstructure de l'exemple 2.1. Par rapport à la ME-microstructure, certaines arêtes ont disparu comme les arêtes qui relient les sommets (a) et (a') à (b').

La relation entre clique et solution est donnée ci-dessous :

Théorème 3.6. *Dans la MME-microstructure, une solution est équivalente à une $(n + e)$-clique.*

Preuve : Dans une ME-microstructure, une clique de taille $n + e$ correspond à une solution. Dans la MME-microstructure, nous avons seulement supprimé des arêtes entre des valeurs qui n'ont été autorisées par aucune contrainte. Donc, la satisfiabilté est préservée par rapport à la ME-microstructure et une $(n + e)$-clique est équivalente à une solution. □

Maintenant, nous pouvons déduire la relation entre la ME-microstructure et la MME-microstructure.

Théorème 3.7. *Étant donné un CSP P, la $\mu_{MME}(P)$ est un graphe partiel de la $\mu_{ME}(P)$.*

Preuve : Les deux microstructures sont constituées de trois sous-graphes :
- un sous-graphe définissant les relations entre tuples (c'est la DR-microstructure),
- un sous-graphe reliant les tuples aux valeurs (cela correspond exactement à la HT-microstructure),
- un sous-graphe définissant les relations entre les valeurs.

Le dernier sous-graphe correspond à un sous-graphe complet dans la ME-microstructure. Dans la MME-microstructure, ce sous-graphe ne correspond pas toujours à un sous-graphe complet. Par exemple, les sommets de x_1 et x_2 sont inter-connectés par un sous-graphe complet dans la ME-microstructure et un sous-graphe non complet dans la MME-microstructure (voir figure 3.12 et 3.10). Donc, l'ensemble des arêtes de la MME-microstructure est inclus dans l'ensemble des arêtes de la ME-microstructure. □

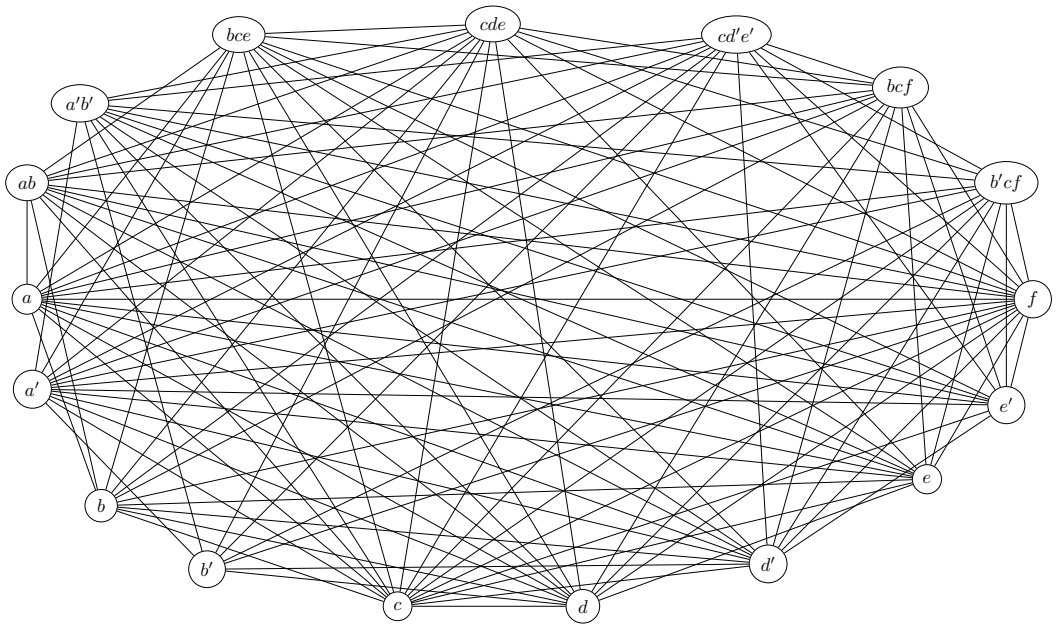

FIGURE 3.12 – La microstructure basée sur le CSP mixte minimal de l'exemple 2.1.

Nous pouvons constater qu'aucune de ces microstructures ne peut être considérée comme une généralisation de la microstructure présentée pour le cas binaire. En effet, étant donné un CSP binaire P, nous avons $\mu(P) \neq \mu_{DR}(P), \mu(P) \neq \mu_{HT}(P)$ et $\mu(P) \neq \mu_{ME}(P)$. Comme les HT et DR-microstructure sont incomparables avec la microstructure classique, alors la ME-microstructure le sera aussi. De plus, si la DR-microstructure correspond exactement à la microstructure du dual, ni la HT-microstructure, ni la ME-microstructure ne correspondent à la microstructure du codage binaire associé, ceci étant dû à la façon dont les graphes ont été complétés.

D'un point de vue structurel, étant donné un CSP binaire P, si la microstructure classique de P ne satisfait pas les conditions d'une classe polynomiale, cela n'empêche pas les autres microstructures (DR, DSR, HT, ME ou MME-microstructure) de les satisfaire. Par exemple, nous pouvons trouver, pour un CSP binaire P, une microstructure $\mu(P)$ triangulée alors que $\mu_{DR}(P)$ est non-triangulée (figure 3.13) et inversement (figure 3.14). Pareillement pour les autres microstructures.

Si une k-clique ($k < n$) dans la microstructure classique correspond toujours à une affectation cohérente d'un ensemble de k variables $Y \subseteq X$, ceci n'est vrai que pour la DR-microstructure. Autrement, dans une microstructure autre que la DR-microstructure, une clique ne correpond pas toujours à une affectation cohérente. Par exemple, (c, c', ab) est une (2,1)-biclique dans la HT-microstructure de la figure 3.7 qui ne correspond pas à une affectation cohérente.

Par ailleurs, si finalement toutes ces microstructures peuvent être calculées en temps polynomial, d'un point de vue pratique, il semble difficile en général de les construire et de les manipuler efficacement, en particulier quand les contraintes ne sont pas données en extension et ceci même pour la microstructure de CSP binaires. Par contre, ce dernier point ne présente pas un obstacle pour l'étude théorique que nous désirons proposer. Il faut rappeler que notre but consiste à introduire un outil théorique qui nous permettra d'étudier les CSP d'arité quelconque.

Dans tout le reste de ce manuscrit, nous considérerons seulement la DR, HT et ME-microstructure dans notre étude d'extension des classes polynomiales. La section suivante présentera des nouveaux résultats portant sur l'exploitation de ces représentations afin d'étendre aux

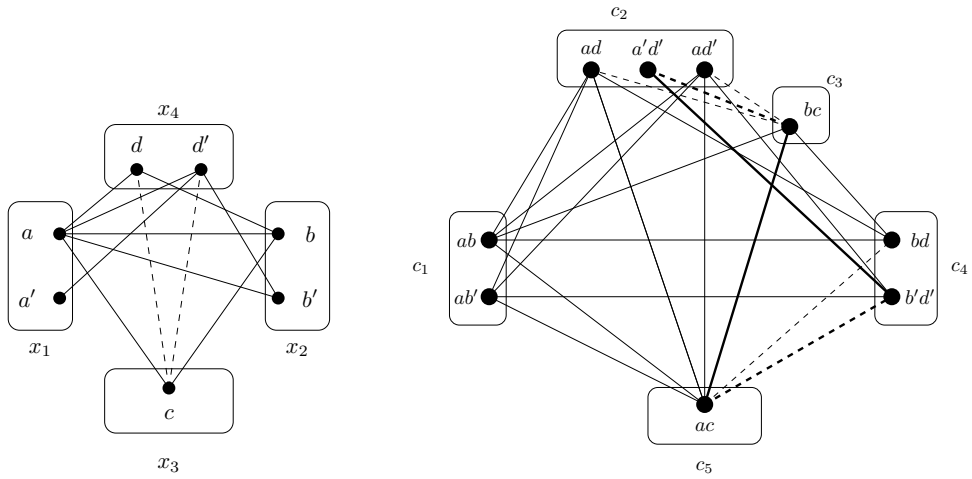

FIGURE 3.13 – Microstructure triangulée (a) et DR-microstructure non-triangulée (b). Les arêtes en pointillés indiquent les relations universelles et les arêtes en gras désignent un cycle sans corde de longueur supérieure ou égale à 4.

CSP d'arité quelconque, la classe polynomiale relationnelle ZOA définie initialement pour les seuls CSP binaires.

3.3 Microstructures et Classe ZOA

Ici, nous illustrerons l'étude théorique présentée dans la section précédente. En effet, nous allons présenter quelques résultats qui peuvent être déduits de l'étude des microstructures. Par exemple, la classe polynomiale ZOA [Cooper et al., 1994] a été initialement introduite pour les CSP binaires. Cette propriété peut être représentée graphiquement en utilisant la microstructure classique. Nous allons maintenant l'étendre en utilisant les microstructures proposées dans les sections précédentes aux CSP d'arité quelconque.

3.3.1 ZOA et la DR-microstructure

Dans cette section, nous proposerons une extension de ZOA aux CSP d'arité quelconque en utilisant la DR-microstructure (la nouvelle propriété sera appelée DZOA). En effet, il semble facile d'appliquer le même principe que pour la microstructure classique des CSP binaires.

Pour cela, par rapport à la définition du ZOA pour le cas binaire, il suffit seulement de remplacer les valeurs par des tuples. Donc, satisfaire DZOA dépendra des relations du CSP. Formellement, la définition sera comme suit :

Définition 3.15 (DZOA)**.** *Un CSP P d'arité quelconque satisfait DZOA (pour Dual Zéro/Un/Tous) si pour chaque contrainte c_{ij}^d de C^d, pour chaque tuple $t_i \in D(x_i^d)$, $R(c_{ij}^d)$ vérifie l'une des conditions suivantes :*

- *(ZÉRO)* $\forall\, t_j \in D(x_j^d), t_i[S(c_i) \cap S(c_j)] \neq t_j[S(c_i) \cap S(c_j)]$,
- *(UN) il existe un seul tuple* $t_j \in D(x_j^d)$ *tel que* $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$,
- *(TOUS)* $\forall\, t_j \in d^D(x_j), t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$.

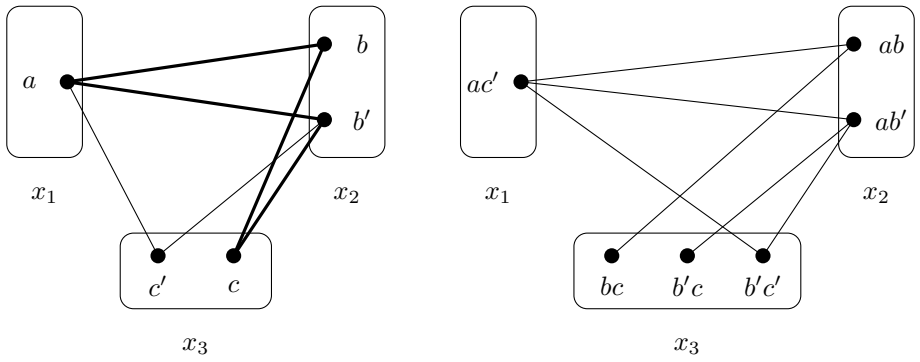

FIGURE 3.14 – Microstructure non-triangulée (a) et DR-microstructure triangulée.

Dans le cas binaire, cette nouvelle propriété (DZOA) est différente de ZOA. La figure 3.15 représente l'exemple d'un CSP binaire satisfaisant ZOA mais qui n'est pas DZOA ((a, b) est en relation avec (a, c') et (a, c) mais non (a'', c) et (a', c)).

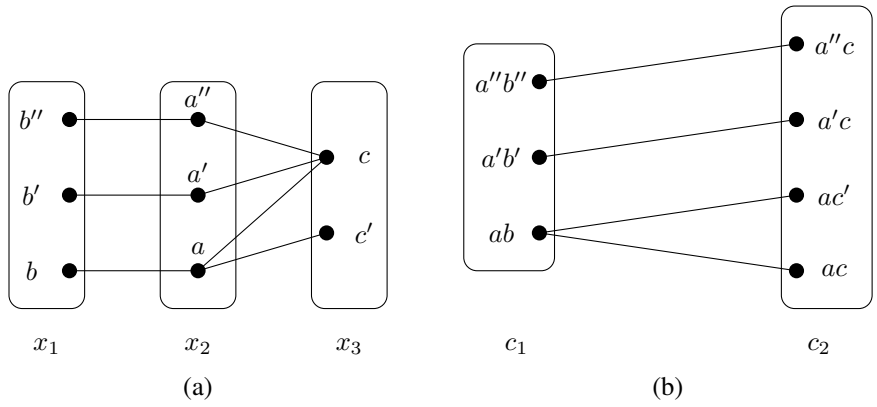

FIGURE 3.15 – Un CSP satisfaisant ZOA mais pas DZOA.

La figure 3.16 illustre le cas d'un CSP binaire satisfaisant DZOA mais pas ZOA (à cause de la valeur a' qui est en relation avec b' et b'' mais pas avec b), ceci confirmera le théorème suivant :

Théorème 3.8. *Étant donné un CSP binaire P,*
- *P satisfait ZOA $\not\Rightarrow$ P satisfait DZOA,*
- *P satisfait DZOA $\not\Rightarrow$ P satisfait ZOA.*

Dans le chapitre 2, nous avons rappelé que la propriété ZOA peut se manifester sous la forme de plusieurs configurations comme *« double éventail »* (voir figure 2.14) de ZOA. Dans le cas de la DR-microstructure, cette configuration n'existe pas.

Lemme 3.4. *Dans un CSP non monovalent, toute relation $R(c_{ij}^d)$ telle que $S(c_i) \cap S(c_j) \neq \emptyset$ ne peut pas être double éventail.*

Preuve : (par absurde) Supposons que dans un CSP non monovalent, il existe une relation $R(c_{ij}^d)$ qui vérifie la configuration *double éventail*. Il existe évidemment un tuple t_i dans $R(c_i)$ qui est en

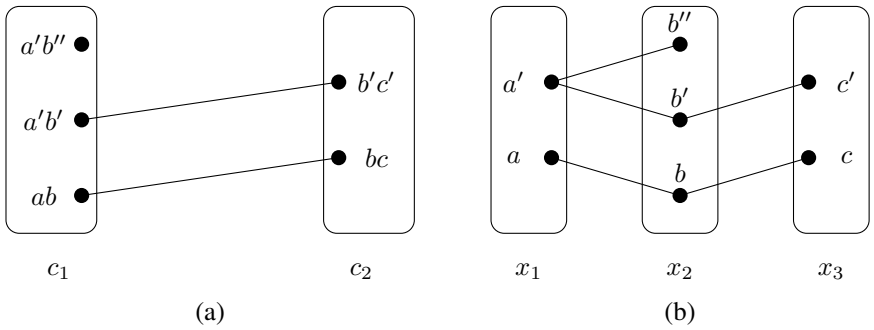

FIGURE 3.16 – Un CSP satisfaisant DZOA mais pas ZOA.

relation avec tous les tuples de $R(c_j)$ et un tuple t_j dans $R(c_j)$ qui est en relation avec tous les tuples de $R(c_i)$. Donc, quel que soit t'_j dans $R(c_j)$ on a $t_i[S(c_i) \cap S(c_j)] = t'_j[S(c_i) \cap S(c_j)]$ et quel que soit t'_i dans $R(c_i)$ on a $t'_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$ (a). Ceci implique que $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$ (b). Or, pour $t''_i \neq t_i$ et $t''_j \neq t_j$ on a : $t''_i[S(c_i) \cap S(c_j)] \neq t''_j[S(c_i) \cap S(c_j)]$. D'après (a), on peut déduire $t''_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$ et $t_i[S(c_i) \cap S(c_j)] = t''_j[S(c_i) \cap S(c_j)]$. Par conséquent, $t_i[S(c_i) \cap S(c_j)] \neq t_j[S(c_i) \cap S(c_j)]$ ce qui est contradictoire avec (b). □

Contrairement au cas binaire, nous pouvons déduire que la configuration *double éventail* ne peut se présenter dans le cas non-binaire en utilisant la DR-microstructure.

3.3.2 ZOA et la HT-microstructure

Pour la HT-microstructure, les arêtes relient les tuples (sommets associés aux variables cachées) aux valeurs (sommets associés aux variables originelles du CSP). Nous étudierons la propriété ZOA dans cette microstructure (elle sera notée HZOA) et nous analyserons la situation selon deux directions : des tuples vers les valeurs et des valeurs vers les tuples.

- *Arêtes des tuples vers les valeurs*. Il existe deux façons pour connecter un tuple t_i à une valeur v_j.
  - Si la variable x_j de cette valeur appartient à la portée de la contrainte c_i de ce tuple, alors t_i est connecté seulement à la valeur y figurant.
  - Si la variable x_j n'appartient pas au scope de la contrainte c_i de tuple t_i, alors t_i est connecté à toutes les valeurs de x_j. Donc, nous avons seulement des connexions de type « Un » ou « Tous ».
- *Arêtes des valeurs vers les tuples*. Étant donnée une contrainte associée à la HT-microstructure, une valeur est connectée aux tuples dans lesquels elle apparaît. Nous discutons ci-dessous les différentes possibilités :
  - connexion "Zéro". Une valeur n'est pas supportée par des tuples. C'est l'équivalent de la connexion "Zéro" de la définition de ZOA pour le CSP binaire.
  - connexion "Un". Une valeur est supportée par un seul tuple. C'est l'équivalent de la connexion "1" de la définiton de ZOA pour les CSP binaires.
  - connexion "Tous". Une valeur est supportée par tous les tuples de la contrainte. C'est aussi l'équivalent de la connexion "Tous" de la définition de ZOA pour les CSP binaires.

Il est donc possible qu'un CSP satisfasse la propriété ZOA dans la HT-microstructure. La figure 3.8 satisfait la propriété HZOA. Comme pour la DR-microstructure, nous comparons maintenant ZOA à HZOA. Nous commençons par le théorème suivant :

Théorème 3.9. *Si un CSP binaire P satisfait HZOA $\Rightarrow$ P satisfait ZOA.*

Preuve : (par absurde) Étant donné un CSP binaire P qui satisfait HZOA mais pas ZOA, c'est-à-dire, il existe au moins une relation $R(c_{ij})$ qui ne satisfait pas la propriété ZOA. Donc, $R(c_{ij})$ contient une des trois configurations illustrées dans la figure 2.15 (du chapitre 2). Nous montrons maintenant qu'il est impossible d'avoir une de ces trois configurations :

- Pour la première configuration (figure 2.15(a) : si nous posons $t_1 = ab$, $t_2 = ab'$ et $t_3 = a'b''$, alors dans la HT-microstructure le sommet correspondant à la valeur a sera connecté aux sommets correspondants aux tuples t_1 et t_2 mais pas t_3. Ceci implique que le CSP P ne satisfait pas HZOA, ce qui est impossible d'après l'hypothèse de départ.
- Pour la deuxième configuration (figure 2.15(b) : pareillement, si nous posons $t_1 = ab$, $t_2 = ab'$ $t_3 = a'b''$ et $t_4 = a'b'$ alors dans la HT-microstructure le sommet correspondant à la valeur a sera connecté aux sommets correspondants aux tuples t_1 et t_2 mais pas t_3 ni t_4. Ceci implique que le CSP P ne satisfait pas HZOA, ce qui est impossible d'après l'hypothèse de départ.
- Pour la troisième configuration (figure 2.15(c) : pareillement, si nous posons $t_1 = ab$, $t_2 = ab'$ $t_3 = a'b''$, $t_4 = a'b'$ et $t_4 = a'b$ alors dans la HT-microstructure le sommet correspondant à la valeur a sera connecté aux sommets correspondants aux tuples t_1 et t_2 mais pas t_3 ni t_4 ni t_5. Ceci implique que le CSP P ne satisfait pas HZOA, ce qui est impossible d'après l'hypothèse de départ.

Donc, ceci nous conduit à déduire que tout CSP satisfaisant HZOA satisfait aussi ZOA. □

Par contre, un CSP binaire P, qui satisfait ZOA, ne satisfait pas forcément HZOA. Revenons à l'exemple de la figure 3.15(a), la relation $R(c_2)$ est ZOA mais le CSP n'est pas HZOA car la valeur a est compatible avec les tuples (a, c) et (a, c') mais pas (a', c) et (a'', c).

3.3.3 ZOA et la (M)ME-microstructure

Enfin, pour la ME-microstructure, nous devons vérifier les conditions définies à la fois pour la DR et HT-microstructures. Pour le sous-graphe dont les sommets correspondent tous aux valeurs des variables, les relations entre les variables sont universelles et par conséquent la connexion "Tous" est toujours vraie. Pour la MME-microstructure, il faut vérifier les conditions de ZOA, DZOA et HZOA.

Pour conclure, par construction, rien ne s'oppose à ce que les conditions de la propriété ZOA soient satisfaites dans la (M)ME-microstructure, mais elles s'avèrent cependant très restrictives, comme d'ailleurs pour le cas binaire.

3.4 Conclusion

Dans ce chapitre, nous avons introduit le concept de microstructure pour le cas des CSP d'arité quelconque. Ce concept pour le cas binaire est désormais bien établi et utilisé comme un outil théorique, notamment pour la définition de nouvelles classes polynomiales pour les CSP. Pour le cas des CSP non-binaires, la notion de microstructure n'était pas clairement établie auparavant. Nous avons donc défini explicitement cette notion pour les CSP d'arité quelconque en nous basant sur les différents codages binaires de CSP étudiés précédemment dans la littérature, de sorte à disposer de microstructures graphiques, plutôt qu'en nous appuyant sur la notion d'hypergraphe comme proposé dans [Cohen, 2003]. Nous avons présenté trois types de microstructures : la DR-microstructure (raffinée avec la DSR-microstructure), la HT-microstructure et la ME-microstructure, qui sont inspirées respectivement de la représentation duale, de la transformation

par variable cachée et de l'approche mixte. Pour le cas binaire, aucune de ces trois microstructures ne correspond à la microstructure classique, de sorte qu'aucune d'entre elles ne peut être considérée comme une généralisation de la notion binaire.

Pour montrer l'intérêt de ce travail, nous avons exploité ces microstructures afin d'étendre la classe polynomiale ZOA définie au niveau des CSP binaires sur la base de la microstructure classique.

Dans les prochains chapitres, nous montrerons comment ces microstructures pourront être utilisées pour étendre certaines autres classes polynomiales comme BTP (chapitre 6) ou définir une nouvelle classe polynomiale pour les CSP d'arité quelconque en nous basant sur le nombre de cliques maximales dans ces graphes (Chapitre 4).

Nous espérons que ces outils seront utilisés au niveau non-binaire comme cela a pu être le cas au niveau binaire pour la microstructure classique. Par contre, il est clair qu'une utilisation pratique de ces notions semble plus que difficile, en particulier pour le cas des contraintes pour lesquelles les relations ne sont pas définies en extension. Toutefois, ces microstructures pourraient être utilisées virtuellement comme dans le cas notamment des filtrages par cohérence locale.

Dans le chapitre suivant, nous proposerons une nouvelle classe polynomiale pour les CSP binaires. Puis, dans le même esprit, nous essayerons de l'étendre au cas n-aire en utilisant les différentes microstructures basées sur les codages binaires.

Chapitre 4

Les microstructures et le nombre de cliques maximales

4.1 Introduction

Dans l'état de l'art, nous avons rappelé que le problème de décision pour CSP est NP-complet. Néanmoins, il existe des classes de CSP, appelées *classes polynomiales*, qui peuvent être résolues en temps polynomial. La plupart des classes polynomiales se présentent rarement en pratique, ce qui diminue d'autant leur intérêt. En revanche, des algorithmes tels que FC, nFC$_i$, RFL ou MAC, dont la complexité en temps théorique est exponentielle (en O(ed^n)), sont à la base de systèmes pratiques pour la résolution de contraintes, et leurs résultats sont souvent impressionnants en termes de temps de calcul, alors qu'ils exploitent *rarement* ces classes polynomiales.

Dans ce chapitre, nous essaierons d'amoindrir le fossé existant entre les travaux théoriques sur les classes polynomiales et l'efficacité pratique des méthodes usuelles, cela en tentant de fournir des éléments de réponse à la question portant sur les raisons de l'efficacité observée pour des algorithmes tels que (n)FC ou (n)RFL. Nous le faisons en réévaluant leur complexité en temps en utilisant un nouveau paramètre, à savoir le nombre de cliques maximales dans la *microstructure* d'un CSP ou dans les *microstructures* non-binaires proposées dans le chapitre 4. Pour le cas binaire, si $\omega_{\#}(\mu(P))$ exprime le nombre de cliques maximales figurant dans la microstructure d'un CSP P, nous montrons que la complexité d'un algorithme tel que FC est en O($n^2 d \cdot \omega_{\#}(\mu(P))$). Cela fournit une nouvelle perspective pour l'étude de l'efficacité des algorithmes de type backtracking en l'associant à un paramètre bien connu en théorie des graphes. En particulier, en réutilisant des résultats connus de la théorie des graphes, nous proposons de nouvelles classes polynomiales de CSP. Le trait saillant de ces classes est qu'elles sont résolues en temps polynomial par des algorithmes à la fois *très généraux* et *largement utilisés*, *sans requérir la nécessité de disposer d'algorithmes de reconnaissance de classes*. À cet égard, notre étude est très proche dans l'esprit de l'étude menée par Rauzy dans le cadre de la satisfiabilité de formules propositionnelles et le comportement de l'algorithme DPLL sur les cas connus de classes polynomiales SAT [Rauzy, 1995].

Ce chapitre est organisé comme suit. Nous présenterons d'abord notre analyse de la complexité de BT, FC, et RFL sur les CSP binaires, puis nous étendrons notre étude aux CSP non-binaires et donc à des algorithmes de la classe nFC$_i$. Nous mettrons ensuite en évidence de nouvelles classes polynomiales issues de la théorie des graphes, qui peuvent ainsi être exploitées dans le domaine des CSP. Ensuite, nous montrerons que cette étude ne pourra pas être étendue au cas non-binaire en utilisant la HT et la ME-microstructure. Cette étude a déjà fait l'objet de publications [El Mouelhi et al., 2012, 2013c]

4.2 Une nouvelle analyse de la complexité pour les CSP binaires

Nous présentons maintenant une analyse de la complexité des algorithmes classiques en nous appuyant sur des paramètres liés à la microstructure. Pour cela, nous introduisons la notion de nœud cohérent maximalement profond.

Définition 4.1. *Un nœud de l'arbre de recherche est un nœud cohérent maximalement profond s'il est cohérent (correspond à une affectation cohérente) et s'il ne possède pas de nœud fils cohérent (quelle que soit la variable suivante).*

Ainsi, un tel nœud correspond soit à une solution, soit à une solution partielle qui ne peut être étendue de façon cohérente sur une variable ultérieure. L'exemple de la figure 4.2 présente une

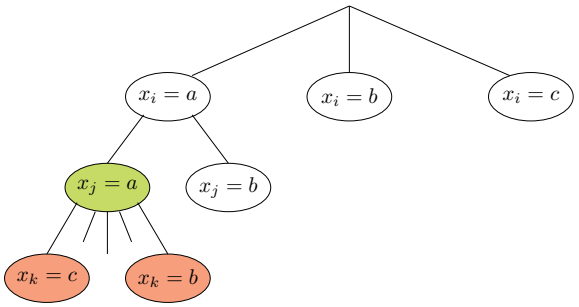

FIGURE 4.1 – Exemple d'un nœud cohérent maximalement profond (en vert).

partie de l'arbre de recherche d'un CSP qui doit affecter la même valeur à toutes les variables. Le nœud associé à la valeur a de la variable x_j correspond à un nœud cohérent maximalement profond car il est cohérent et tous ses nœuds fils sont sont incohérents. Par contre, le nœud associé à la valeur c de x_k ne correspond pas à un nœud cohérent maximalement profond car l'affectation (a,a,c) n'est pas cohérente.

Le résultat suivant peut-être considéré comme central pour notre étude.

Proposition 4.1. *Étant donné un CSP binaire $P = (X, D, C)$, il existe une application injective de l'ensemble des nœuds cohérents maximalement profonds explorés par BT vers l'ensemble des cliques maximales de $\mu(P)$.*

Preuve : Soit $(v_1, v_2, \ldots, v_i)$ un nœud cohérent maximalement profond exploré par BT. Par définition, $(v_1, v_2, \ldots, v_i)$ est une solution partielle, et donc pour tout $1 \leq j, k \leq i$, soit il n'existe pas de contrainte c_{jk} dans C dont la portée est $\{x_j, x_k\}$, soit (v_j, v_k) figure dans la relation $R(c_{jk})$. Dans les deux cas, $\{(x_j, v_j), (x_k, v_k)\}$ constitue une arête de $\mu(P)$. Donc $\{(x_1, v_1), \ldots, (x_i, v_i)\}$ est une clique de $\mu(P)$ et donc, elle est incluse dans au moins une clique maximale de $\mu(P)$. Nous la noterons $Cl(v_1, v_2, \ldots, v_i)$.

Nous montrons maintenant que Cl constitue une application injective de l'ensemble des nœuds cohérents maximalement profonds vers l'ensemble des cliques maximales. Par construction de BT, si $(v_1, v_2, \ldots, v_i)$ et $(v'_1, v'_2, \ldots, v'_{i'})$ sont deux nœuds maximalement profonds explorés, ils doivent différer d'au moins une valeur sur une variable. Précisément, ils doivent différer au moins sur le nœud où les chemins correspondants se diffèrencient dans l'arbre de recherche, et donc sur les nœuds correspondants à une certaine variable x_j affectée à une certaine valeur sur un chemin, et à une autre valeur sur l'autre. Comme il n'y a pas d'arête dans $\mu(P)$ connectant deux valeurs d'une même variable, il ne peut pas y avoir de clique maximale contenant à la fois $(v_1, v_2, \ldots, v_i)$ et $(v'_1, v'_2, \ldots, v'_{i'})$. Donc, Cl est nécessairement une application injective. □

En utilisant cette propriété, nous pouvons facilement borner le nombre de nœuds figurant dans un arbre de recherche induit par une recherche de type backtracking, et donc aussi sa complexité en temps, en termes de propriété liée à sa microstructure. Comme il est d'usage, nous supposons qu'un test de contrainte (vérifier si $(v_i, v_j) \in R(c_{ij})$) est réalisable en temps constant.

Proposition 4.2. *Le nombre de nœuds $N_{BT}(P)$ figurant dans l'arbre de recherche développé par BT pour résoudre un CSP binaire $P = (X, D, C)$, vérifie $N_{BT}(P) \leq nd \cdot \omega_{\#}(\mu(P))$. Sa complexité en temps est en O($n^2 d \cdot \omega_{\#}(\mu(P))$).*

Preuve : Considérons d'abord le nombre de nœuds cohérents. Parce que n'importe quel nœud de l'arbre de recherche figure à une profondeur au plus égale à n et que le chemin de la racine à un nœud cohérent contient uniquement des nœuds cohérents, comme corollaire direct de la Proposition 4.1, nous obtenons que l'arbre de recherche contient au plus $n \cdot \omega_{\#}(\mu(P))$ nœuds cohérents. Maintenant, par définition de BT, un nœud cohérent possède au plus d fils (un par valeur candidate pour la variable suivante), et les nœuds incompatibles n'en ont pas. Il s'ensuit que l'arbre de recherche a au plus $nd \cdot \omega_{\#}(\mu(P))$ nœuds de toute nature.

La complexité en temps se déduit directement puisque chaque nœud correspond à l'extension de l'affectation partielle à une variable supplémentaire x_{i+1}, ce qui implique au plus un test de contrainte pour chaque autre variable déjà affectée (vérifier la satisfaction de $c_{j(i+1)}$ pour chaque variable x_j déjà affectée), et comme il n'en existe nécessairement moins de n, on obtient le résultat. □

On peut constater dans l'énoncé de la Proposition 4.2 ainsi que dans les suivantes, que le nombre de cliques maximales $\omega_{\#}(\mu(P))$ pourrait être remplacé par le nombre de cliques maximales *de taille au plus* $n - 1$. Ceci parce que dès qu'un chemin exploré est contenu dans une n-clique, c'est-à-dire, dans une solution, aucun retour arrière ne se produira ultérieurement sur ce chemin.

Nous analysons maintenant FC et RFL. Il apparaît clairement que la proposition 4.1 vaut également pour chacun de ces algorithmes. Le nombre de nœuds explorés découle du fait que seuls les nœuds cohérents sont explorés. La complexité en temps se déduit du fait que le coût de filtrage en un nœud par FC est nd et par RFL O(ed^2) (qui correspond à la meilleure complexité de AC).

Proposition 4.3. *Le nombre de nœuds $N_{FC}(P)$ figurant dans l'arbre de recherche développé par FC ou par RFL pour résoudre un CSP binaire $P = (X, D, C)$, vérifie $N_{FC}(P) \leq n \cdot \omega_{\#}(\mu(P))$. La complexité en temps de FC est en O($n^2 d \cdot \omega_{\#}(\mu(P))$), et celle de RFL est en O($ned^2 \cdot \omega_{\#}(\mu(P))$).*

Il est important de constater que la taille relative des arbres de recherche de BT et FC fournie par l'analyse classique et celle obtenue par l'approche que nous proposons sont les mêmes. Notamment, dans le pire des cas, l'arbre de recherche de BT est d fois plus grand en nombre de nœuds que celui de FC sur le même CSP.

En ce qui concerne les algorithmes comme MAC, son comportement est différent de ceux qui sont analysés ici, vu que MAC développe un arbre de recherche binaire (contrairement à BT, FC et RFL) et il ne choisit pas la même variable (à affecter) dans un même niveau de l'arbre de recherche. Par conséquent, MAC ne s'inscrit pas naturellement dans l'évaluation que nous avons proposée (voir figure 4.2). Donc, il faudrait une preuve différente.

Dans ce manuscrit, nous allons conjecturer, qu'il développe un nombre de nœud borné par le nombre de cliques maximales dans la microstructure mais sa complexité exacte doit également être étudiée.

Conjecture 4.1. *La complexité en temps de MAC pour résoudre un CSP P est polynomiale en fonction de nombre de cliques maximales dans $\mu(P)$.*

Dans le reste de ce manuscrit, nous utiliserons la notation suivante :

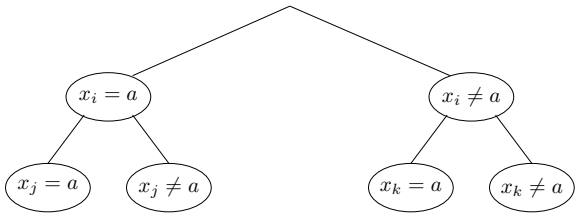

FIGURE 4.2 – Exemple d'un nœud cohérent maximalement profond (grisé).

Notation 4.1. *Nous notons CL l'ensemble des CSP binaires dont la microstructure possède un nombre polynomial de cliques maximales.*

4.3 Une nouvelle analyse pour les CSP non-binaires basée sur la DR-microstructure

Nous passons maintenant à l'étude des CSP n-aires. Dans le chapitre précédent, nous avons illustré plusieurs types de microstructures pour les CSP n-aires. Ici, nous montrons que la microstructure la plus adéquate pour étendre notre analyse au cas n-aire est la DR-microstructure car :

- Pour la *HT-microstructure*, un tel résultat ne peut exister. En effet, la HT-microstructure est un graphe biparti et donc ses cliques maximales sont toutes de taille 2 puisque les cliques maximales sont limitées aux arêtes. Donc, le nombre de cliques maximales est toujours polynomial.
- Pour la *ME-microstructure*, un tel résultat ne peut exister, mais pour des raisons différentes. Par construction, les arêtes de l'ensemble $E_3 = \{ \{(x_i, v_i),\ (x_j, v_j)\} \mid \ x_i \neq x_j\}$ de la définition 3.11 permettent toutes les combinaisons de valeurs possibles. Ceci rendra le nombre de cliques maximales exponentiel à l'exception des CSP monovalent (CSP dont la taille de tous les domaines est 1).

Donc, seule la DR-microstructure est exploitable ici. Nous analysons maintenant la complexité en temps des algorithmes de type nFC$_i$ en termes de nombre de cliques maximales dans cette microstructure.

4.3.1 Complexité de nBT et nFC

Nous étudions maintenant la complexité des algorithmes de résolution des CSP n-aires. Dans un premier temps, nous poserons une hypothèse sur l'ordre dans lequel ces algorithmes explorent les variables, puis nous discuterons de cette restriction.

Définition 4.2 (ordre compatible)**.** *Soit $P = (X, D, C)$ un CSP. Un ordre total $(x_1, x_2, \ldots x_n)$ sur X est dit* ***compatible avec les contraintes de*** *C s'il y a k contraintes $c_{i_1}, c_{i_2}, \ldots c_{i_k}$ dans C ($1 \leq k \leq e$) qui vérifient :*

- $\bigcup_{1 \leq l \leq k} S(c_{i_l}) = X$
- *il y a k variables $x_{i_1}, x_{i_2}, \ldots x_{i_k}$ telles que pour tout ℓ dans $\{1, \ldots, k\}$, $x_{i_\ell} \in S(c_{i_\ell})$ et $\bigcup_{1 \leq j \leq \ell} S(c_{i_j}) = \{x_i \mid i = i_1, \ldots, i_\ell\}$.*

En d'autres termes, l'ordre est tel que les variables figurant dans la portée d'une première contrainte c_{i_1} apparaissent toutes d'abord, puis toutes les variables figurant dans la portée d'une certaine contrainte C_{i_2} apparaissent ensuite (sauf pour celles qui figurent déjà dans c_{i_1}), etc. Les

variables $x_{i_1}, \ldots, x_{i_k}$ dans la définition sont telles que x_{i_j} est la dernière variable affectée dans la portée de c_{i_j}. Nous considèrerons ces variables comme étant des *jalons* dans l'ordre.

Par exemple, avec la notation de la définition, nous devons avoir $S(c_{i_1}) = \{x_1, x_2, \ldots x_{i_1}\}$ et $S(c_{i_1}) \cup S(c_{i_2}) = \{x_1, x_2, \ldots x_{i_1}, x_{i_1+1}, \ldots x_{i_2}\}$. La variable x_{i_1} est un jalon (dernière variable affectée dans la portée de c_{i_1}) de même que x_{i_2} pour c_{i_2}.

Dans l'hypothèse où un tel ordre sur les variables est utilisé, nous pouvons donner une généralisation de la proposition 4.1 au cas n-aire.

Proposition 4.4. *Soit P un CSP n-aire. Supposons que nBT explore les variables dans un ordre compatible avec les contraintes de C. Alors, il existe une application injective de l'ensemble des nœuds cohérents maximalement profonds (x_i, v_i) de l'arbre de recherche tels que x_i est un jalon, vers l'ensemble des cliques maximales de $\mu_{DR}(P)$.*

Preuve : Soit t une affectation correspondant à un nœud tel que défini, et notons x_{i_j} pour la dernière variable affectée dans t (x_{i_j} est un jalon par hypothèse). De même, soit t' une autre affectation maximale compatible avec le jalon $x_{i_{j'}}$ comme dernière variable. Notons T l'ensemble $\{t[S(c)] \mid c \in C, S(c) \subseteq \{x_1, \ldots, x_{i_j}\}\}$, c'est-à-dire, l'ensemble de toutes les projections de t sur les portées de toutes les contraintes totalement affectées par t, et idem pour T'. Alors T (resp. T') est inclus dans une clique maximale $Cl(t)$ (resp. $Cl(t')$) de $\mu_{DR}(P)$. Maintenant, supposons $j' \geq j$ (sans perte de généralité). Avec $t \neq t'$, le fait que t soit maximalement cohérent, et le fait que t' soit cohérent, permet d'affirmer que t' diffère de t d'au moins une variable x_ℓ avec $\ell \leq i_j$. Ainsi, cette variable est affectée différemment dans chaque cas, et il existe une certaine contrainte c_{i_ℓ} telle que $t[S(c_{i_\ell})]$ est différent de $t'[S(c_{i_\ell})]$, et donc, t et t' ne peuvent être inclus dans une même clique. Donc, Cl définit une application injective de l'ensemble des affectations considérées vers l'ensembles des cliques maximales de $\mu_{DR}(P)$. □

En utilisant cette propriété, nous pouvons borner le nombre de nœuds dans un arbre de recherche induit par une recherche de type backtracking, ainsi que sa complexité en temps, relativement à la DR-microstructure.

Proposition 4.5. *Soit $P = (X, D, C)$ un CSP n-aire. Supposons que nBT utilise un ordre sur les variables qui est compatible avec les contraintes de C. Le nombre de nœuds $N_{nBT}(P)$ dans l'arbre de recherche de nBT sur P vérifie $N_{nBT}(P) \leq nd^a \cdot \omega_\#(\mu_{DR}(P))$. Sa complexité en temps est en O($nea \cdot d^a \cdot \omega_\#(\mu_{DR}(P))$).*

Preuve : Du fait de la proposition 4.4, le sous-arbre induit par l'arbre de recherche sur les jalons contient au plus $\omega_\#(\mu_{DR}(P))$ nœuds. Pour atteindre un jalon à partir du précédent, c'est-à-dire, pour étendre une affectation de $x_1, \ldots, x_{i_j}$ à l'affectation de $x_1, \ldots, x_{i_{j+1}}$ (avec les notations de la définition 4.2), nBT explore au plus a variables (ceci par définition d'un ordre compatible avec les contraintes). Par conséquent, il explore au plus d^a combinaisons de valeurs (nBT n'a aucune raison pour exclure une affectation avant d'affecter toutes les variables figurant dans la portée d'une contrainte). Puisqu'une branche contient au plus n nœuds, et donc finalement n jalons, on obtient le résultat.

La complexité en temps se déduit directement puisque chaque nœud nécessite au plus e tests de contraintes, chacun pouvant s'effectuer en O(a) avec une structure de données appropriée. □

Un résultat similaire est valable pour nFC$_i$ ($i \geq 2$) et nRFL.

Proposition 4.6. *Soit $P = (X, D, C)$ un CSP n-aire. Supposons que nFC$_i$ ($i \geq 2$) et nRFL utilisent un ordre sur les variables qui est compatible avec les contraintes de C. Le nombre de nœuds $N_{nFC_i}(P)$ (respectivement $N_{nRFL}(P)$ dans l'arbre de recherche de nFC$_i$ (resp. nRFL) sur P vérifie $N_{nFC_i}(P) \leq nr \cdot \omega_\#(\mu_{DR}(P))$ (resp. $N_{nRFL}(P) \leq nr \cdot \omega_\#(\mu_{DR}(P))$). La complexité en temps de nFC$_i$ et nRFL est en O($nea \cdot r^2 \cdot \omega_\#(\mu_{DR}(P))$).*

Preuve : Du fait de la proposition 4.4, le sous-arbre induit par l'arbre de recherche sur les jalons contient au plus $\omega_{\#}(\mu_{DR}(P))$ nœuds. Pour atteindre un jalon à partir du précédent, c'est-à-dire, pour étendre une affectation de $x_1, \ldots, x_{i_j}$ à l'affectation de $x_1, \ldots, x_{i_{j+1}}$ (avec les notations de la définition 4.2), nFC$_i$ et nRFL explorent seulement les r tuples autorisés par c lors de l'exploration des variables dans $S(c)$, plutôt que toutes les d^a combinaisons de valeurs (en raison de l'utilisation de GAC).

Pour la complexité, nous devons tenir compte du coût supplémentaire dû à l'application de GAC, qui est O $(e \cdot a \cdot r)$ à chaque nœud. Ceci nous donne une complexité en O$(nr \cdot \omega_{\#}(\mu_{DR}(P)) \times ear)$ =O$(nea \cdot r^2 \cdot \omega_{\#}(\mu_{DR}(P)))$. □

Pour la suite, nous utiliserons la notation suivante :

Notation 4.2. *Nous notons DCL l'ensemble des CSP dont la DR-microstructure possède un nombre polynomial de cliques maximales.*

4.3.2 Complexité sans hypothèse sur l'ordre

On pourrait estimer que notre restriction posée sur les ordres d'affectation des variables en termes de compatibilité avec les contraintes n'est pas respectée par tous les ordres raisonnables. Par exemple, il n'y a aucune raison en général, pour qu'une heuristique très usitée telle que *dom / deg* ou $dom/wdeg$ respectent ce type d'ordre. Nous montrons donc ici que ce type de restriction s'avère nécessaire. Pour montrer cela, nous construisons une famille de CSP qui possèdent un nombre linéaire de cliques dans leur DR-microstructure (linéaire en son nombre de sommets), mais pour lesquelles nFC$_5$ explore un arbre de recherche de taille exponentielle (dans le nombre de sommets) pour un certain ordre sur les variables.

Les CSP (X, D, C) de cette famille sont construites comme suit. Avec e le nombre de contraintes, nous considérons deux variables distinctes x_0 et x'_0 de X qui seront communes à toutes les contraintes, et nous aurons $X \setminus \{x_0, x'_0\}$ qui est partitionné en e ensembles $X_1, \ldots, X_e$ (X_i sera associé à c_i). Ainsi, chaque contrainte $c_i \in C$ possède une portée $S(c_i) = \{x_0, x'_0\} \cup X_i$. Maintenant, le domaine est $\{v_1, \ldots, v_e\}$ pour toutes les variables ($d = e$). Enfin, les tuples permis pour une contrainte c_i sont précisément de la forme $((x_0, v_j), (x'_0, v_{i+j}), \ldots)$, pour $j = 1, \ldots, e$ (les indices sont pris modulo e) et sans restriction pour les affectations de X_i. On note que pour $i \neq i'$, les restrictions pour les tuples permis par c_i et $c_{i'}$ sur $\{x_0, x'_0\}$ ne coïncident jamais, de sorte qu'il n'existe aucune arête dans $\mu_{DR}(P)$. Ainsi, le nombre de cliques dans $\mu_{DR}(P)$ est exactement égal au nombre de sommets $|V| = e^{2+(n-2)/e}$. Ci-dessous, nous illustrons un exemple de relations qui satisfont les conditions énoncées précédemment.

$R(c_1)$			
x_0	x'_0	x_1	$\cdots$
v_1	v_2	v_{11}	$\cdots$
v_2	v_3	v_{12}	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

$R(c_2)$			
x_0	x'_0	x_2	$\cdots$
v_1	v_3	v_{21}	$\cdots$
v_2	v_4	v_{22}	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

$R(c_3)$			
x_0	x'_0	x_3	$\cdots$
v_1	v_4	v_{31}	$\cdots$
v_2	v_5	v_{32}	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

$\ldots$

$R(c_e)$			
x_0	x'_0	x_e	$\cdots$
v_1	v_{e-1}	v_1	$\cdots$
v_2	v_e	v_2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

D'autre part, supposons que nFC$_5$ explore toutes les variables dans les ensembles X_i et explore seulement x_0 et x'_0 après elles. Ainsi, puisque que toutes les valeurs pour x_0 possèdent un support dans toutes les contraintes, et de même pour x'_0, aucune valeur ne sera retirée avant d'atteindre x_0 ou x'_0, et donc toutes les $e^{n-2} \sim |V|^e$ combinaisons de valeurs seront explorées, c'est-à-dire, elles seront exponentielles et donc très supérieures au nombre de cliques figurant dans $\mu_{DR}(P)$. La figure 4.3 présente l'arbre de recherche développé par nFC$_5$ s'il instancie x_0 et x'_0 en dernier.

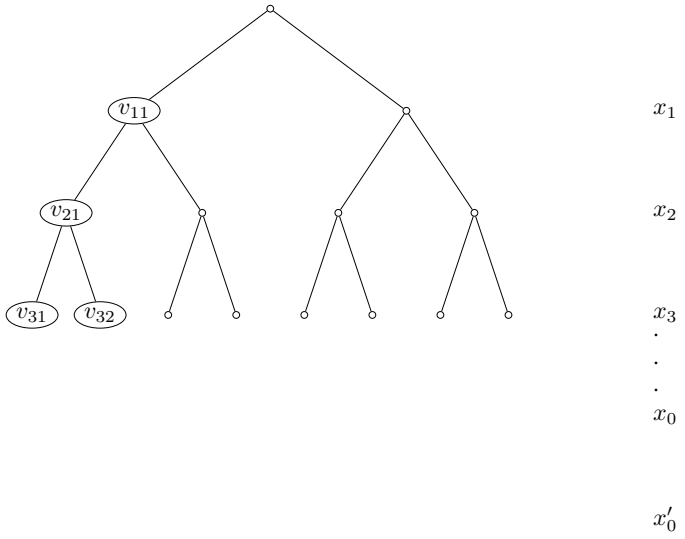

FIGURE 4.3 – Une partie d'un exemple où un algorithme comme nFC$_5$ développe un arbre dont le nombre de nœuds est exponentiel alors que le nombre de clique dans la DR-microstructure est polynomial.

4.4 Quelques classes polynomiales pour le backtracking

Le nombre de cliques dans un graphe peut croître de façon exponentielle avec la taille du graphe [Wood, 2007] et il en est de même du nombre $\omega_{\#}(G)$ de *cliques maximales* dans un graphe G [Moon and Moser, 1965]. Toutefois, pour certaines classes de graphes, le nombre de cliques maximales peut être borné par un polynôme en la taille du graphe. Si la (DR-)microstructure d'un (d'une famille de) CSP P appartient à l'une de ces classes, l'analyse que nous avons présentée dans les sections précédentes permet de conclure que P est résoluble en temps polynomial par des algorithmes classiques d'énumération, et ce, *sans avoir à reconnaître l'appartenance du CSP à cette classe*.

Dans cette section, nous étudierons plusieurs classes de graphes vérifiant cette propriété et nous commenterons leur pertinence en termes de problèmes de satisfaction de contraintes.

4.4.1 Graphes "sans-triangle" ou bipartis

Nous rappelons qu'un k-*cycle* dans un graphe $G = (V, E)$ est une séquence $(v_1, v_2, \ldots v_{k+1})$ de sommets distincts, sauf pour $v_1 = v_{k+1}$, vérifiant $\forall i, 1 \leq i \leq k, \{v_i, v_{i+1}\} \in E$.

Définition 4.3. *Un graphe **sans-triangle** (ou **triangle-free** en anglais) est un graphe non-orienté sans* 3*-cycle.*

Il est facile de voir que le nombre de cliques maximales dans un graphe *sans-Triangle* est exactement égal au nombre de ses arêtes. En effet, chaque arête est une clique, et par définition, il ne peut exister de plus grande clique. Avec notre analyse, nous pouvons affirmer que pour la classe des CSP dont la (DR-)microstructure est *sans-Triangle*, les algorithmes (n)BT, (n)FC et (n)RFL sont des procédures de résolution fonctionnant en temps polynomial. Notons cependant que cette classe de CSP constitue d'une certaine façon un cas *dégénéré*, puisque excepté pour les CSP ayant au plus

deux variables (cas binaire) ou deux contraintes (cas n-aire), les CSP avec (DR-)microstructure *sans-triangle* sont incohérentes.

Un autre cas dégénéré est constitué par la classe des graphes bipartis. Un graphe est dit *biparti* s'il ne contient pas de cycle impair. Encore une fois, un graphe biparti ne peut contenir de clique de plus de deux sommets, et donc aucune affectation partielle de plus de trois variables ne sera considérée par BT (dont la complexité sera ainsi en O(d^3)).

Nous passons maintenant à des classes plus intéressantes, qui aussi, contiennent pour l'essentiel des CSP incohérents, mais pour lesquelles notre analyse fournit une meilleure complexité en temps que celle offerte par l'analyse classique.

4.4.2 Graphes planaires, toroïdaux, et "embedded"

Définition 4.4 (planaire)**.** *Un graphe **planaire** est un graphe qui peut être dessiné dans le plan sans que deux arêtes ne se chevauchent.*

[Wood, 2007] a prouvé que le nombre de cliques d'un graphe planaire $G = (V, E)$ est au plus $8(|V| - 2)$.

Définition 4.5 (toroïdal)**.** *Un graphe **toroïdal** est un graphe qui peut être dessiné sur un tore sans que deux arêtes ne se chevauchent.*

Une généralisation de la définition des graphes planaires et toroïdaux, notée *graphe plongeable*, est donnée dans la définition suivante :

Définition 4.6 (plongeable)**.** *Un graphe **plongeable** (ou « Embedded » en anglais) est un graphe qui peut être dessiné sur une surface (sphère,...) sans que deux arêtes ne se chevauchent.*

[Dujmovic et al., 2011] ont montré que tout graphe toroïdal possède au plus $8(|V| + 9)$ cliques et que tout graphe plongeable dans une surface possède un nombre linéaire de cliques puisque majoré par $8(|V|+27)$ au pire. Puisque la microstructure $\mu(P)$ (resp. $\mu_{DR}(P)$) d'un CSP P contient nd sommets (resp. er sommets), alors si $\mu(P)$ (resp. $\mu_{DR}(P)$) appartient à l'une de ces classes de graphes, alors $\omega_{\#}(\mu(P))$ (resp. $\omega_{\#}(\mu_{DR}(P))$) possède au plus O(nd) cliques (resp. O(er)).

Grâce aux propositions 4.2-4.3 et 4.5-4.6, nous obtenons immédiatement le résultat suivant.

Théorème 4.1. *Soit Em qui désigne la classe de tous les CSP dont les microstructures sont planaires, toroïdales, ou plongeables dans une surface. Alors les CSP de Em sont résolus en un temps*
- *O($n^2 d \cdot \omega_{\#}(\mu(P))$)=O($n^3 d^2$) par BT ou FC,*
- *O($ned^2 \cdot \omega_{\#}(\mu(P))$)=O($n^2 ed^3$) par RFL,*
- *O($nad^a \cdot \omega_{\#}(\mu(P))$)=O($ne^2 ard^a$) par nBT,*
- *O($nar^2 \cdot \omega_{\#}(\mu(P))$)=O($ne^2 ar^3$) par nFC$_i$ ou nRFL.*

Rappelons que cette famille de graphes ne peut contenir en tant que mineur ni une 8-clique (pour les graphes toroïdaux), ou ni une 5-clique ou $K_{3,3}$ (pour les graphes planaires). Cela a pour conséquence, en particulier, que tous les CSP binaires (resp. n-aires) dans Em sur au moins 8 variables (resp. contraintes) sont incohérents. On peut donc dire raisonnablement que cette classe est pour le moins dégénérée. Néanmoins, si l'on se réfère à l'analyse classique de la complexité, on constate que, par exemple, BT résout ces CSP en un temps O(d^8), pour le cas où d est grand, ce temps est significativement supérieur à O($n^3 d^2$).

4.4.3 Graphes CSG

Nous étudions maintenant la classe des *Graphes CSG*[Chmeiss and Jégou, 1997] dont nous avons rappelé sa définition dans le chapitre 2. Ici, nous précisons que les graphes CSG^k possèdent

au plus $|V|^k$ cliques maximales [Chmeiss and Jégou, 1997] et qu'il existe un algorithme de complexité polynomiale en O($|V|^{2(k-1)}(|V| + |E|)$) pour les énumérer. L'existence de cet algorithme confère à la classe des CSP qui ont une (DR-)microstructure CSG^k le statut de classe polynomiale pour toute valeur k fixée. Nous sommes cependant en mesure de montrer que des algorithmes *génériques* tels que (n)BT, FC, nFC$_i$ ou (n)RFL s'exécutent en temps polynomial sur de tels CSP, sans même qu'il soit nécessaire de reconnaître l'appartenance des CSP traités à cette classe (et donc sans avoir besoin de calculer la microstructure). À nouveau, ce résultat est issu du nombre de cliques maximales par l'application des propositions 4.1-4.3 et 4.5-4.6.

Théorème 4.2. *Pour tout entier k fixé, un CSP dont la (DR-)microstructure est CSG^k peut être résolu en un temps*

- *O($n^2d \cdot \omega_{\#}(\mu(P))$)=O($n^{k+2}d^{k+1}$) par BT et FC,*
- *O($ned^2 \cdot \omega_{\#}(\mu(P))$)=O($n^{k+1}ed^{k+2}$) par RFL,*
- *O($nead^a \cdot \omega_{\#}(\mu(P))$)=O($ne^{k+1}ar^kd^a$) par nBT,*
- *O($near^2 \cdot \omega_{\#}(\mu(P))$)=O($ne^{k+1}ar^{k+2}$) par nFC$_i$ ou nRFL.*

Nous pouvons constater de plus que les complexités en temps sont même meilleures que celles issues de l'algorithme dédié (proposé dans [Chmeiss and Jégou, 1997]). Par exemple, celui-ci calcule la microstructure d'un CSP binaire et énumère toutes les cliques maximales ou bien s'arrête dès qu'une n-clique est trouvée. Ainsi, la complexité en temps est en O($(nd)^{2(k-1)}(nd + n^2d^2)$) = O($(nd)^{2k}$). Néanmoins, nous pouvons noter que l'algorithme est défini pour des graphes CSG^k quelconques, alors que les (DR)-microstructures de CSP sont des graphes très particuliers.

Il semble, en termes de CSP, que les graphes CSG soient généralement moins restrictifs que les classes précédentes de graphes. Par exemple, il est possible d'avoir des graphes CSG possédant des n-cliques (resp. des e-cliques) pour toute valeur de n (resp. e), contrairement au cas des graphes planaires. En particulier, il existe des CSP cohérents possédant une (DR-)microstructure qui est un graphe CSG^k. C'est le cas notamment pour la classe CSG^0 qui contient exactement les CSP binaires cohérents de domaines monovalents (une valeur par domaine) ou les CSP n-aires cohérents avec exactement un tuple autorisé par relation. Néanmoins, les CSP qui ont une (DR-)microstructure qui est un graphe CSG^1 peuvent être cohérents ou non et il est facile de construire un CSP avec plusieurs solutions, ce qui correspond à une collection de cliques de taille n (cas binaire) ou e (cas n-aire). En outre, contrairement aux classes des sections précédentes, dans les graphes CSG^k (avec $k \geq 1$), il n'y a pas de restriction sur les valeurs de n, d, e, a ou r.

En pratique, les familles de benchmarks `Hanoi` et `Domino`, utilisées dans les compétitions CSP pour tester les solveurs, ont une microstructure triangulée après application de la cohérence d'arc. Donc, l'utilisation des algorithmes comme RFL ou MAC permet d'exploiter les classes traitables que nous avons présentées ici. Toutefois, cet ensemble de classes de CSP doit encore être étudié en détail pour évaluer son intérêt pratique.

4.4.4 $D(CL)$ vs quelques classes polynomiales

Les classes CL et DCL se basent toutes les deux sur le nombre de cliques maximales dans la microstructure. Le nombre de cliques maximales dépend toujours du nombre de solution d'un CSP. Dans le cas binaire, nous pouvons avoir un CSP P qui appartient à la fois à CL et à DCL vu que le nombre de cliques maximales sera polynomial dans la microstructure classique et dans la DR-microstructure. Cependant, ceci n'implique pas que $\mu(P)$ et $\mu_{DR}(P)$ appartiennent à la même classe de graphe. Autrement dit, nous pouvons avoir un CSP binaire ayant un nombre polynomial de cliques maximales et avec une microstructure triangulée et une DR-microstructure non-triangulée ou inversement (voir figure 3.13 et figure 3.14).

Maintenant, nous montrerons que cette classe polynomiale est différente des classes polynomiales présentées dans l'état de l'art.

4.4.4.1 Cas des CSP binaires

Dans cette section, nous comparons $(D)CL$ aux classes polynomiales relatives aux CSP binaires. Pour cela, nous utiliserons, comme pour le filtrage par cohérence, les notations suivantes :

Définition 4.7. *Étant données deux classes $\mathcal{C}_1$ et $\mathcal{C}_2$, nous disons que :*
- *$\mathcal{C}_1$ et $\mathcal{C}_2$ sont **incomparables** (notée $\mathcal{C}_1 \perp \mathcal{C}_2$) si aucune des deux relations n'est vraie.*
- *$\mathcal{C}_1$ et $\mathcal{C}_2$ sont **égales** (notée $\mathcal{C}_1 = \mathcal{C}_2$) si $\mathcal{C}_1 \subseteq \mathcal{C}_2$ et $\mathcal{C}_2 \subseteq \mathcal{C}_1$.*

Nous montrons maintenant que la classe des CSP ayant un nombre de cliques maximales est différente de certaines autres classes polynomiales connues à savoir BTP (broken-triangle property), RC (row-convex), RRM (renamable right monotone), $TREE$ (CSP arborescents) et ZOA (0-1-Tous).

Théorème 4.3. $(D)CL \cap BTP \neq \emptyset$ *et* $(D)CL \perp BTP$.
$(D)CL \cap RC \neq \emptyset$ *et* $(D)CL \perp RC$.
$(D)CL \cap RRM \neq \emptyset$ *et* $(D)CL \perp RRM$.
$(D)CL \cap TREE \neq \emptyset$ *et* $(D)CL \perp TREE$.
$(D)CL \cap ZOA \neq \emptyset$ *et* $(D)CL \perp ZOA$.

Preuve : Pour montrer que l'intersection entre $(D)CL$ et ces classes polynomiales est non vide, il suffit de prendre un CSP binaire monovalent avec trois variables et seulement deux contraintes (le CSP contient donc une seule contrainte universelle), ce CSP est BTP, RC, RRM, arborescent, ZOA et aussi $(D)CL$. Pour montrer que $(D)CL$ est différente des autres classes polynomiales, nous pouvons considérer un CSP avec n variables, e contraintes, d valeurs par domaines et ayant un graphe de microstructure complet, il est donc BTP, RC, RRM, arborescent ZOA mais non $(D)CL$. L'exemple de la figure 4.4(a) illustre le cas d'un CSP binaire $(D)CL$ (avec quatre cliques maximales) mais non ZOA (car la valeur 5 est compatible seulement avec deux valeurs de x_j) et 4.4(b) illustre le cas d'un CSP avec six cliques maximales (qui correspondent exactement aux arêtes) et qui ne satisfait ni BTP, ni RRM ni $TREE$ et ni RC. □

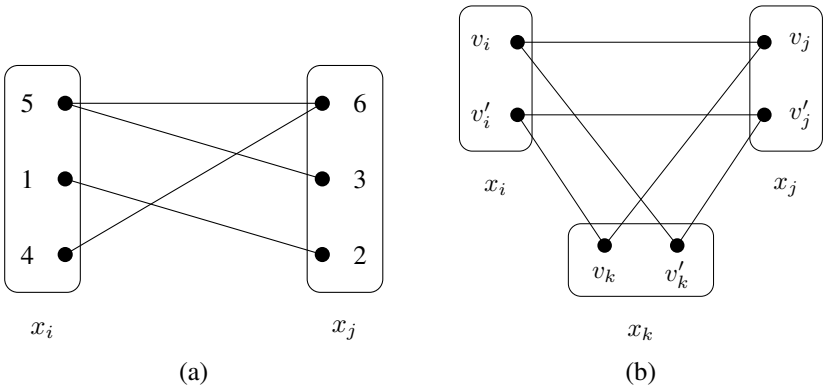

FIGURE 4.4 – (a) un exemple $(D)CL$ mais non ZOA. (b) un exemple $(D)CL$ mais il n'est ni BTP, ni RRM, ni $TREE$, ni CCM et ni RC.

Nous finissons cette partie par comparer $(D)CL$ à CCM (Chordal Complement Microstructure) et PM (Perfect Microstructure).

Théorème 4.4. $(D)CL \cap CCM \neq \emptyset$ *et* $(D)CL \perp CCM$.
$(D)CL \cap PM \neq \emptyset$ *et* $(D)CL \perp PM$.

Preuve : Pour montrer que l'intersection est non vide, nous considérons un CSP binaire P_1 avec deux variables x_1 et x_2 telles que $D(x_1) = \{a, a'\}$ et $D(x_2) = \{b, b'\}$ et une seule contrainte c_{12} telle que $R(c_{12}) = \{(a, b), (a', b'), (a', b)\}$. Ce CSP est dans CCM, PM et $(D)CL$. Pour montrer que CCM et $(D)CL$ sont différentes, nous considérons l'exemple de la figure 4.4 qui est $(D)CL$ mais non CCM. Si nous considérons un CSP P_2 avec n variables, e contraintes, d valeurs par domaines et ayant un graphe de microstructure complet, P_2 est CCM mais non $(D)CL$. Pour montrer que PM et $(D)CL$, nous pouvons considérer le CSP P_2 qui est PM mais non $(D)CL$. Sin on considère un CSP binaire monovalent et incohérent ayant 7 variables et dont le graphe de microstructure est un cycle de longueur 7, il n'est pas PM mais il est dans $(D)CL$. □

4.4.4.2 Cas des CSP non-binaires

Pour le cas non-binaire, nous montrons que DCL est différente de $IFUN$ (Incrementally Functional), MC (Max Closed) et BTW_k (Bounded Tree Width).

Théorème 4.5. $DCL \cap MC \neq \emptyset$ *et* $DCL \perp MC$.
$DCL \cap BTW_k \neq \emptyset$ *et* $DCL \perp BTW_k$.

Preuve : Pour montrer que l'intersection entre DCL et ces classes polynomiales est non vide, il suffit de prendre un CSP ternaire monovalent avec deux contraintes c_1 et c_2 (telles que $S(c_1) \cap S(c_2) \neq \emptyset$), ce CSP est IFUN, MC, BTW_k et aussi DCL. Pour montrer que DCL est différente des autres classes polynomiales, nous pouvons considérer un CSP acyclique avec n variables, e contraintes, d valeurs par domaines et telles que les contraintes autorisent toutes les combinaisons des valeurs possibles, il est donc MC et BTW_k mais non DCL. Si on considère un CSP monovalent avec un graphe de contraintes complet, alors ce CSP est DCL mais non BTW_k et si on considère l'exemple de la figure 2.16(a), il est DCL (deux cliques maximales) mais non MC. □

Théorème 4.6. $IFUN \subsetneq DCL$.

Preuve : Un CSP satisfaisant $IFUN$ a au plus une solution [Cohen et al., 2011], donc une seule clique maximale. Pour montrer que l'équivalence est fausse, nous considérons un CSP avec deux solutions, ce CSP n'est pas $IFUN$. □

4.5 Conclusion

Dans ce chapitre, nous avons présenté une analyse de la complexité temporelle des algorithmes génériques classiques de résolution de CSP dans une perspective nouvelle et surtout différente des études menées jusqu'à présent. Notre analyse exprime la complexité en termes de nombre de cliques maximales dans la (DR-)microstructure du CSP à résoudre. Cette analyse révèle que BT, FC et RFL visitent chaque clique maximale de la (DR-)microstructure au plus une fois. Pour MAC, une preuve différente qui convient à un arbre binaire de recherche devrait être réalisée.

A partir de cette analyse, nous déduisons des classes polynomiales de CSP qui peuvent être résolues par des algorithmes classiques en temps polynomial, *sans avoir à reconnaître que le CSP figure dans la classe*. Aussi, les résultats obtenus apportent un éclairage nouveau sur l'analyse de la

complexité des CSP. La première perspective de ce travail est constituée par l'étude de classes de graphes possédant un nombre polynomial de cliques maximales. L'étude de [Rosgen and Stewart, 2007] revêt ici un intérêt particulier car elle a permis de caractériser précisément ces classes de graphes sur la base de graphes d'intersection.

Une autre perspective importante consiste à mettre en évidence des liens qui pourraient exister entre notre analyse et les classes polynomiales obtenues par des approches différentes. Il devrait être assez clair notamment que nos classes sont orthogonales aux classes polynomiales basées sur la structure. Par exemple, il n'existe aucune raison pour qu'un CSP arborescent ne possède pas un nombre exponentiel de cliques. Plus généralement, les classes basées sur la structure nécessitent généralement des algorithmes dédiés, et les algorithmes de backtracking génériques sont de complexité polynomiales sur elles uniquement s'ils procèdent, par exemple, via l'exploitation d'un ordre d'affectation des variables spécifique. Par contre, pour les classes définies par un langage de contraintes, il serait possible que certaines soient capturées par notre analyse, tout comme c'est le cas pour le problème de satisfiabilité dans les travaux de [Rauzy, 1995]. Enfin, les classes hybrides sont beaucoup plus proches dans l'esprit de notre approche et l'étude en profondeur des liens entre ces classes et notre analyse constitue donc une perspective naturelle à court terme.

Enfin, une perspective importante porte sur une nouvelle analyse de complexité basée sur un nouveau paramètre et qui permettrait de capturer plus de classes polynomiales.

Troisième partie

BTP pour CSP d'arités quelconques et résultats expérimentaux

Chapitre 5

BTP : une étude pratique

5.1 Introduction

L'étude des *classes polynomiales* pour CSP constitue depuis longtemps un domaine de recherche important qui s'avère aujourd'hui très actif. Cependant, les travaux réalisés jusqu'à présent se sont révélés pour l'essentiel théoriques. En effet, ils se cantonnent en général à la définition de classes de CSP pour lesquels des algorithmes de résolution de complexité polynomiale, sont proposés. La plupart de ces classes se basent sur des propriétés très restrictives et elles s'avèrent donc difficilement exploitables en pratique, et ne sont donc pas exploités au sein de solveurs généraux. L'intérêt pratique des classes polynomiales est ainsi très limité.

Dans le chapitre 4, nous avons mentionné qu'en effectuant un filtrage par AC, certaines familles de benchmarks tels que `Domino` et `Hanoi` appartiennent à des classes polynomiales connues à savoir « la classe des CSP ayant une microstructure triangulée ». D'une façon un peu plus générale, les algorithmes de filtrages qui serviront comme moyen de transformation de CSP, peuvent aider à la recherche des classes polynomiales car ils suppriment des arêtes incohérentes et ne modifient pas l'ensemble de solutions. En plus, certains algorithmes de filtrages sont de complexité polynomiale.

Dans ce chapitre, nous aborderons la question des classes polynomiales pour CSP d'un point de vue plus général et nous montrerons la présence des classes polynomiales dans les benchmarks de la compétition CP 2008. Cette approche consiste à mettre en évidence la présence de ces classes en utilisant les techniques de *filtrage* comme celles qui sont généralement exploitées dans les solveurs à chaque étape de pré-traitement. En outre, nous croyons que cela permet de prouver l'appartenance de certaines classes simplifiées à des classes polynomiales. Ces classes seront appelées *classes polynomiales cachées*.

Pour montrer l'intérêt de cette approche, nous allons focaliser notre étude sur l'analyse de la classe BTP : cette classe dispose de plusieurs propriétés intéressantes dans ce contexte. Tout d'abord, BTP capte plusieurs autres classes polynomiales définies précédemment dans la littérature. Ensuite, c'est une classe qui peut se présenter plus facilement dans les benchmarks vu qu'elle est moins restrictive que certaines autres classes. En outre, elle possède des propriétés algorithmiques qui devraient faciliter sa manipulation implicite par les solveurs. D'une part, un solveur CSP basé sur des algorithmes tels que MAC ou RFL peut résoudre n'importe quel CSP satisfaisant BTP directement et efficacement sans l'aide d'algorithmes spécifiques. D'autre part, l'intérêt que recèle ici BTP, outre ses propriétés théoriques en terme d'inclusion d'autres classes, est que cette classe est *close* par rapport à la suppression de valeurs et/ou de tuples, c'est-à-dire, tout CSP satisfaisant BTP le restera après application de filtrages par cohérence. D'un point de vue pratique, nous montrons que des *benchmarks*, généralement exploités pour les comparaisons de solveurs, appartiennent à

BTP ou à une de ses classes cachées.

Dans la deuxième partie, nous montrerons comment BTP pourrait être utilisée pour éliminer des valeurs par fusion. En effet, la réduction des domaines peut se réaliser soit par les techniques de filtrage soit par l'élimination des valeurs par substitution ou interchangeabilité. Si les techniques de filtrages permettent de supprimer les valeurs ne pouvant pas prendre part à une solution, la deuxième méthode consiste à supprimer des valeurs, cohérentes et incohérentes, de domaines de variables, tout en préservant la satisfiabilité du CSP, afin d'améliorer la complexité de résolution qui dépend, entre autres, de la taille des domaines.

Ce chapitre sera organisé comme suit. Dans la section 5.2.1, nous présenterons notre cadre formel pour les classes cachées et nous l'illustrerons en considérant la classe BTP dans la section 5.2.2. Dans la deuxième section, nous commencerons par rappeler certaines méthodes d'élimination des valeurs. Ensuite, nous proposerons une nouvelle approche, pour les CSP binaires, basée sur BTP pour fusionner les valeurs n'apparaissant dans aucun triangle cassé tout en montrant l'efficacité de cette méthode d'un point de vue pratique. Par la suite, nous essaierons de généraliser cette approche aux CSP d'arité quelconque. Avant de conclure, nous parlerons de la polynomialité de cette généralisation et de la complétude de la recherche d'un ordre sur les variables garantissant cette polynomialité. La première partie de ce chapitre a été publiée dans [El Mouelhi et al., 2014] et la deuxième dans [Cooper et al., 2014].

5.2 BTP et ses classes cachées

Dans cette section, nous présenterons un nouveau cadre théorique pour l'étude pratique des classes polynomiales. Par la suite, nous montrerons l'utilité de cette étude en l'appliquant sur la classe BTP et nous montrerons que ceci permettra d'avoir des résultats intéressants.

5.2.1 Classes polynomiales cachées

L'étude des classes polynomiales est de plus en plus développée en programmation par contraintes surtout d'un point de vue théorique. Mais, il est rare que les classes polynomiales soient explicitement exploitées en pratique. Dans certains cas, leur traitement requiert des algorithmes ad-hoc et est trop coûteux en terme de temps. Ou, même si elles peuvent être facilement exploitées (par exemple, en temps linéaire), elles n'apparaissent pas vraiment dans les problèmes réels, et donc, il n'est pas facile de les exploiter par des solveurs CSP classiques de l'état de l'art. Ici, nous aborderons cette question grâce à l'utilisation de filtrages qui participent à l'efficacité pratique des solveurs comme la cohérence d'arc AC ou des filtrages plus puissants. En effet, certaines classes polynomiales peuvent parfois être « cachées » alors qu'elles pourraient être « découvertes » par application de filtrages. À cette fin, nous proposerons un cadre général qui peut être utilisé pour n'importe quelle classe polynomiale et tout type de *transformation* de CSP, comme le filtrage par exemple.

Définition 5.1. *Étant donné un CSP $P = (X, D, C)$, T est dite une **transformation** de P si $T(P) = (T_{var}(X), T_{dom}(D), T_{cons}(C))$ vérifie :*

- $T_{var}(X) \subseteq X$
- $T_{dom}(D) = \{T_{dom}(D(x')) : x' \in T_{var}(X)$ *et* $T_{dom}(D(x')) \subseteq D(x)\}$
- $\forall c_i \in C, T_{cons}(c_i)$ *vérifie :*
  - $T_{cons}(S(c_i)) = S(c_i) \backslash \{x \in X : x \notin T_{var}(X)\}$ *et*
  - $T_{cons}(R(c_i)) \subseteq R(c_i)[T_{cons}(S(c_i))]$.
- $\forall c' \in T_{cons}(C)$ *telle que* $c' \notin C$, $T_{cons}(R(c')) \subseteq \prod_{x' \in S(c')} T_{dom}(D(x'))$.

Notons que si une contrainte c' de $T_{cons}(C)$ n'appartient pas à C, sa relation est une sous-relation de la relation universelle associée à la portée $S(c')$ qui est définie implicitement dans P.

Comme transformations classiques de CSP, nous trouvons généralement :
– la transformation par suppression de variables,
– l'élimination de valeurs,
– l'ajout de contraintes et
– la suppression de tuples dans les relations de compatibilité de contraintes.

Le filtrage, par exemple, ne correspond pas seulement à la suppression de valeurs, mais aussi à l'ajout de contraintes avec suppression de tuples comme la cohérence de chemin dans certains cas (voir chapitre 2, figure 2.9). D'autre part, l'instanciation de variables peut être considérée comme un cas particulier de filtrage vu qu'une affectation $x_i = v_i$ peut être considérée comme un filtrage donnant un nouveau domaine $T_{dom}(D(x_i)) = \{v_i\}$.

Nous pouvons maintenant définir la notion de transformation de CSP pour un cas plus général et nous allons considérer seulement les transformations définies par la simplification de CSP. Mais nous pourrions aussi définir les transformations basées sur l'ajout de nouvelles variables et de nouvelles valeurs dans les domaines, ainsi que l'ajout de tuples dans les relations existantes ou supprimées.

Enfin, nous signalons que certaines propriétés classiques des transformations, comme la préservation de l'ensemble des solutions ne sont pas nécessaires ici.

Définition 5.2. *Étant donnés une transformation T et un ensemble (appelé aussi une classe) de CSP $\mathcal{P}$, nous avons $T(\mathcal{P}) = \{T(P) : P \in \mathcal{P}\}$.*

Donc, étant données une transformation T et une classe de CSP $\mathcal{C}$, il est possible qu'un CSP P, n'appartenant pas à $\mathcal{C}$, apparaisse dans $\mathcal{C}$ après l'avoir transformée par T. L'appartenance de tels CSP à $\mathcal{C}$ sera donc mise en évidence par T :

Définition 5.3. *Étant données une transformation T et une classe de CSP $\mathcal{C}$, la classe de CSP mise en évidence par T pour $\mathcal{C}$ est $\mathcal{C}^T = \{P : T(P) \in \mathcal{C}\}$.*

Maintenant, nous pouvons introduire la notion de *classe cachée* :

Définition 5.4. *Étant donnés un ensemble $\mathcal{P}$ de CSP, une transformation T et une classe $\mathcal{C}$, $\mathcal{P}$ est une **classe cachée** de $\mathcal{C}$ pour T, si $t(\mathcal{P}) \subseteq \mathcal{C}$, si c'est $\mathcal{P} \subseteq \mathcal{C}^T$. $\mathcal{P}$ est appelée* classe polynomiale cachée *de $\mathcal{C}$ pour T si :*
– *T préserve la cohérence,*
– *T peut être réalisée en temps polynomial,*
– *$\mathcal{C}$ est polynomiale.*

Dans le cas d'un CSP $P \in \mathcal{P}$ tel que $\mathcal{P}$ est une classe cachée de $\mathcal{C}$ pour une transformation t, et si $\mathcal{C}$ est une classe traitable, il est donc suffisant d'appliquer le filtrage T pour avoir une version simplifiée de P qui est traitable. En supposant que le coût de filtrage T est polynomial, $\mathcal{P}$ sera aussi une classe traitable.

Ce type d'approche est particulièrement adapté aux solveurs CSP de l'état de l'art, car ils utilisent généralement des filtrages avant et pendant la recherche.

Proposition 5.1. *Pour toutes classes $\mathcal{C}_1$ et $\mathcal{C}_2$ telles que $\mathcal{C}_1 \subseteq \mathcal{C}_2$, et pour toute transformation T, nous avons $\mathcal{C}_1^T \subseteq \mathcal{C}_2^T$.*

Preuve : Soit $P \in \mathcal{C}_1^T$. Par définition, $T(P) \in \mathcal{C}_1$. Puisque $\mathcal{C}_1 \subseteq \mathcal{C}_2$, nous avons $T(P) \in \mathcal{C}_2$, et donc $P \in \mathcal{C}_2^T$. Donc $\mathcal{C}_1^T \subseteq \mathcal{C}_2^T$, et nous aurons $\mathcal{C}_1^T \subseteq \mathcal{C}_2^T$. □

Notons que si $\mathcal{C}_1 \subsetneq \mathcal{C}_2$, nous n'aurons pas forcément $\mathcal{C}_1^T \subset \mathcal{C}_2^T$. Par exemple, si T est une transformation de CSP binaires qui calcule le CSP minimal de tout CSP binaire, et si $\mathcal{C}_1$ est la classe

des CSP binaires minimaux alors que $\mathcal{C}_2$ est la classe des CSP binaires arc-cohérentes (y compris les CSP binaires avec des domaines vides), nous avons $\mathcal{C}_1 \subsetneq \mathcal{C}_2$ alors que $\mathcal{C}_1^T = \mathcal{C}_2^T$.

Étudier les transformations permet de détecter la classe des CSP qui deviennent traitables après transformations. Pour convenir à la mise en œuvre dans les solveurs qui exécutent une séquence de transformations (par exemple le filtrage par AC utilisé par MAC ou RFL), il serait souhaitable d'utiliser des transformations qui préservent les propriétés caractérisant la classe polynomiale considérée. En d'autres termes, nous voulons exclure les transformations T telles que pour une classe polynomiale $\mathcal{C}$ et un CSP $P \in \mathcal{C}$, nous avons $T(P) \notin \mathcal{C}$.

Cette question a été traitée dans [Cooper et al., 2010] à l'aide de la définition des classes qui sont *closes* par rapport à la restriction des domaines (précisément fermées même après réduction des domaines). Nous généralisons cette notion à tout type de transformation.

Définition 5.5. *Une classe $\mathcal{C}$ de CSP est dite **close** par rapport à une transformation T si elle est fermée pour T, c'est-à-dire $\forall P \in \mathcal{C}, T(P) \in \mathcal{C}$ (ou $T(\mathcal{C}) \subseteq \mathcal{C}$). Une transformation T est dite close si elle définit une classe de CSP close.*

Certaines propriétés peuvent être directement déduites de cette notion :

Proposition 5.2. *Toute classe $\mathcal{C}$ de CSP est close par rapport à l'identité (notée Id), à savoir $Id(\mathcal{C}) \subseteq \mathcal{C}$.*

Dans [Cooper et al., 2010], ce concept a été utilisé dans le cadre de filtrage de domaines puisque BTP est close par rapport à tout filtrage de domaines. Au-delà du filtrage, ce concept pourra aussi être exploité en tenant compte des suppressions de variables liées aux propriétés structurelles des réseaux de contraintes. Par exemple :

Proposition 5.3. *La classe $TREE$ est close par rapport à toute transformation de CSP qui se limite à supprimer des variables (sommets du graphe de contraintes).*

Proposition 5.4. *La classe ZOA est close par rapport à la cohérence d'arc.*

Proposition 5.5. *La classe RC n'est pas close par rapport à la cohérence de chemin.*

D'autres propriétés peuvent être déduites en utilisant la comparaison entre les classes modifiées par le moyen de transformations.

Proposition 5.6. *Si $\mathcal{C}$ est close par rapport à une transformation T, alors $\mathcal{C} \subseteq \mathcal{C}^T$.*

Preuve : Soit $P \in \mathcal{C}$. Comme $\mathcal{C}$ est close pour T, $T(P) \in \mathcal{C}$, et donc $P \in \mathcal{C}^T$. □

Corollaire 5.1. *Si $\mathcal{C}$ est close par rapport à une transformation T, alors $\mathcal{C}^{Id} \subseteq \mathcal{C}^T$.*

Preuve : Puisque $\mathcal{C} \subseteq \mathcal{C}^T$ et $\mathcal{C}^{Id} = \mathcal{C}$, nous avons $\mathcal{C}^{Id} \subseteq \mathcal{C}^T$. □

En considérant le filtrage comme moyen de transformations, nous pouvons déduire des propriétés plus spécifiques comme c'est indiqué dans la proposition suivante :

Proposition 5.7. *Soient T_1 et T_2 deux transformations qui sont des filtrages telles que $T_1 \leq T_2$. Pour toute classe $\mathcal{C}$ qui est close par rapport à T_1 et T_2, nous avons $\mathcal{C}^{T_1} \subseteq \mathcal{C}^{T_2}$.*

Preuve : soit $P \in \mathcal{C}^{T_1}$, nous montrons que $P \in \mathcal{C}^{T_2}$. Puisque $P \in \mathcal{C}^{T_1}$, nous avons $T_1(P) \in \mathcal{C}$. En plus, puisque $T_1 \leq T_2$, $T_2(T_1(P)) = T_2(P)$. Donc, comme $T_1(P) \in \mathcal{C}$ et comme $\mathcal{C}$ est close par rapport à T_2, $T_2(T_1(P)) \in \mathcal{C}$ et $T_2(P) = T_2(T_1(P)) \in \mathcal{C}$. Par conséquent $P \in \mathcal{C}^{T_2}$. □

Nous pouvons noter que cette propriété n'est pas vérifiée pour les relations strictes. En effet, nous pouvons définir des classes de CSP $\mathcal{C}$ qui sont closes par rapport à T_1 et T_2 telles que $T_1 < T_2$, alors que $\mathcal{C}^{T_1} \subsetneq \mathcal{C}^{T_2}$ est fausse. Par exemple, si $\mathcal{C}$ est la classe contenant tous les CSP binaires, et si $T_1 = AC$ et $T_2 = SAC$ ($T_1 < T_2$), d'après la propriété 5.6, nous avons $\mathcal{C}^{T_1} = \mathcal{C} = \mathcal{C}^{T_2}$.

Bien que certaines propriétés générales ne puissent pas tenir, nous pouvons donner quelques exemples de ces relations avec des transformations et des classes différentes :

Exemple 5.1 ($\mathcal{C}_1^{T_1} \subsetneq \mathcal{C}_2^{T_2}$)**.** *Considérons $\mathcal{C}_1 = \mathcal{C}_2 = BTP$ qui est close par rapport à $T_1 = Id$ et $T_2 = AC$. Nous savons que $T_1 < T_2$ et comme nous avons $\mathcal{C}_1^{t_1} \subsetneq \mathcal{C}_2^{T_2}$ alors $BTP^{Id} \subsetneq BTP^{AC}$.*

Exemple 5.2 ($\mathcal{C}_1^{T} \subsetneq \mathcal{C}_2^{T}$)**.** *Considérant $\mathcal{C}_1 = TREE$ et $\mathcal{C}_2 = BTP$. De [Cooper et al., 2010], nous savons que la classe $TREE \subsetneq BTP$. En plus, $TREE$ et BTP sont closes par rapport à $T = AC$. Nous avons $\mathcal{C}_1^{T} \subsetneq \mathcal{C}_2^{T}$, c'est-à-dire $TREE^{AC} \subsetneq BTP^{AC}$.*

Exemple 5.3 ($\mathcal{C}_1^{T_1} = \mathcal{C}_2^{T_2}$)**.** *Ici nous considérons n'importe quelle transformation de CSP $t_1 = DCC$ qui supprime les sommets d'un couple-cycle d'un CSP binaire et $T_2 = Id$. $\mathcal{C}_1 = TREE$ alors que $\mathcal{C}_2 = CSP$, c'est-à-dire, l'ensemble de tous les CSP binaires possibles. Nous avons $\mathcal{C}_1^{T_1} \subseteq \mathcal{C}_2^{T_2}$, c'est-à-dire $TREE^{DCC} \subseteq CSP^{Id}$. Mais nous avons aussi $\mathcal{C}_2^{T_2} \subseteq \mathcal{C}_1^{T_1}$ donc $CSP^{Id} \subseteq TREE^{DCC}$. Bien sûr, si CSP^{Id} est bien une classe cachée, elle n'est pas une classe polynomiale cachée. Pareillement, $TREE^{DCC}$ est une classe cachée mais elle n'est pas une classe polynomiale cachée car DCC ne préserve pas la cohérence.*

L'utilisation de transformations comme DCC est proche de la notion de backdoor [Williams et al., 2003]. Un backdoor est un ensemble de variables défini par rapport à un algorithme particulier de sorte qu'une fois les variables du backdoor affectées, le problème devient facile à résoudre par cet algorithme. Par exemple, une fois les variables d'un coupe-cycle affectées, le sous-problème induit peut être résolu en temps linéaire par un algorithme comme MAC ou RFL [Dechter and Pearl, 1987a]. Cette approche est une première façon d'exploiter les classes traitables qui sont cachées ou non.

Dans la prochaine section, nous allons envisager une nouvelle façon d'exploiter les classes cachées tout en exploitant le filtrage comme moyen de transformation. Bien qu'il soit possible d'étudier plusieurs classes polynomiales, notre étude est basée ici sur la classe BTP car elle est considérée comme une classe polynomiale importante pour les problèmes CSP.

5.2.2 Le cas de BTP

Avant d'appliquer les transformations sur la classe BTP, nous montrerons l'intérêt de notre étude en suivant l'évolution des CSP transformés par filtrage. Pour ce faire, nous introduirons les notions suivantes qui serviront à tester l'appartenance de certains benchmarks à BTP ou à ces classes cachées.

Définition 5.6 (triangle cassé, variable cassée)**.** *Nous disons que nous disposons d'un **triangle cassé** sur une paire de valeurs $v'_k, v''_k \in D(x_k)$ s'il existe deux valeurs $v_i \in D(x_i)$, $v_j \in D(x_j)$ de deux variables distinctes $x_i, x_j \in X \setminus \{x_k\}$ telles que*

- $(v_i, v_j) \in R(c_{ij})$,
- $(v_i, v'_k) \in R(c_{ik})$,
- $(v_j, v''_k) \in R(c_{jk})$,
- $(v_i, v''_k) \notin R(c_{ik})$ *et*
- $(v_j, v'_k) \notin R(c_{jk})$.

*La variable x_k sera dite **variable cassée** et le triangle cassé sera noté (v'_k, v_i, v_j, v''_k).*

Définition 5.7 (triplet cassé)**.** *Si pour un triplet de variables x_i, x_j et x_k, il existe un triangle cassé sur chaque variable, alors ce triplet est dit **cassé**.*

L'exemple de la figure 5.1 (a) présente le cas d'un triangle cassé (v'_k, v_i, v_j, v''_k) sur la variable x_k. L'exemple de la figure 5.1 (b) présente le cas d'un triplet cassé. Pour la variable x_i, le triangle cassé est (v_i, v_j, v''_k, v'_i). Pour la variable x_j, le triangle cassé est (v_j, v_i, v'_k, v'_j). Pour la variable x_k, le triangle cassé est (v'_k, v_i, v_j, v''_k).

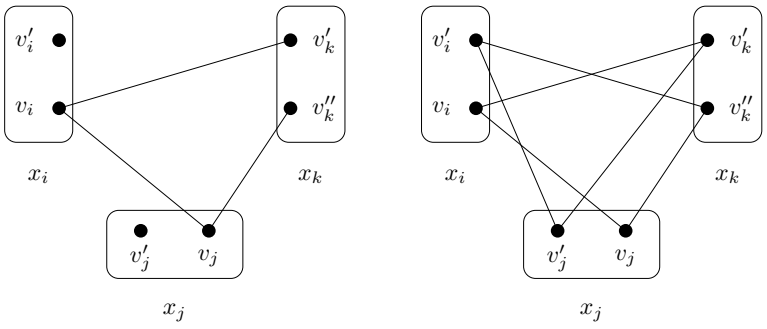

FIGURE 5.1 – (a) un triangle cassé, (b) un triplet cassé.

5.2.2.1 Résultats théoriques

Nous commençons tout d'abord par rappeler les relations entre BTP et certaines autres classes polynomiales en utilisant les notations que nous avons proposées dans la section précédente :

Théorème 5.1 ([Cooper et al., 2010])**.** *(i)* $RRM \subsetneq BTP$.
(ii) $TREE \subsetneq BTP$.
(iii) *DUAL-*$TREE \subsetneq BTP$.

Ensuite, nous nous intéresserons à la relation entre BTP et certaines de ses classes cachées et nous concentrerons notre étude sur les transformations basées sur le filtrage.

Théorème 5.2. *(i)* $BTP \subsetneq BTP^{AC} \subsetneq BTP^{PIC} \subsetneq BTP^{maxRPC} \subsetneq BTP^{SAC}$.
(ii) $BTP^{maxRPC} \subsetneq BTP^{NIC}$.
(iii) $BTP^{SAC} \perp BTP^{NIC}$.

Preuve : La propriété 2.68 est notamment vraie pour AC. Donc on a $BTP \subsetneq BTP^{AC}$ d'après le corollaire 5.1. Si nous considérons le benchmark $L_{2,2}$ qui représente le problème de nombres de Langford [1], nous pouvons noter qu'elle n'est pas BTP mais arc-incohérente et donc elle appartient à BTP^{AC}. Donc, $BTP \subsetneq BTP^{AC}$.

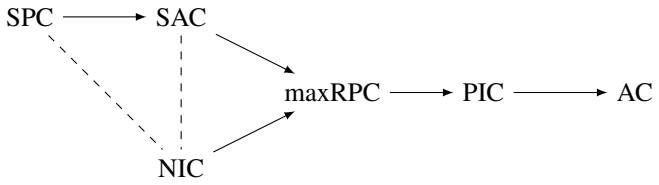

FIGURE 5.2 – Relation entre les cohérences utilisées comme transformation pour identifier les classes cachées de BTP.

D'après les propriétés 2.68 et 5.7 et la relation entre les cohérences illustrées dans la figure 5.2, nous avons $BTP^{AC} \subsetneq BTP^{PIC} \subsetneq BTP^{maxRPC} \subsetneq BTP^{SAC}$ et $BTP^{maxRPC} \subsetneq BTP^{NIC}$.

1. Le benchmark $L_{k,m}$ du problème de nombres de Langford (pour plus de détails, voir problème 024 de CSPLib [csp]) consiste à arranger k ensembles de nombres $\{1, \ldots, m\}$ telle que chaque occurrence du nombre i apparaît i fois après l'occurrence précédente.

Si on considère le problème des 4 reines (appartenant à une famille de benchmarks dite queens), ce benchmark appartient à BTP^{PIC} mais pas à BTP^{AC}. Donc $BTP^{AC} \subsetneq BTP^{PIC}$. Pareillement si nous considérons le benchmark $L_{3,4}$ (respectivement $L_{3,5}$ et $L_{2,14}$) pour montrer que $BTP^{PIC} \subsetneq BTP^{maxRPC}$ (resp. $BTP^{maxRPC} \subsetneq BTP^{SAC}$ et $BTP^{maxRPC} \subsetneq BTP^{NIC}$).

$L_{2,5}$ est dans BTP^{NIC} mais pas dans BTP^{SAC}. Réciproquement, la figure 3(e) de [Debruyne and Bessière, 2001] présente un CSP appartenant à BTP^{SAC} mais pas à BTP^{NIC}. Donc $BTP^{SAC} \perp BTP^{NIC}$. □

Nous notons, comme mentionné dans la preuve précédente, qu'il existe un lien entre BTP et ses classes cachées grâce à la relation qui existe entre les cohérences illustrée dans la figure 5.2.

Comme rappelé dans la figure 5.2, nous avons $SAC < SPC$. Malheureusement, ce résultat ne peut pas être exploité ici car BTP n'est pas close par rapport à SPC. En effet, l'application de PC peut supprimer des tuples dans les relations, c'est-à-dire des paires de valeurs $(v_i, v_j) \in R(c_{ij})$ qui n'ont pas de support dans la variable x_k. Ce faisant, le CSP peut ne plus satisfaire la propriété après application d'un tel filtrage. Le théorème suivant formalise cette assertion :

Théorème 5.3. *BTP n'est pas close par rapport à PC ni SPC.*

Preuve : Nous pouvons prouver ce résultat en nous basant sur un contre-exemple de 4 variables, x_1, x_2, x_3 et x_4. La variable x_3 doit être la dernière dans l'ordre pour que cet exemple satisfasse BTP (au moins après x_1 et x_2). La figure 5.3 montre des parties indépendantes de sa microstructure. Premièrement, dans la figure 5.3 (a), nous avons deux triangles cassés sur x_1 et x_2. Pour cela, nous avons besoin de huit valeurs (huit sommets dans le graphe de microstructure) et six tuples binaires (six arêtes). Deuxièmement, dans la figure 5.3 (b), nous avons quatre valeurs, et cinq tuples binaires (cinq arêtes). Les arêtes en pointillés correspondent aux arêtes nécessaires pour la satisfaction de BTP. En considérant l'ordre $x_4 < x_1 < x_2 < x_3$ sur les variables, il est facile de voir que ce CSP (partielle) satisfait BTP.

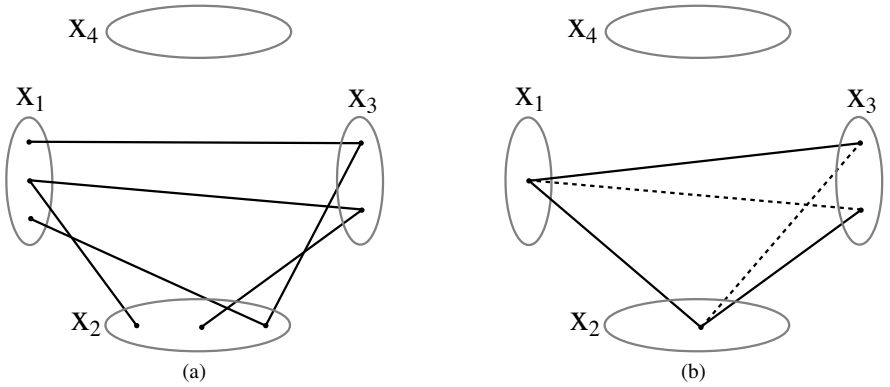

FIGURE 5.3 – (a) deux triangles cassés sur x_1 et x_2, (b) un motif BTP sur x_3 grâce aux deux arêtes en pointillés.

Par contre, un filtrage par PC va supprimer tous les tuples puisqu'aucun tuple ne possède un support dans le domaine de x_4, et de plus, BTP sera satisfaite après filtrage. Donc, nous ajoutons des valeurs et des tuples de façon à ce qu'aucune arête de la microstructure ne soit supprimée après filtrage par PC à l'exception des arêtes en pointillés. De cette façon, les deux triangles cassés de la figure 5.3 (a) ne seront pas supprimés et un nouveau triangle cassé sur x_3 va apparaitre dans la figure 5.3 (b). Par conséquent, il n'y aura pas d'ordre sur les variables x_1, x_2 et x_3 permettant la satisfaction de BTP. Donc, pour chaque arête (sauf celles en pointillés), nous ajoutons deux

valeurs, une dans chaque domaine de deux autres variables. Par exemple, pour l'arête de x_1 à x_2, nous ajoutons une valeur dans le domaine de x_3 et une valeur dans le domaine de x_4.

Finalement, nous ajoutons 5 arêtes afin de connecter ces quatre valeurs et dans le but d'avoir une 4-clique (avec six arêtes) puisqu'elles satisfont mutuellement PC. En procédant ainsi, nous avons la garantie qu'aucune de ces six arêtes ne sera supprimée par application du filtrage par PC. Donc, pour les 9 arêtes qui ne sont pas en pointillés, nous ajoutons 9×2 sommets et 9×5 arêtes.

Globalement, nous avons une microstructure avec $12+(9\times2) = 30$ sommets et $11+(9\times5) = 56$ arêtes. Ce CSP satisfait BTP avant filtrage. En effet, il est suffisant de considérer l'ordre $x_4 < x_1 < x_2 < x_3$ dans lequel aucun triangle cassé n'apparaît. Par construction, un filtrage par PC va supprimer seulement les deux arêtes en pointillés. Donc, BTP ne sera plus satisfaite puisque la suppression de ces arêtes engendrera l'apparition d'un triangle cassé sur x_3 par rapport aux variables x_1 et x_2 alors que les deux triangles cassés sur x_1 et x_2 sont préservés. D'où, BTP n'est pas close par rapport à PC. Comme le CSP, que nous avons construit, est aussi arc-cohérent, il est donc fortement chemin-cohérente. Donc BTP n'est pas close par rapport à SPC. □

Comme BTP n'est pas close par rapport à PC, ni SPC, nous obtenons le théorème suivant :

Théorème 5.4. *(i)* $BTP^{Id} \perp BTP^{(S)PC}$.

(ii) Pour tout filtrage $\phi \in \{AC, PIC, maxRPC, SAC, NIC\}$, $BTP^{\Phi} \perp BTP^{(S)PC}$.

(iii) $BTP^{PC} \subseteq BTP^{SPC}$.

Preuve : Le CSP construit dans la preuve précédente permet d'établir que $BTP^{Id} \not\subseteq BTP^{(S)PC}$. Elle permet également de montrer que, pour tout filtrage $\phi \in \{AC, PIC, maxRPC, SAC, NIC\}$, nous avons $BTP^{\Phi} \not\subseteq BTP^{(S)PC}$. Réciproquement, le benchmark $L_{2,3}$ appartient à BTP^{PC} et BTP^{SPC} grâce à la suppression des arêtes associées à certains tuples. Donc, elle n'appartient pas à BTP^{Id}, ni à BTP^{Φ} pour tout filtrage $\phi \in \{AC, PIC, maxRPC, SAC, NIC\}$. D'où nous avons $BTP^{Id} \perp BTP^{(S)PC}$ et $BTP^{\Phi} \perp BTP^{(S)PC}$. Maintenant nous prouvons que $BTP^{PC} \subseteq BTP^{SPC}$. Pour cela, nous considérons un CSP P appartenant BTP^{PC}. Par définition, $PC(P) \in BTP$. Il est bien connu que $SPC(P) = AC(PC(P))$ [Lecoutre, 2009], et comme BTP est close par rapport à AC, $SPC(P) \in BTP$. Par conséquent $BTP^{PC} \subseteq BTP^{SPC}$. □

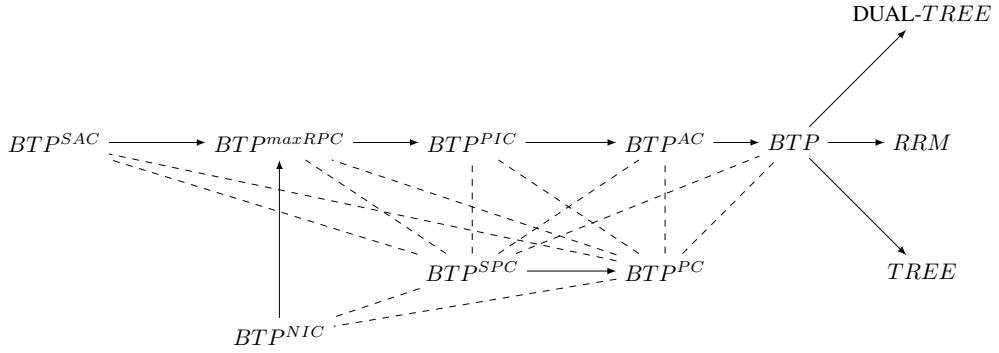

FIGURE 5.4 – Relation entre classes cachées de BTP.

Ces résultats sont résumés dans la figure 5.4 où un arc de $\mathcal{C}_2$ à $\mathcal{C}_1$ (resp. une ligne en pointillés entre $\mathcal{C}_1$ et $\mathcal{C}_2$) signifie que $\mathcal{C}_1 \subseteq \mathcal{C}_2$ (resp. $\mathcal{C}_1 \perp \mathcal{C}_2$).

Enfin, nous considérons la résolution des CSP satisfaisant BTP par des algorithmes classiques comme MAC ou RFL. Le théorème 7.6 de [Cooper et al., 2010] indique que MAC résout un CSP satisfaisant BTP en temps polynomial. Ceci est valable aussi pour RFL, car on a besoin d'appliquer seulement AC à chaque étape de la recherche. Ce résultat peut également être étendu à BTP^{AC}.

Théorème 5.5. *MAC (resp. RFL) résout tout CSP appartenant à BTP^{AC} en temps polynomial sans avoir besoin d'appliquer AC avant la résolution* [2].

Preuve : Nous considérons un CSP P appartenant à BTP^{AC}. Nous pouvons remarquer que l'affectation d'une valeur v à une variable x peut être considérée comme l'élimination de toutes les valeurs sauf v du domaine de x et qu'après suppression des valeurs de P et l'application de AC, le CSP qui en résulte est nécessairement BTP.

Lors de la résolution du CSP P, MAC choisit une première variable x, il lui affecte une valeur v et ensuite il applique AC. À cette étape, si aucun domaine n'est vide, nous savons qu'une solution existe avec l'affectation actuelle et MAC peut la trouver en temps polynomial comme décrit dans la preuve du théorème 7.6 de [Cooper et al., 2010]. Si un domaine devient vide, MAC considère la décision négative $x \neq v$ et applique de nouveau la cohérence d'arc. S'il n'y a pas de domaine vide, MAC peut trouver la solution en temps polynomial comme auparavant. Sinon, il n'existe pas de solution et la recherche s'arrête. Donc, MAC résout P en temps polynomial.

Nous raisonnons d'une façon similaire pour RFL, sauf qu'ici nous avons à considérer, dans le pire des cas, toutes les valeurs de la première variable choisie. □

5.2.2.2 BTP dans les benchmarks

Maintenant, nous nous demandons si certains CSP, souvent exploités comme benchmarks pour les comparaisons de solveurs, appartiennent à la classe BTP ou à une de ses classes cachées. Dans cette direction, nous considérons 2681 benchmarks binaires de la compétition CSP 2008 [3] et quelques méthodes de filtrages classiques, à savoir AC, PIC, $maxRPC$, SAC, NIC et SPC. Pour chaque benchmark, on vérifie si le CSP d'origine et les CSP obtenus par application du filtrage considéré appartiennent à BTP. Pour cela, il faut bien évidemment reconnaître l'existence de CSP satisfaisant BTP, mais plutôt que d'utiliser l'algorithme décrit dans [Cooper et al., 2010], nous avons utilisé une approche plus simple à mettre en œuvre et qui est fournie par l'algorithme 3.

Algorithme 3 : Check BTP ($P = (X, D, C)$) : Booléen

```
1  P^O ← (X^O, D^O, C^O)
2  X^O ← {x_i^O : ∀x_i ∈ X}
3  D^O ← {D(x_i^O) : x_i^O ∈ X^O et D(x_i^O) = {1, 2, ..., n}}
4  test ← vrai
5  pour i, j et k telles que i ≠ j i ≠ k, j ≠ k et test = vrai faire
6      si il existe un triangle cassé sur x_k alors
7          ajouter la contrainte x_k^O < max(x_i^O, x_j^O)
8          si il existe deux contraintes x_i^O < max(x_j^O, x_k^O) et x_j^O < max(x_i^O, x_k^O) alors
9              test ← faux
10 si test = faux alors
11     retourner faux
12 sinon
13     retourner Résoudre (P^O = (X^O, D^O, C^O) )
```

Pour chaque triplet de variables, notre approche consiste à tester s'il s'agit d'un triplet cassé, si c'est le cas l'algorithme s'arrête en nous indiquant que le CSP traité ne satisfait pas BTP. Dans le

2. Nous supposons que MAC n'applique pas AC en pré-traitement et qu'il l'applique seulement après chaque instanciation.
3. Voir http ://www.cril.univ-artois.fr/CPAI08 pour plus de détails.

cas contraire, nous nous intéressons à la détection d'un seul triangle cassé pour chaque variable d'un triplet. Au cours de la recherche, nous considérons un CSP ternaire (toutes les contraintes sont d'arité trois) $P^O = (X^O, D^O, C^O)$ qui aura une solution seulement s'il existe un ordre sur les variables pour lequel P satisfait BTP.

Pour construire P^O, nous associons une variable x_i^O ($\in X^O$) à chaque variable x_i du CSP traité. Le domaine de chaque variable x_i^O contient des valeurs de 1 à n (le nombre de variables de CSP P) qui indique la position de x_i dans l'ordre. Une contrainte sera créée chaque fois qu'un triangle cassé est détecté sur une variable (si dans un triplet de variable x_i, x_j et x_k, un triangle cassé est détecté sur x_k, on aura une contrainte de type $x_k^O < max(x_i^O, x_j^O)$ qui se rajoute au problème P^O). Cet algorithme teste tous les triplets. S'il détecte un triplet cassé, il s'arrête en signalant que le CSP traité ne satisfait pas BTP. La fonction résoudre ($P^O = (X^O, D^O, C^O)$) permet de résoudre le problème CSP ternaire $P^O = (X^O, D^O, C^O)$. Si P^O est satisfiable, alors il existe un ordre sur les variables pour lequel P satisfait BTP. Sinon, P ne satisfait pas BTP (comme mentionné dans la preuve du théorème 3.2 de [Cooper et al., 2010]).

Maintenant, nous présenterons quelques résultats dans des tableaux en utilisant, pour un CSP donné, les notations qui sont définies ci-dessous :

– #BT : nombre de triangles cassés (**B**roken **T**riangle),
– #BTV : nombre de triplets cassés (**B**roken **T**riplet of **V**ariables),
– #BV : nombre de variables cassées (**B**roken **V**ariable),

Pour le calcul de ces paramètres, notre algorithme ne s'arrête pas au premier triplet cassé, il continue jusquà la fin du traitement de tous les triplets possibles.

Le tableau 5.1 fournit les résultats sur quelques benchmarks qui appartiennent à BTP ou à une de ses classes cachées. Nous avons sélectionné quelques benchmarks qui permettent d'illustrer tous les cas possibles pour BTP. La colonne BTP indique si une instance satisfait (yes) ou non (no) BTP. La colonne x/y indique pour la famille de benchmark correspondant, le nombre de benchmarks satisfaisant BTP (valeur x) par rapport au nombre de benchmarks y de la famille.

Benchmark	n	e	d	$\#BT$	$\#BTV$	$\#BV$	BTP	x/y
normalized-mps-diamond.xml	2	1	2	0	0	0	yes	1/49
normalized-hanoi-5_ext	30	29	243	28	0	28	yes	5/5
normalized-langford-2-4-ext	8	28	8	5700	56	8	no	0/4
normalized-composed-25-1-25-0_ext	33	247	10	171210	727	33	no	0/10
normalized-domino-100-100	100	100	100	-	0	100	no	0/23
normalized-geom-40-2-ext	40	78	2	530	55	39	no	0/22
normalized-driver-01c-sat_ext	71	217	4	922	74	67	no	0/7

TABLE 5.1 – Étude de BTP sur quelques benchmarks.

D'une façon générale, nous pouvons constater que :

– certains benchmarks satisfont BTP :
  • soit du fait de l'existence d'un ordre bien déterminé sur les variables (par exemple, pour `hanoi` on a yes dans la colonne BTP et une valeur différente de 0 dans la colonne #BT)
  • soit par rapport à tous les ordres sur les variables et donc de l'absence de triangle cassé (pour `diamond` on a yes dans la colonne BTP et 0 dans la colonne #BT).
– certaines autres ne satisfont pas BTP car :
  • aucun ordre n'existe du fait de la présence d'un triplet cassé (nous pouvons considérer l'exemple de `langford` qui a no dans la colonne BTP et une valeur différente de 0 dans la colonne #BTV)
  • il est impossible de trouver un ordre malgré l'absence de triplet cassé (no dans la colonne BTP et 0 dans la colonne #BTV, `domino` est un exemple).

Pour information, le temps de résolution de ces benchmarks par MAC (pas de leur reconnaissance) est inférieur à 0,01 secondes.

La figure 5.5 présente un CSP à trois variables qui ne satisfait pas BTP, mais il satisfait BTP^{AC}.

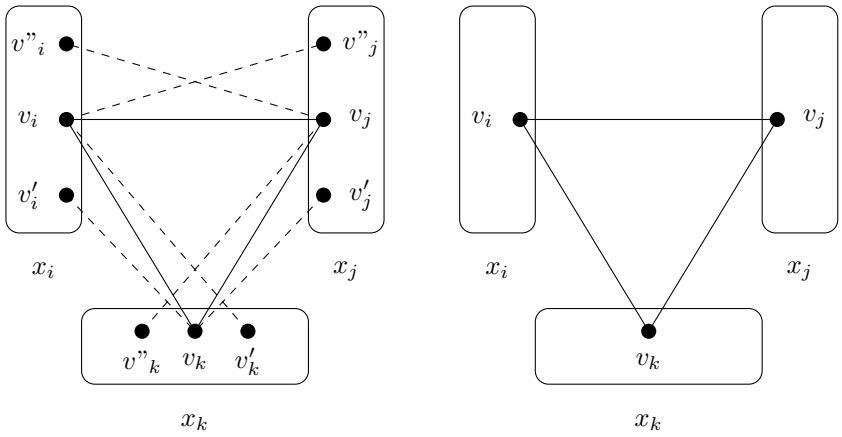

FIGURE 5.5 – Un CSP appartenant à BTP^{AC}.

Sur cette figure, nous remarquons que les arêtes en pointillés contribuent à former trois triangles cassés (donc non BTP). Ces arêtes disparaitront après application de la cohérence d'arc. Ceci peut se manifester dans certains benchmarks comme le montre la table 5.2 pour laquelle nous avons aussi considéré des benchmarks satisfaisant BTP, pour juger du nombre de triangles cassés supprimés par AC.

Benchmark	n	e	d	$\#BT$ après AC	$\#BTV$ après AC	BTP après AC	Nombre
normalized-mps-diamond.xml	2	1	2	0	0	yes	1/49
normalized-hanoi-5_ext	30	29	243	0	0	yes	5/5
normalized-langford-2-4-ext	8	28	8	588	34	no	0/4
normalized-composed-25-1-25-0_ext	33	247	10	169192	727	no	10/10
normalized-domino-100-100	100	100	100	0	0	yes	15/23
normalized-geom-40-2-ext	40	78	2	530	55	no	0/22
normalized-driverlogw-01c-sat_ext	71	217	4	755	69	no	0/7

TABLE 5.2 – Étude de BTP sur quelques benchmarks après application de AC.

À partir de la table 5.2, nous pouvons établir quatre constats prévisibles. En effet, le filtrage par AC a permis :

- de supprimer certains triangles cassés (voire tous) d'un benchmark qui ne satisfait pas BTP initialement mais le devient après AC,
- de supprimer des triangles cassés d'un benchmark qui n'est BTP ni avant, ni après filtrage AC.
- de supprimer tous les triangles cassés d'un benchmark qui satisfait BTP initialement par rapport à un ordre bien déterminé sur les variables.

On constate enfin qu'il existe des benchmarks qui ne satisfont pas BTP après AC et pour lesquelles le nombre de triangles cassés est resté le même. Cela conduit tout naturellement à l'emploi de niveaux de filtrage plus puissants qui sont capables de supprimer d'autres triangles cassés. La cohérence de chemin, par exemple, permet de supprimer des triangles cassés qui ne peuvent jamais être supprimés par la cohérence d'arc, ceci se manifeste dans la figure 5.6.

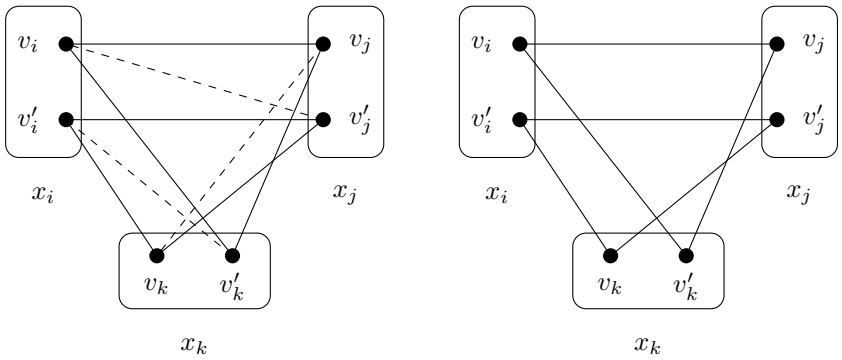

FIGURE 5.6 – Un CSP appartenant à BTP^{PC}.

Les arêtes en pointillés seront supprimées après application de PC. L'application de l'arc cohérence ne touchera à aucune arête de cette micro-structure.

Comme pour AC, PC ne permet pas de supprimer tous les triangles cassés. Nous allons maintenant illustrer un exemple (figure 5.7) globalement cohérent alors qu'il n'est pas BTP et il n'appartiendra à aucune de ses classes cachées même en utilisant un filtrage par cohérence plus puissant que la cohérence de chemin forte comme moyen de transformations.

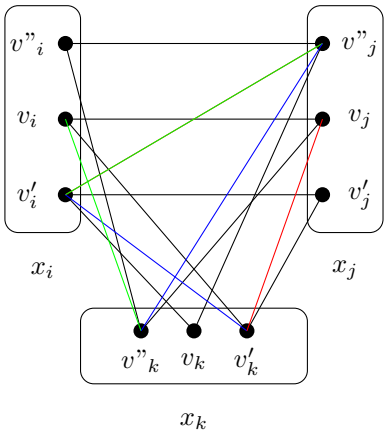

FIGURE 5.7 – Un CSP non-BTP même après SPC.

La table 5.4 indique le nombre de benchmarks qui appartiennent à BTP ou à une de ses classes cachées considérées et le nombre de benchmarks qui sont cohérents par rapport au filtrage correspondant. On peut noter que, sans aucune transformation, seuls 12 benchmarks sont BTP. En revanche, grâce à l'exploitation des classes cachées, nous pouvons constater que plusieurs benchmarks appartiennent à une classe traitable. Par exemple, 550 benchmarks (parmi lesquels 47 sont SAC-cohérents) appartiennent à la classe BTP^{SAC}. Bien sûr, nous pouvons observer que la classe cachée la plus grande correspond à une classe transformée par le filtrage ayant la puissance la plus élevée. Ainsi, le plus grand nombre de benchmarks appartenant à une classe cachée est évidemment atteint par NIC et SPC.

La table 5.3 illustre les noms de certains benchmarks qui appartiennent à BTP ou à une de ses

classes cachées. Tout d'abord, nous pouvons constater la diversité de ces benchmarks (benchmarks académiques, aléatoires ou benchmarks du monde réel). Ensuite, les résultats présentés mettent également en évidence le fait que BTP est une classe traitable hybride. En effet, certains benchmarks appartiennent à BTP grâce à leur structure particulière (leur graphe de contrainte est acyclique) comme `hanoi-3` ou `graph12-w0` tandis que certaines autres comme `pigeons-20-ord` sont BTP en raison de leurs relations particulières.

Finalement, du point de vue résolution, l'appartenance à la classe BTP ou à une de ses classes cachées peut expliquer l'efficacité de la plupart des solveurs sur ces benchmarks. En effet, dans [Cooper et al., 2010] (théorème 7.6), il a été démontré que MAC peut résoudre tout CSP satisfaisant BTP en temps polynomial sans aucun traitement supplémentaire. Ce résultat peut être facilement étendu à RFL et plus généralement à tout algorithme qui maintient à chaque nœud un niveau de cohérence (basée uniquement sur la suppression des valeurs) au moins aussi puissant que la cohérence d'arc. Comme la plupart des solveurs de l'état de l'art exploitent un tel niveau de cohérence, ils sont capables de résoudre des CSP satisfaisant BTP en temps polynomial. Il en est de même pour les CSP appartenant à une classe cachée de BTP dès que le solveur applique la cohérence appropriée avant la résolution, sauf pour la classe BTP^{AC} pour laquelle aucun pré-traitement n'est nécessaire comme indiqué dans le théorème 5.5. Par exemple, MAC et RFL résolvent le benchmark `large-80-sat` sans aucun retour arrière indépendamment de l'ordre sur les variables.

TABLE 5.3 – Quelques benchmarks appartenant à BTP ou à une de ses classes cachées.

Benchmarks	BTP	BTP^{AC}	BTP^{PIC}	BTP^{maxRPC}	BTP^{SAC}	BTP^{NIC}	BTP^{SPC}
bqwh-15-106-43_ext	no	no	no	no	no	no	yes
domino-100-100	no	yes	yes	yes	yes	yes	yes
ehi-90-315-96_ext	no	no	yes	yes	yes	yes	yes
ehi-90-315-97_ext	no	no	no	yes	yes	yes	yes
fapp17-0300-10	no	yes	yes	yes	yes	yes	yes
graph12-w0	yes	yes	yes	yes	yes	yes	yes
hanoi-3_ext	yes	yes	yes	yes	yes	yes	yes
langford-4-8	no	no	no	no	no	yes	yes
large-80-sat_ext	no	yes	yes	yes	yes	yes	yes
os-taillard-4-95-0	no	no	no	no	yes	yes	yes
pigeons-20-ord	yes	yes	yes	yes	yes	yes	yes
queens-4	no	no	yes	yes	yes	yes	yes
rand-23-23-253-131-48021_ext	no	no	no	no	no	yes	no
rand-2-40-180-84-900-93_ext	no	no	no	no	yes	no	yes
will199GPIA-6	no	no	no	no	no	yes	no

TABLE 5.4 – Nombre de benchmarks appartenant à BTP ou à une de ses classes cachées et le nombre de benchmarks cohérents par rapport au filtrage considéré.

	BTP	BTP^{AC}	BTP^{PIC}	BTP^{maxRPC}	BTP^{SAC}	BTP^{NIC}	BTP^{SPC}
# inst.	12	191	400	493	550	900	594
# cons.	-	46	47	47	47	83	71

5.3 BTP pour la fusion des valeurs

Dans cette section, nous rappellerons certaines méthodes de fusion de valeurs introduites précédemment dans la littérature. Ensuite, nous proposerons une nouvelle approche pour la fusion des valeurs basée sur BTP. Nous finirons par illustrer une étude expérimentale pour mettre en valeur notre approche.

5.3.1 Interchangeabilité et substitutuabilité

Ici, nous rappelons quelques méthodes qui consistent à réduire la taille des domaines par la suppression des valeurs. Ces méthodes ont été définies par Freuder dans [Freuder, 1991] et elles sont appelées *interchangeabilité* et *substitutuabilité*. Nous commençons par rappeler leurs définitions.

Définition 5.8 (Interchangeable (FI)). *Une valeur $v_i \in D(x_i)$ est **complètement interchangeable** avec $v'_i \in D(x_i)$, si et seulement si :*
- *toute solution contenant v_i reste solution en substituant v_i par v'_i, et*
- *toute solution contenant v'_i reste solution en substituant v'_i par v_i.*

L'interchangeabilité est une propriété tellement forte qu'elle existe très rarement en pratique. Donc, une définition moins restrictive, appelée substitution, a été proposée.

Définition 5.9 (Substitution (Sub)). *Étant données deux valeurs v_i et $v'_i \in D(x_i)$, v_i est **substituable** par v'_i, si et seulement si, substituer v_i dans toute solution impliquant v'_i donne une autre solution.*

Définition 5.10 (substituabilité de voisinage (NSub)). *Étant données deux valeurs v_i et $v'_i \in D(x_i)$, v'_i est **substituable de voisinage** par v_i, si et seulement si, pour toute contrainte c_{ij}, $\{v_j \in D(x_j) \mid (v'_i, v_j) \in R(c_{ij})\} \subseteq \{v_j \in D(x_j) \mid (v_i, v_j) \in R(c_{ij})\}$*

Dans [Karakashian et al., 2010], Karakashian et al. ont établi les liens entre quelques formes d'interchangeabilités et de substitutuabilités et ont aussi corrigé quelques relations de [Freuder, 1991]. Les résultats qui nous intéressent sont donnés dans le théorème 5.6 (la relation « propriété1 $\rightarrow$ propriété2 » est utilisée pour exprimer que la propriété1 implique propriété2).

Théorème 5.6. *FI $\rightarrow$ Sub.*
NSub $\rightarrow$ Sub.
NSub et FI sont incomparables.

5.3.2 Fusion de valeurs pour CSP binaires basée sur BTP

Dans cette section, nous considérons une méthode, basée sur BTP, pour réduire la taille des domaines tout en préservant la satisfiabilité. Au lieu d'éliminer des valeurs par les opérations classiques de réduction comme la cohérence d'arc ou la substitution de voisinage, nous allons fusionner deux valeurs satisfaisant une condition bien particulière. En fait, nous montrons que l'absence de triangles cassés sur deux valeurs d'une variable x_k, dans un CSP binaire, permet de fusionner ces deux valeurs dans le domaine de x_k, tout en préservant la satisfiabilité.

Cette règle généralise la notion de l'interchangeabilité virtuelle [Likitvivatanavong and Yap, 2013] ainsi que la substitution de voisinage [Freuder, 1991]. Dans [Cohen et al., 2013], Cohen et al ont prouvé que, pour une variable donnée x_k d'un CSP binaire P arc-cohérent, si aucun *triangle cassé* (illustré dans la figure 5.1 ne se produit sur deux valeurs $v'_k, v''_k \in D(x_k)$ et par rapport à deux affectations de deux autres valeurs des variables x_i et x_j (avec $i \neq j \neq k$), la variable x_k peut être éliminée de P sans changer la satisfiabilité de P.

En d'autres termes, si x_k (une variable de P) n'est pas une variable cassé quelque soit le triplet de variables considéré, alors x_k peut être éliminée du CSP de départ et le CSP obtenu par cette transformation respectera la satisfiabilité de P.

Ici, nous montrerons que même lorsque cette règle d'élimination de variable ne peut être appliquée à cause de l'existence d'au moins un triangle cassé sur x_k, il se peut qu'il existe deux valeurs $v'_k, v''_k \in D(x_k)$, sur lesquelles aucun triangle ne s'est produit. Si c'est le cas, nous pouvons effectuer une opération de réduction de domaine qui consiste à fusionner les valeurs v'_k et v''_k.

Définition 5.11. ***Fusionner*** *les valeurs $v'_k, v''_k \in D(x_k)$ d'un CSP binaire consiste à remplacer v'_k, v''_k dans $D(x_k)$ par une nouvelle valeur v_k qui sera compatible avec toutes les valeurs qui sont compatibles avec au moins une des deux valeurs v'_k ou v''_k. Une* ***condition de fusion de valeurs*** *v'_k et v''_k d'un CSP binaire P est une propriété $T(x_k, v'_k, v''_k)$ calculable en temps polynomial de façon que lorsque cette propriété existe, alors le CSP obtenue après fusion de valeurs v'_k et v''_k est satisfiable, si et seulement si, le CSP de départ est satisfiable aussi.*

Nous définissons maintenant formellement la condition de fusion de valeurs sur la base de BTP.

Définition 5.12. *La paire de valeurs $v'_k, v''_k \in D(x_k)$ est* ***sans-TC*** *(ou* ***BT-free****) s'il n'existe pas de triangle cassé sur v'_k, v''_k.*

Proposition 5.8. *Pour un CSP binaire, être sans-TC (BT-free) est une condition de fusion de valeurs. En outre, étant donnée une solution de CSP obtenu après une fusion de valeurs, nous pouvons trouver une solution au CSP de départ en temps linéaire.*

Preuve : Étant donné un CSP P et P' le nouveau CSP obtenu après fusion des deux valeurs v'_k et v''_k en v_k. Clairement, si P est satisfiable alors P' est aussi satisfiable. Il suffit de montrer que si P' a une solution s qui affecte v_k à x_k, alors P a une solution. Soient s' et s'' des affectations identiques à s sauf que s' affecte v'_k à x_k et s'' affecte v''_k à x_k. Supposons que s' et s'' ne sont pas des solutions de P. Alors, il existe deux variables $x_i, x_j \in X \setminus \{x_k\}$ telles que $(s[\{x_i\}], v'_k) \notin R(c_{ik})$ et $(s[\{x_j\}], v''_k) \notin R(c_{jk})$. s est une solution de P, donc on a $(s[x_i], v_k) \in R(c_{ik})$. Par définition de la fusion, il faut que

- $(s[x_i], v'_k) \in R(c_{ik})$ ou
- $(s[x_i], v''_k) \in R(c_{ik})$

et

- $(s[x_j], v'_k) \in R(c_{jk})$ ou
- $(s[x_j], v''_k) \in R(c_{jk})$.

Comme on a

- $(s[x_i], v'_k) \notin R(c_{ik})$ et
- $(s[\{x_j\}], v''_k) \notin R(c_{jk})$,

alors nous devons forcément avoir

- $s[\{x_i\}], v''_k) \in R(c_{ik})$ et
- $(s[\{x_j\}], v'_k) \in R(c_{jk})$.

Finalement, $(s'[\{x_i\}], s''[\{x_j\}]) \in R(c_{ij})$ car s est une solution de P'. Par conséquent, les valeurs $s'[\{x_i\}]$, $s''[\{x_j\}]$,v'_k et v''_k forment un triangle cassé (v''_k,$s'[\{x_i\}]$,$s''[\{x_j\}]$,v'_k), ce qui est contradictoire avec l'hypothèse de départ. Donc, l'absence d'un triangle cassé sur les valeurs v'_k et v''_k permet de les fusionner en une seule valeur tout en préservant la satisfiabilité du CSP.

Reconstruire une solution de P d'une solution s de P' exige tout simplement la vérification que s' ou s'' est une solution de P. □

Nous pouvons constater que la règle de fusion par BTP donnée par la proposition 5.8, généralise la substitution de voisinage [Freuder, 1991] si v''_k est le voisin substituable par v'_k, alors l'absence de triangle cassé sur v'_k, v''_k et la fusion de v'_k et v''_k produit un CSP qui est identique (à l'exception de l'appellation de la valeur v'_k par v_k) au CSP obtenu en éliminant tout simplement v''_k de $D(x_k)$. La fusion par BTP généralise également la règle de fusion proposée par Likitvivatanavong et Yap [Likitvivatanavong and Yap, 2013]. L'idée de base derrière leur règle est que si les deux valeurs v'_k et v''_k sont compatibles avec toutes les mêmes valeurs de toutes les autres variables, sauf en ce qui concerne au plus une autre variable, alors nous pouvons fusionner v'_k et v''_k. Ceci est clairement pris en compte par la règle de fusion par BTP vu que v'_k et v''_k ont été choisies d'une façon qui garantit l'absence des triangles cassés sur ces deux valeurs.

D'après la proposition 5.8, l'opération de fusion par BTP préserve non seulement la satisfiabilité, mais nous pouvons également reconstruire une solution en temps polynomial du CSP d'origine P à partir d'une solution du CSP P^m auquel nous avons appliqué une séquence d'opérations de fusion jusqu'à la convergence vers un point fixe. Il est connu que pour la substituabilité de voisinage, toutes les solutions du CSP d'origine peuvent être générées en O($N(de + n^2)$), où N est le nombre de solutions du CSP original [Cooper, 1997]. Nous montrons maintenant qu'un résultat similaire est également valable pour le cas plus général défini par la fusion par BTP.

Proposition 5.9. *Soit P un CSP binaire et supposons qu'on dispose de l'ensemble des solutions du CSP P^m obtenu après application d'une séquence d'opérations de fusion par BTP. Toutes les N solutions de P peuvent être alors déterminées en O(Nn^2d).*

Preuve : Soit P' un CSP obtenu après avoir effectué une seule opération de fusion par BTP sur les valeurs $v'_k, v''_k \in D(x_k)$ de P. Comme nous avons vu dans la preuve de la proposition 5.8, compte tenu de l'ensemble des solutions $sol(P')$ de P', nous pouvons générer l'ensemble des solutions de P en testant pour chaque $s \in Sol(P')$ si s' ou s'' (ou les deux) sont des solutions de P. Cela nécessite une complexité temporelle O (n) par solution, car il y a au plus $n-1$ contraintes à tester impliquant la variable x_k, et au moins l'une des solution s' ou s'' est une solution de P.

Le nombre total d'opérations de fusion par BTP effectuées pour transformer P en P^m est au plus $n(d-1)$. Par conséquent, le temps total pour générer toutes les N solutions de P de l'ensemble des solutions de P^m est O(Nn^2d). □

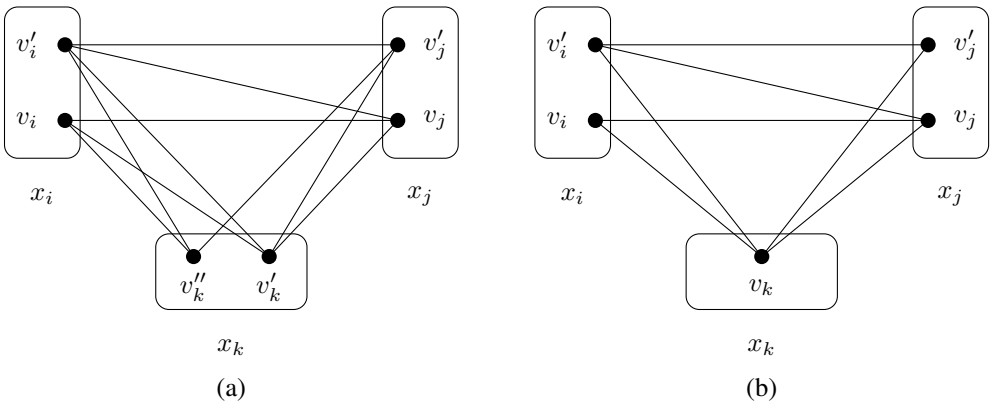

FIGURE 5.8 – (a) un motif avec un triangle cassé sur les valeurs v_j, v'_j de la variable x_j. (b) après fusion par BTP des valeurs v''_k et v'_k de $D(x_k)$ en v_k, le triangle cassé a disparu.

La version allégée de la substitution de voisinage a la propriété que deux différentes séquences convergentes d'éliminations par substitution de voisinage produisent nécessairement des CSP isomorphes P_1^m, P_2^m [Cooper, 1997]. Or, ceci n'est pas le cas pour la fusion par BTP. Tout d'abord, et peut-être d'une manière surprenante, la fusion par BTP peut avoir comme effet secondaire l'élimination de certains triangles cassés. Ceci est illustré par un motif à 3 variables représenté dans la figure 5.8. L'exemple de la figure 5.8(a) contient un triangle cassé (v_j, v_i, v''_k, v'_j) sur les valeurs v_j, v'_j de la variable x_j, mais après fusion par BTP des valeurs $v'_k, v''_k \in D(x_k)$ dans une nouvelle valeur v_k, comme dans la figure 5.8(b), ce triangle cassé a disparu. Deuxièmement, la fusion par BTP de deux valeurs de $D(x_k)$ peut créer un triangle cassé sur une variable $x_i \neq x_j$, comme illustré par la figure 5.9. Le CSP de la figure 5.9 (a) ne contient pas de triangle cassé, mais après la fusion par BTP de $v_j, v'_j \in D(x_j)$ dans une nouvelle valeur c, un triangle cassé (v'_i, v''_j, v_k, v_i) a été créé sur les valeurs v_i, v'_i de $D(x_i)$.

La figure 5.5 illustre le cas d'un CSP qui ne satisfait pas BTP initialement à cause de la présence d'un triangle cassé sur chaque variable de triplet. Après fusion par BTP, ce CSP satisfera BTP.

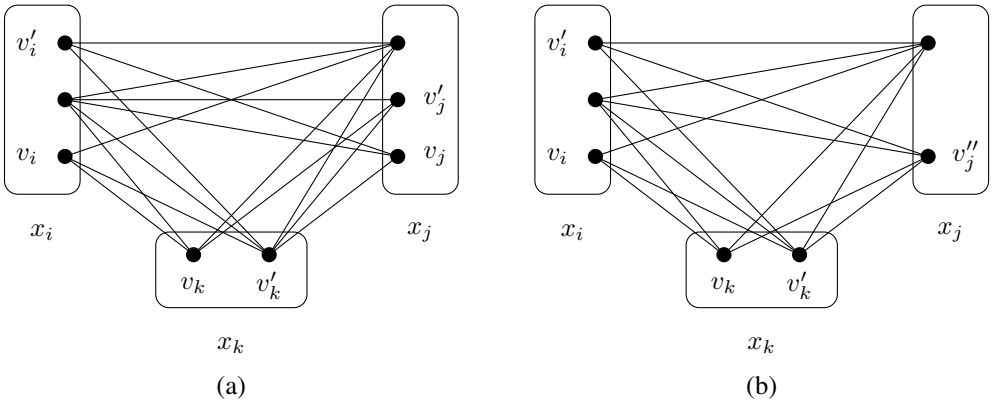

FIGURE 5.9 – (a) un motif sans triangle cassé. (b) après fusion par BTP des valeurs v_j de v'_j de $D(x_j)$, un triangle cassé est apparu sur les valeurs $v_i, v'_i \in D(x_i)$.

5.3.3 Résultats expérimentaux

Pour tester l'utilité de la fusion par BTP, nous avons effectué de nombreux tests expérimentaux sur des benchmarks de la compétition internationale CSP 2008. Pour chaque benchmark (sauf les benchmarks contenant des contraintes globales car notre librairie ne les prend pas en compte), nous avons lancé la fusion par BTP jusqu'à la convergence vers un point fixe avec un délai de traitement d'une heure. Au total, nous avons obtenu les résultats pour 2 547 benchmarks sur 3 811. Pour les autres benchmarks, il faudra plus de temps pour que la fusion par BTP se termine.

Famille	#-benchmarks	#-valeurs	#-valeurs supprimées	%age supprimées
BH-4-13	6	7 334	3 201	44%
BH-4-4	10	674	322	48%
BH-4-7	20	2 102	883	42%
ehi-85	98	2 079	891	43%
ehi-90	100	2 205	945	43%
graph-coloring/school	8	4 473	104	2%
graph-coloring/sgb/book	26	1 887	534	28%
jobShop	45	6 033	388	6%
marc	1	6 400	6 240	98%
os-taillard-4	30	2 932	1 820	62%
os-taillard-5	28	6 383	2 713	43%
rlfapGraphsMod	5	14 189	5 035	35%
rlfapScens	5	12 727	821	6%
rlfapScensMod	9	9 398	1 927	21%
autres	1 919	1 396	28	0,02%

TABLE 5.5 – Résultats expérimentaux sur les benchmarks.

Tous les benchmarks de la famille `hanoi` satisfont la propriété BTP (voir table 5.1) et la fusion par BTP réduit tous les domaines de ses variables à une seule valeur. Après avoir établie la cohérence d'arc, 46 benchmarks deviendront BTP (elles appartiennent à BTP^{AC}), nous pouvons, entre-autres, citer tous les benchmarkss de la famille `domino`. Nous n'avons pas compté les benchmarks pour lesquels la cohérence d'arc détecte une incohérence par la production d'un CSP trivial avec des domaines de variables qui sont vides (et qui satisfait trivialement BTP). Pour tous les benchmarks de la famille `pigeons` avec un suffixe `-ord`, la fusion par BTP réduit de nouveau tous les domaines à une seule valeur. C'est parce que la fusion par BTP peut éliminer les triangles cassés, comme nous l'avons signalé à la section 5.3.2, et ceci peut donc rendre un CSP BTP même s'il ne l'était pas au départ. Le même phénomène s'est produit pour un benchmark de 680 variables de la famille `rlfapGraphsMod` ainsi que le benchmark à 3 variables `ogdPuzzle`.

La table 5.5 donne un récapitulatif des résultats expérimentaux. Nous n'incluons pas les cas mentionnés ci-dessus qui sont entièrement résolus par la fusion par BTP. Nous donnons plus de détails sur les benchmarks où la fusion par BTP a été le plus efficace. Tous les autres benchmarks sont regroupés dans la dernière ligne de la table. La table indique :

- le nom de la famille de benchmarks,
- le nombre de benchmarks de cette famille qui ont été testés,
- la moyenne de nombre total de valeurs de cette famille de benchmarks,
- la moyenne de nombre de valeurs supprimées de cette famille de benchmarks (c'est-à-dire le nombre d'opérations de fusion par BTP effectuées),
- le pourcentage de la moyenne de valeurs supprimées.

Nous pouvons voir que pour certains types de problème (`marc`, `os-taillard-4`, la fusion par BTP est très efficace et le pourcentage de valeurs supprimées dépasse le 50 %, alors que pour d'autres (regroupés dans la dernière ligne de cette table) la fusion par BTP a supprimé un nombre très limité de valeurs. Nous signalons aussi que certains benchmarks comme `domino/normalized-domino-100-100` satisfont BTP après application de fusion par BTP alors qu'il ne l'était pas au début. Tous les benchmarks qui sont devenus BTP après fusion par BTP appartiennent aussi à BTP^{AC}. D'un point de vue théorique, appliquer la cohérence d'arc ou la fusion par BTP peut donner deux résultats différents. Par exemple, les sommets du motif de la figure 2.18(a) seront supprimés après application de AC alors que la fusion par BTP ne supprimera aucun sommet.

5.4 Conclusion

Dans ce chapitre, nous avons étudié expérimentalement la propriété BTP de deux façon différentes et nous avons obtenu chaque fois des résultats intéressants. Dans la première partie, nous avons étudié la notion de *classe polynomiale cachée* en relation avec les méthodes utilisées dans les solveurs CSP de l'état de l'art. Si le concept de *structure cachée* a déjà été évoqué dans [Williams et al., 2003] avec les notions de $Backbones$ et *backdoors*, ici nous le traitons différemment tout en introduisant un cadre qui semble plus général. Dans cette direction, nous avons mis en place un cadre formel pour la définition de la notion de *classe cachée découverte par des transformations de CSP*. Ce cadre permet de couvrir le concept de structure cachée proposée dans [Williams et al., 2003] mais il contribue également à développer d'autres approches. En particulier, nous avons étudié la notion de classes traitables détectables après filtrage : le filtrage est considéré ici comme un cas particulier de transformations. Plus précisément, nous avons illustré notre approche avec la classe BTP [Cooper et al., 2010] . D'un point de vue pratique, nous avons montré que certains benchmarks classiquement utilisés par la communauté, appartiennent à BTP après l'application de filtrages classiques (comme AC, SAC, PIC, etc). Du point de vue résolution, des algorithmes tels que MAC ou RFL fonctionnent en effectuant une séquence de transformations de CSP (affecta-

tions des valeurs aux variables, filtrage par cohérence d'arc...). Ainsi, il est possible que l'efficacité de ces solveurs soit due au fait que le CSP obtenu, après avoir traité certains nœuds de l'arbre de recherche, appartient à une des classes cachées.

Grâce à cette analyse, il serait également possible de mettre en évidence de nouvelles classes traitables basées sur l'analyse des CSP facilement résolues par des algorithmes standards, comme le cas des CSP qui semblent ne pas appartenir explicitement aux classes polynomiales connues. Enfin, la notion de transformations de CSP a été présentée ici d'une manière restrictive, car elle est essentiellement définie par des simplifications de CSP. Il serait intéressant de l'étendre, par exemple en considérant les transformations qui ajoutent des variables ou des valeurs, ou celles qui utilisent la relaxation de contraintes. Il est important donc d'étendre cette étude sur des classes polynomiales autres que BTP et d'utiliser d'autres moyen de transformations de complexité polynomiale. Grâce à une telle analyse, nous pourrions peut-être définir des nouvelles classes polynomiales qui pourraient être détectables en utilisant ces nouvelles transformations. Par exemple, ce que nous avons présenté dans la deuxième section, à savoir la fusion par BTP, peut être considéré comme une nouvelle forme de transformation de CSP basée sur la fusion de valeurs. En effet, la fusion par BTP décrit une nouvelle opération de réduction de domaines pour les CSP binaires qui est strictement plus forte que la substitution de voisinage. Des tests expérimentaux ont montré que, pour plusieurs benchmarks l'application de la fusion par BTP jusqu'à la convergence vers le point fixe peut réduire d'une façon significative le nombre total de valeurs.

Pour le moment, nous ne disposons pas d'un résultat théorique sur l'effet de l'ordre des variables sur le résultat final obtenu aprés application de la fusion par BTP. Donc, il serait intéressant d'envisager cette piste de recherche et d'étendre ces résultats aux CSP d'arité quelconque.

Chapitre 6

DBTP : une extension de BTP aux CSP d'arité quelconque

6.1 Introduction

Dans l'état de l'art, nous avons rappelé que la classe BTP est *hybride*, c'est-à-dire qu'elle capte des classes *structurelles* (comme les CSP binaires de la classe $TREE$) et des classes *relationnelles* (comme les CSP binaires de la classe RRM). Dans le cas binaire, nous avons montré que BTP fournit de résultats intéressants en théorie [Cooper et al., 2010] et en pratique (chapitre 5). Malheureusement, la définition de BTP couvre seulement les CSP binaires malgré une première extension aux CSP n-aires (lemme 4.6 de[Cooper et al., 2010]) qui n'a pas été développée en théorie, ni en pratique.

Dans ce chapitre, nous étudions l'extension de la classe polynomiale BTP en utilisant la DR-microstructure et nous notons DBTP la nouvelle propriété. Nous montrons que DBTP dispose des propriétés et des résultats proches de ceux de BTP. En effet, nous verrons également que l'ensemble des CSP satisfaisant DBTP définissent une nouvelle classe polynomiale qui couvre simultanément des classes structurelles comme les CSP β-$acycliques$ ainsi que des classes définies par des restrictions de langages. Nous montrerons également que $DBTP$ est incomparable avec plusieurs autres classes bien connues de la littérature (par exemple *ZOA*, *row-convex* ou *max-closed*. En plus de ces résultats théoriques, nous prouverons que DBTP constitue une propriété close pour les filtrages classiques comme notamment la cohérence d'arc. Il semblerait ainsi que DBTP recèle un intérêt pratique puisque les CSP satisfaisant DBTP peuvent être résolues en temps polynomial par l'usage d'algorithmes comme MAC et RFL notamment. Une partie de ce travail présentée dans ce chapitre a été publiée dans [El Mouelhi et al., 2013a] et une version complète a été acceptée à [El Mouelhi et al., 2015].

6.2 BTP et la DR-microstructure

Dans cette section, nous étudions l'extension de la propriété BTP au cas non-binaire et nous montrons qu'en nous focalisant sur la DR-microstructure nous obtenons des résultats similaires à ceux de BTP. Néanmoins, nous verrons que cette extension (qui sera notée *DBTP* pour *Dual Broken Triangle Property*) ne constitue pas pour autant une généralisation de BTP aux contraintes d'arité quelconque puisque pour le cas particulier des CSP binaires, BTP et $DBTP$ sont formellement différentes (voir théorème 6.14).

Dans un premier temps, nous présentons la propriété DBTP d'un point de vue relationnel :

Définition 6.1 (Dual Broken-Triangle Property)**.** *Un CSP $P = (X, D, C)$ vérifie la **Dual Broken Triangle Property** (DBTP) par rapport à un ordre $\prec$ sur les contraintes si pour tout triplet de contraintes (c_i, c_j, c_k) tel que $c_i \prec c_j \prec c_k$, pour tout $t_i \in R(c_i)$, $t_j \in R(c_j)$ et $t_k, t'_k \in R(c_k)$ tels que*
- $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$
- $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$
- $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$

alors
- *soit* $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$
- *soit* $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$

Nous notons $DBTP$ l'ensemble de ces CSP.

Nous pouvons remarquer que la relation entre deux tuples dans la définition de DBTP est inspirée de la DR-microstructure. En fait, quand l'intersection entre deux portées de deux contraintes différentes est vide, nous considérons que chaque tuple de la relation associée à la première contrainte est compatible avec tout tuple de la relation associée à la deuxième contrainte. Autrement dit, il s'agit d'une extension de BTP en se basant sur la DR-microstructure vu qu'elle peut s'exprimer via l'expression d'une propriété relative à la compatibilité entre triplets de tuples figurant dans des triplets de relations de compatibilité.

Cette propriété peut également être considérée comme l'expression de BTP appliquée sur la représentation duale d'un CSP. C'est d'ailleurs en termes d'expression duale que DBTP a d'abord été exprimée dans [Cooper et al., 2010] puisque dans cet article, les auteurs ont évoqué le fait qu'un CSP dual, et donc binaire, pouvait vérifier BTP.

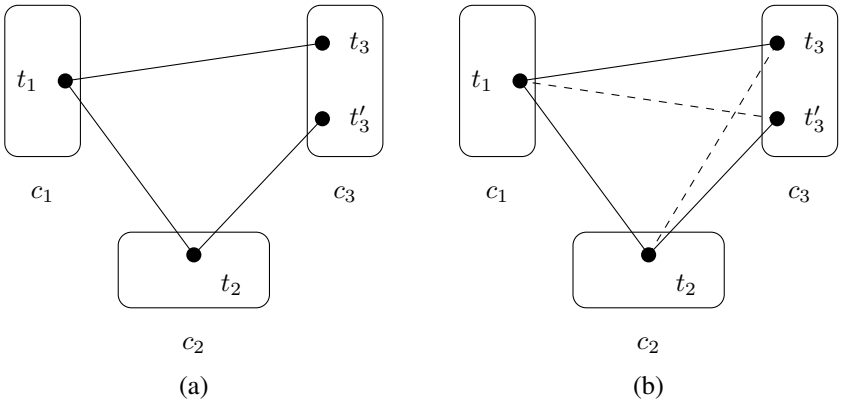

FIGURE 6.1 – Illustration de la propriété DBTP sur trois contraintes c_1, c_2 et c_3.

La figure 6.1 présente la DR-microstructure du CSP P concernant trois contraintes. Dans la figure 6.1(a), nous pouvons observer la présence d'un triangle cassé sur c_3 si nous considérons l'ordre $c_1 \prec c_2 \prec c_3$ et ainsi, P ne vérifie pas DBTP par rapport à cet ordre. Au contraire, dans la figure 6.1(b), si soit t_1 et t'_3, soit t_2 et t_3 (arêtes en pointillées) sont compatibles, alors P vérifie DBTP relativement à l'ordre $\prec$.

6.2.1 Propriétés

Comme BTP, DBTP s'appuie sur le concept de *Triangle Cassé* et constitue cependant une classe polynomiale très différente. Il est bien connu que tout CSP P possède une solution ssi P^d (le CSP

dual de P) possède une solution. Le théorème suivant met en évidence les liens entre DBTP et BTP sur le CSP dual :

Théorème 6.1. *Un CSP $P = (X, D, C)$ vérifie DBTP par rapport à un ordre $\prec$ sur les contraintes ssi le dual de P vérifie BTP par rapport à l'ordre $\prec$.*

Preuve : P vérifie DBTP par rapport à l'ordre $\prec$

$\Leftrightarrow$ pour tout triplet de contraintes (c_i, c_j, c_k) tel que $c_i \prec c_j \prec c_k$, pour tout $t_i \in R(c_i)$, $t_j \in R(c_j)$ et $t_k, t'_k \in R(c_k)$ tels que
- $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$,
- $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$ et
- $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$

alors
- soit $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$,
- soit $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$

$\Leftrightarrow$ pour tout triplet de variables (x^d_i, x^d_j, x^d_k) tel que $x^d_i \prec x^d_j \prec x^d_k$, pour tout $t_i \in D(x^d_i)$, $t_j \in D(x^d_j)$ et $t_k, t'_k \in D(x^d_k)$ tels que
- $(t_i, t_j) \in R(c^d_{ij})$,
- $(t_i, t_k) \in R(c^d_{ik})$ et
- $(t_j, t'_k) \in R(c^d_{jk})$

alors
- soit $(t_i, t'_k) \in R(c^d_{ik})$,
- soit $(t_j, t_k) \in R(c^d_{jk})$

$\Leftrightarrow$ P^d vérifie BTP par rapport à l'ordre $\prec$. $\square$

Nous pouvons aussi noter que DBTP est très différente de BTP. En particulier, un CSP binaire peut vérifier DBTP sans satisfaire BTP. Par exemple, le CSP binaire décrit dans la figure 6.2 est DBTP par rapport à l'ordre $c_{ij} \prec c_{jk} \prec c_{ik}$ mais elle n'est pas BTP. Il n'est pas surprenant que DBTP n'implique pas BTP car, même si le CSP original et son expression duale représentent le même problème, leurs structures et microstructures sont très différentes. Les liens entre DBTP et BTP seront étudiés de façon plus détaillée dans la partie 6.3 qui leur est dédiée.

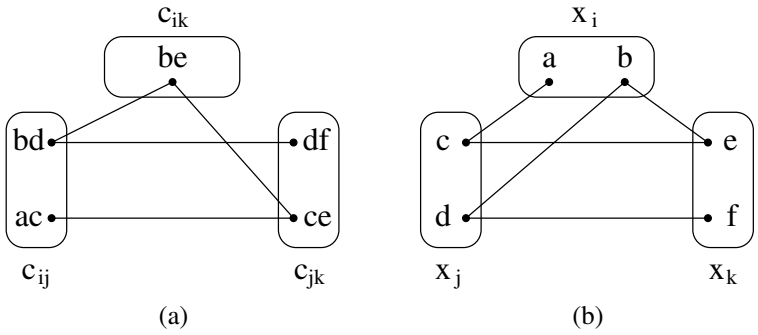

FIGURE 6.2 – Un CSP vérifiant DBTP (a) mais pas BTP (b).

Nous montrons maintenant que la classe des CSP vérifiant DBTP constitue une classe polynomiale (sous l'hypothèse que les contraintes du CSP soient exprimées en extension). Pour cela, et du fait du théorème 6.1, les preuves s'appuient sur les mêmes schémas que ceux utilisés dans [Cooper et al., 2010].

Lemme 6.1. *Tout CSP $P = (X, D, C)$ qui vérifie DBTP par rapport à un ordre $\prec$ sur les contraintes peut être résolu en $O(e^2.a.r^2)$.*

Preuve : La première étape consiste à construire le dual de P, ce qui peut être réalisé en O($e^2.a.r^2$). Ensuite, comme le dual de P est BTP, nous savons qu'il peut être résolu en O($e^2.r^2$) [Cooper et al., 2010]. Ainsi, la complexité globale est en O($e^2.a.r^2$). □

Le lemme 6.2 exprime le fait qu'un ordre $\prec$ sur les contraintes et associé à DBTP peut être calculé (le cas échéant) en temps polynomial. En effet, il suffit d'appliquer la même approche proposée dans [Cooper et al., 2010] pour le calcul d'un ordre sur les variables pour BTP.

Lemme 6.2. *Étant donné un CSP $P = (X, D, C)$, déterminer s'il existe un ordre $\prec$ sur les contraintes tel que P est DBTP par rapport à $\prec$ (et le trouver le cas échéant) peut être réalisé en temps polynomial.*

Preuve : Un algorithme possible consiste à calculer d'abord le dual de P, puis à déterminer si un ordre $\prec$ tel que le dual de P est BTP existe comme proposé dans [Cooper et al., 2010]. Chaque étape est polynomiale (voir la preuve précédente et [Cooper et al., 2010]). Par conséquent, la complexité globale est polynomiale. □

Du fait de ces deux lemmes, nous pouvons déduire le théorème suivant :

Théorème 6.2. *$DBTP$ est une classe polynomiale.*

6.2.2 Conservation par filtrage et ses conséquences sur la résolution

Nous savons que la propriété BTP est close pour tout filtrage qui supprime des valeurs de domaines. Ici, nous étudions ce qu'il en est de la propriété DBTP dans le cas de l'application d'un algorithme de filtrage sur un CSP satisfaisant DBTP.

Proposition 6.1. *DBTP est close pour tout filtrage qui se limite à supprimer des valeurs dans les domaines ou des tuples dans les relations.*

Preuve : Considérons un CSP P vérifiant DBTP par rapport à un ordre donné sur les contraintes. La suppression d'une valeur du domaine d'une variable x de P conduit à une suppression de tuples pour les contraintes dont la portée contient x. En d'autres termes, cela revient à supprimer des valeurs dans les domaines des variables du dual de P. Par conséquent, dans les deux cas, les suppressions de valeurs ou de tuples dans le CSP d'origine conduisent à supprimer les valeurs des variables duales. Comme BTP est close pour les cohérences filtrant les domaines, le dual de P, après ces suppressions, vérifie encore BTP. Par conséquent, P demeure DBTP. □

Grâce à cette propriété, nous pouvons déduire, comme pour BTP, que $DBTP \subsetneq DBTP^{AC}$ et $DBTP \subsetneq DBTP^{PWC}$. Cette propriété est donc valable pour tout filtrage de domaine (par exemple pour la cohérence d'arc généralisée ou la cohérence inverse de chemin), qu'il soit appliqué sur le CSP original ou son CSP dual. C'est également le cas pour l'inter-cohérence dont l'application est équivalente à celle de l'application de la cohérence d'arc sur le CSP dual [Beeri et al., 1983]. Par contre, la propriété n'est pas valable pour la cohérence de chemin car elle rajoute des relations et évidemment des relations à le CSP d'origine.

Comme MAC [Sabin and Freuder, 1994] maintient la cohérence d'arc à chaque étape de la recherche, nous pouvons définir MPWC comme l'algorithme correspondant au maintien de l'inter-cohérence (**M**aintaining *P*air*W*ise *C*onsistency).

Théorème 6.3. *Si $P = (X, D, C)$ vérifie DBTP, alors MPWC résout P en temps polynomial pour tout ordre.*

Preuve : Comme l'inter-cohérence sur P est équivalente à la cohérence d'arc sur le dual de P [Janssen et al., 1989], l'application de MPWC sur P est équivalente à celle de MAC sur le dual de P. De plus, comme P est DBTP, P^d est BTP et ainsi, selon le théorème 7.6 de [Cooper et al.,

2010], MPWC résout P en temps polynomial. □

Finalement, nous pouvons dériver un résultat similaire sur MAC quand l'application de la cohérence d'arc implique l'inter-cohérence. Ici, nous considérons seulement le cas où l'inter-cohérence est une conséquence logique de l'application de la cohérence d'arc. En d'autres termes, nous obtenons l'inter-cohérence par application de la consistance d'arc et non par application explicite de l'inter-cohérence.

Lemme 6.3. *Étant donné un CSP $P = (X, D, C)$. Si le problème P' obtenu à partir de P par suppression de valeurs par application de AC n'implique pas de domaines vides et si l'application de AC implique l'inter-cohérence, alors le dual de P' est arc-cohérent.*

Preuve : considérons P' le problème obtenu à partir de P par suppression de valeurs par application de AC sans impliquer de domaines vides. Puisque P' est AC et l'application de AC engendre l'inter-cohérence, P' est alors inter-cohérent. Comme l'inter-cohérence d'un CSP est équivalente à la cohérence d'arc sur son dual [Janssen et al., 1989], alors le dual de P' est arc-cohérent. □

Avant d'énoncer certains autres résultats sur la résolution par MAC des CSP satisfaisant DBTP, nous rappelons la relation entre la cohérence d'arc et l'inter-cohérence.

Lemme 6.4 (Propriété 8.1 page 146 dans [Jégou, 1991])**.** *Soit $P = (X, D, C)$ un CSP tel que $\forall c_i, c_j \in C, |S(c_i) \cap S(c_j)| \leq 1$. Si le CSP P vérifie la cohérence d'arc, alors il vérifie l'inter-cohérence.*

Lemme 6.5. *Soit $P = (X, D, C)$ un CSP vérifiant la cohérence d'arc et tel que $\forall c_i, c_j \in C, |S(c_i) \cap S(c_j)| \leq 1$. Si le CSP P' obtenu à partir de P en supprimant certaines valeurs et en appliquant AC ne possède pas de domaine vide, alors son dual vérifie la cohérence d'arc.*

Preuve : Considérons P' obtenu à partir de P en supprimant certaines valeurs et en appliquant AC, tel qu'il ne possède pas de domaine vide. Puisque P' vérifie la cohérence d'arc, d'après le lemme 6.4, il vérifie aussi l'inter-cohérence. Ainsi, comme l'inter-cohérence sur P est équivalente à la cohérence d'arc sur le dual de P [Janssen et al., 1989], le dual de P' vérifie la cohérence d'arc. □

Théorème 6.4. *Si $P = (X, D, C)$ satisfait DBTP et si à chaque étape de la recherche l'application de la cohérence d'arc implique l'inter-cohérence, alors MAC peut résoudre P en temps polynomial.*

Preuve : Si l'application de la cohérence d'arc n'engendre ni un domaine ni une relation vide, alors le problème résultant est inter-cohérent et possède au moins une solution. D'après le lemme 6.3, le problème obtenu après suppression de valeurs en appliquant AC reste toujours inter-cohérent. Par conséquent, quand on applique MAC au problème de départ, nous maintenons aussi l'inter-cohérence. En plus, comme l'inter-cohérence est équivalente à la cohérence d'arc sur le problème dual [Janssen et al., 1989], le théorème 7.6 de [Cooper et al., 2010] implique que MAC résoudra P en temps polynomial vu que le dual satisfait BTP. □

Ce résultat est en particulier vérifié pour MAC quand tout couple de contraintes partage au plus une variable.

Théorème 6.5. *Si $P = (X, D, C)$ est tel que $\forall c_i, c_j \in C, |S(c_i) \cap S(c_j)| \leq 1$, et qu'il vérifie la cohérence d'arc ainsi que DBTP, alors MAC peut résoudre P en temps polynomial.*

Preuve : Si après l'obtention de la cohérence d'arc, aucun domaine, ni relation n'est vide, alors P vérifie l'inter-cohérence et possède une solution. D'après le lemme 6.3, le problème obtenu après la suppression de valeurs et le filtrage par cohérence d'arc demeure inter-cohérent. Par conséquent, lors de l'application de MAC sur le problème initial, nous maintenons également l'inter-cohérence. De plus, comme l'inter-cohérence est équivalente à la cohérence d'arc sur le problème dual [Janssen et al., 1989], le théorème 7.6 de [Cooper et al., 2010] fait que MAC

résout P en temps polynomial puisque le dual est BTP. □

Ce théorème est bien entendu vérifié pour tout CSP binaire puisque l'intersection des portées des contraintes est au plus de de taille 1.

Les résultats présentés dans cette section sur MAC sont aussi vrais pour RFL puisqu'il maintient aussi la cohérence d'arc à chaque nœud de l'arbre de recherche.

6.3 Relations entre DBTP et BTP

La classe $DBTP$ diffère nécessairement de la classe BTP puisque $DBTP$ peut contenir des CSP non-binaires alors que BTP n'a été définie qu'au niveau binaire. Ceci conduit à ne se poser la question de leur comparaison que dans ce cadre. Comme le montre la figure 6.2, un CSP peut vérifier DBTP mais pas BTP. Inversement, un CSP peut vérifier BTP sans qu'elle ne vérifie DBTP. Ce cas est illustré dans la figure 6.3 (où les triangles cassés coloriés prouvent que DBTP n'est pas vérifiée). Ces résultats étaient prévisibles, puisque, même si le CSP d'origine et son dual représentent le même problème, leur structure et leur microstructure sont en fait différentes. Ainsi, à partir des exemples des figures 6.2 et 6.3, nous obtenons :

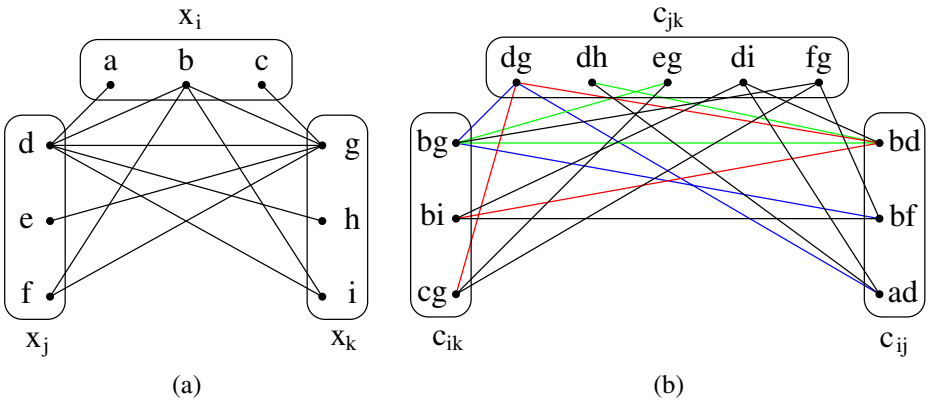

FIGURE 6.3 – Un CSP vérifiant BTP (a) mais pas DBTP (b).

Théorème 6.6. *Soit $P = (X, D, C)$ un CSP binaire.*
- *P vérifie DBTP $\not\Rightarrow$ P vérifie BTP,*
- *P vérifie BTP $\not\Rightarrow$ P vérifie DBTP.*

Les résultats précédents reposent sur la présence de triangles cassés dans la microstructure du CSP ou de son CSP dual. Dans chaque cas, ces triangles cassés concernent des valeurs qui pourraient être supprimées par un certain filtrage comme la cohérence d'arc notamment. Ainsi, comme DBTP et BTP sont closes par rapport aux filtrages de domaines, nous focalisons notre étude sur les CSP binaires qui satisfont la cohérence d'arc ainsi que l'inter-cohérence (du fait du lemme 6.5). Sous ces hypothèses, nous déduisons le lemme suivant :

Lemme 6.6. *Étant donné un CSP binaire $P = (X, D, C)$ vérifiant la cohérence d'arc, si pour un triplet (x_i, x_j, x_k) de variables, nous disposons d'un triangle cassé, alors il existe deux triangles cassés pour le triplet (c_{ij}, c_{ik}, c_{jk}) dans le CSP dual.*

Preuve : Soient x_i, x_j, $x_k \in X$ telles que :
- $(v_i, v_j) \in R(c_{ij})$,
- $(v_i, v_k) \in R(c_{ik})$,

– $(v_j, v'_k) \in R(c_{jk})$,
– $(v_i, v'_k) \notin R(c_{ik})$ et
– $(v_j, v_k) \notin R(c_{jk})$.

Comme P est inter-cohérent (d'après lemme 6.5), il existe des valeurs $v'_i \in D(x_i)$ et $v'_j \in D(x_j)$ telles que $v_i \neq v'_i$, $v_j \neq v'_j$, $(v'_i, v'_k) \in R(c_{ik})$ et $(v'_j, v_k) \in R(c_{jk})$.

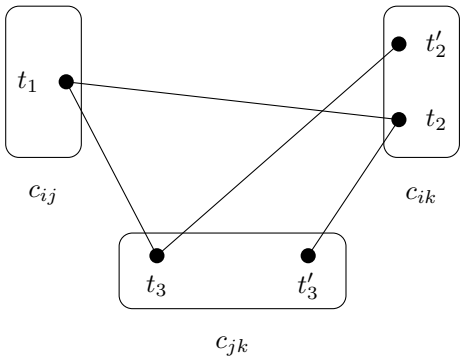

FIGURE 6.4 – Exemple explicatif de la preuve du lemme 6.6.

Si on pose
– $t_1 = (v_i, v_j)$,
– $t_2 = (v_i, v_k)$,
– $t'_2 = (v'_i, v'_k)$,
– $t_3 = (v_j, v'_k)$ et
– $t'_3 = (v'_j, v_k)$.

Ainsi, nous pouvons voir qu'il existe deux triangles cassés (t'_2, t_3, t_1, t_2) et (t'_3, t_2, t_1, t_3) respectivement sur c_{ik} et c_{jk}. □

Par conséquent, quand un triangle cassé pour un triplet (x_i, x_j, x_k) impose la condition $x_k < max(x_i, x_j)$ sur l'ordre $<$ des variables (car si la variable x_k était la dernière dans l'ordre, le CSP ne satisferait pas BTP par définition), cela revient à imposer les deux conditions $c_{jk} \prec max(c_{ij}, c_{ik})$ et $c_{ik} \prec max(c_{ij}, c_{jk})$ pour le triplet (c_{ij}, c_{ik}, c_{jk}) sur l'ordre $\prec$ des contraintes. Il s'ensuit que tout CSP binaire arc-cohérent et inter-cohérent qui satisfait BTP et dispose de deux triangles cassés pour deux variables différentes d'une même triplet de variables ne peut satisfaire DBTP puisque nous obtiendrions tous les triangles cassés possibles pour le triplet correspondant de contraintes.

Inversement, considérons un CSP binaire avec neuf variables $\{x_a, x_b, \ldots, x_i\}$. Nous définissons cet exemple en reproduisant plusieurs fois un même motif qui est tel que chaque valeur apparaissant dans une occurrence de ce motif n'apparaît dans aucune autre occurrence.

Ce motif consiste en un triangle cassé sur une variable z pour un triplet (x, y, z) (c'est-à-dire qui impose la condition $z < max(x, y)$ sur $<$) et chaque valeur des variables x, y et z est liée à une valeur donnée d'une variable qui n'est pas impliquée dans ce triplet. Nous reproduisons ce schéma 9 fois de telle sorte que les conditions suivantes soient imposées :
– $x_a < max(x_b, x_c)$,
– $x_b < max(x_e, x_h)$,
– $x_c < max(x_e, x_g)$,
– $x_d < max(x_a, x_g)$,
– $x_e < max(x_a, x_i)$,

– $x_f < max(x_d, x_e)$,
– $x_g < max(x_h, x_i)$,
– $x_h < max(x_b, x_d)$ et
– $x_i < max(x_c, x_f)$.

La figure 6.5(b) décrit ce motif pour le triplet (x_a, x_b, x_c), un triangle cassé sur x_a (correspondant à la condition $x_a < max(x_b, x_c)$) et une variable indépendante x_e tandis que la figure 6.5 (a) décrit la partie correspondante dans le CSP dual. En faisant cela, la microstructure de notre CSP binaire et celle de son CSP dual ont 9 composantes connexes. On peut noter que ce CSP n'est pas BTP parce que les 9 conditions rendent impossible la construction d'un ordre approprié sur les variables. En revanche, il est DBTP (par rapport à l'ordre $c_{ab} \prec c_{ac} \prec c_{ad} \prec c_{bf} \prec c_{bh} \prec c_{ci} \prec c_{df} \prec c_{dh} \prec c_{ef} \prec c_{ei} \prec c_{gh} \prec c_{gi} \prec c_{af} \prec c_{bc} \prec c_{bd} \prec c_{ce} \prec c_{cg} \prec c_{de} \prec c_{dg} \prec c_{fi} \prec c_{hi} \prec c_{ae} \prec c_{ag} \prec c_{be} \prec c_{bg} \prec c_{cd} \prec c_{cf} \prec c_{di} \prec c_{fh} \prec c_{ai} \prec c_{bi} \prec c_{eg} \prec c_{eh} \prec c_{fg} \prec c_{ah} \prec c_{ch}$), et il vérifie la cohérence d'arc et l'inter-cohérence. Nous avons utilisé a_1 et a_2 comme valeurs du domaine de la variable x_a et pareillement pour b_1, b_2, c_1, c_2, etc.

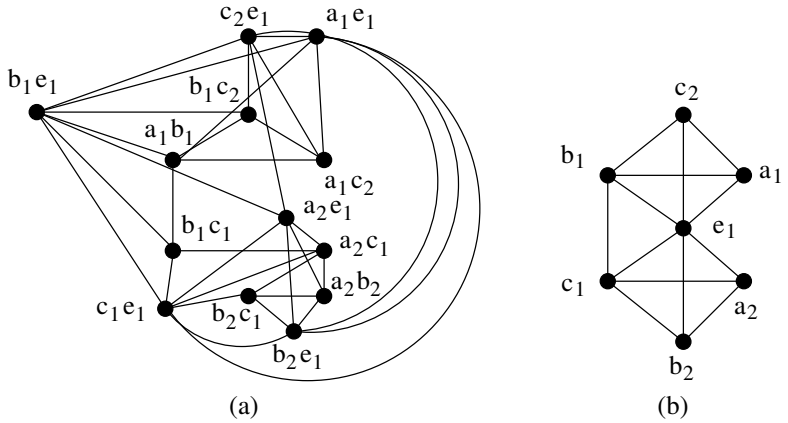

FIGURE 6.5 – Morceau d'un CSP non BTP mais vérifiant DBTP, la cohérence d'arc et l'inter-cohérence.

Une partie du théorème suivant se déduit immédiatement de ce qui précède :

Théorème 6.7. *$BTP^{AC} \cap DBTP^{AC} \neq \emptyset$ et $BTP^{AC} \perp DBTP^{AC}$.*

Preuve : Pour montrer que l'intersection entre les deux classes est non vide, il suffit de considérer un CSP P monovalent et cohérent à trois variables et trois contraintes. P satisfait BTP, DBTP et évidemment BTP^{AC} et $DBTP^{AC}$. Pour montrer que les deux classes sont différentes, l'exemple de la figure 6.5 appartient à $DBTP^{AC}$ mais pas à BTP^{AC}. Si nous considérons l'exemple de la figure 6.9, il appartient à BTP^{AC} mais pas à $DBTP^{AC}$. □

Nous pouvons notamment démontrer qu'un CSP binaire chemin-cohérent et DBTP est BTP :

Théorème 6.8. *Si un CSP binaire P vérifie DBTP par rapport à un ordre $\prec$ sur les contraintes et s'il vérifie la cohérence de chemin, alors P vérifie BTP quelque soit l'ordre sur les variables.*

Preuve : Nous supposons que P ne vérifie pas BTP. Alors, tout ordre sur les variables ne permet pas d'avoir cette propriété. Nous en choisissons un quelconque (sans aucune précision) et nous montrerons que ce cas là ne pourra exister. Pour un ordre $<$ sur les variables, il existe un triplet $x_i < x_j < x_k$ tel que $\exists v_i \in D(x_i), v_j \in D(x_j)$ et $v_k, v'_k \in D(x_k)$,
– $(v_i, v_j) \in R(c_{ij})$ (on note $t_1 = (v_i, v_j)$),
– $(v_i, v_k) \in R(c_{ik})$ (on note $t_2 = (v_i, v_k)$),

– $(v_j, v'_k) \in R(c_{jk})$ (on note $t_3 = (v_j, v'_k)$),
– $(v_j, v_k) \notin R(c_{jk})$ et
– $(v_i, v'_k) \notin R(c_{ik})$.

Comme P vérifie la cohérence de chemin, $\exists v'_i \in D(x_i), v'_j \in D(x_j)$ et $v''_k \in D(x_k)$ tel que
– $(v_i, v'_j) \in R(c_{ij})$ (on note $t'_1 = (v_i, v'_j)$),
– $(v'_i, v_j) \in R(c_{ij})$ (on note $t''_1 = (v'_i, v_j)$),
– $(v'_i, v'_k) \in R(c_{ik})$ (on note $t'_2 = (v'_i, v'_k)$),
– $(v_i, v''_k) \in R(c_{ik})$ (on note $t''_2 = (v_i, v''_k)$),
– $(v'_j, v_k) \in R(c_{jk})$ (on note $t'_3 = (v'_j, v_k)$) et
– $(v_j, v''_k) \in R(c_{jk})$ (on note $t''_3 = (v_j, v''_k)$).

Par conséquent, il est facile de voir qu'il n'y a aucun ordre sur les contraintes (à cause de la présence d'un triangle cassé sur chaque contrainte (t_2, t_1, t_3, t'_2) sur c_{ik}, $(t_1, t''_2, t''_3, t''_1)$ sur c_{ij} et (t_3, t_1, t_2, t'_3) sur c_{jk}) pour que P satisfasse DBTP, et donc chaque fois qu'on choisit un ordre il est impossible d'avoir DBTP ce qui nous a conduit à déduire que P ne satisfait pas DBTP. Ainsi, nous obtenons une contradiction et P satisfait BTP. □

Nous étudions maintenant le cas des CSP acycliques pour lesquels [Cooper et al., 2010] a déjà prouvé que ces CSP binaires vérifient BTP. Nous allons montrer que cela est également vrai pour DBTP. Soit $TREE$ l'ensemble des CSP binaires dont le graphe de contraintes est acyclique.

Théorème 6.9. $TREE \subsetneq DBTP$.

Preuve : Soit DUAL-$TREE$ l'ensemble des CSP binaires dont chaque élément est le dual d'un CSP de $TREE$. Comme cela a été montré dans [Cooper et al., 2010], DUAL-$TREE \subsetneq BTP$. Par conséquent, $TREE \subsetneq DBTP$. □

Ce résultat peut être étendu aux CSP d'arité quelconque. Pour cela, nous devons considérer la notion de cyclicité dans les hypergraphes, pour lesquels différents degrés ont été définis [Fagin, 1983]. Ici, nous nous intéressons en particulier à l'α-acyclicité et à la β-acyclicité. Nous allons montrer que les CSP β-acycliques vérifient DBTP, alors que ce n'est pas le cas pour les CSP α-acycliques.

Il a récemment été montré dans [Duris, 2012] que les hypergraphes β-acycliques peuvent être définis en appliquant les deux règles suivantes, qui doivent mener à l'hypergraphe vide :

(1) Si une hyperarête est vide, elle est retirée de E.

(2) Si un sommet est un de type « nest » (i.e. l'ensemble des hyperarêtes le contenant est une chaîne pour la relation d'inclusion), alors il est retiré de H (i.e. il est retiré de V ainsi que des hyperarêtes qui le contiennent).

Théorème 6.10 ([Duris, 2012])**.** *Un hypergraphe H est β-acyclique ssi, après l'application successive des règles (1) et (2) jusqu'à ce qu'aucune ne puisse l'être, nous obtenons l'hypergraphe vide.*

En utilisant ces définitions, nous pouvons maintenant établir le théorème suivant :

Théorème 6.11. *Étant donné un CSP (X, D, C), il existe un ordre sur les contraintes $\prec$, tel que $\forall c_i, c_j, c_k \in C$ tel que $c_i \prec c_j \prec c_k$, nous avons $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ ou $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$ ssi (X, D, C) possède un hypergraphe de contraintes β-acyclique.*

Preuve : ($\Rightarrow$) Par contraposition. Nous montrons que si l'hypergraphe de contraintes (X, C) est β-cyclique, alors, il ne peut exister d'ordre sur les contraintes.

Considérons un hypergraphe de contraintes (X, C) qui est β-cyclique. Il possède donc un cycle de Graham, que l'on notera par la séquence d'hyperarêtes $(c_1, ..., c_m, c_{m+1})$. Considérons un ordre quelconque sur les contraintes $\prec$. Nécessairement, parmi les contraintes de ce cycle, il existe une contrainte maximum c_k par rapport à l'ordre $\prec$. Considérons ses deux voisines dans le cycle, notées

c_i et c_j (avec $c_i, c_j \prec c_k$). Par définition des cycles de Graham, nous savons que $S(c_i) \cap S(c_k)$ et $S(c_j) \cap S(c_k)$ sont incomparables. Aussi, nous n'avons ni $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ ni $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$, et donc aucun ordre correct sur les contraintes $\prec$ ne peut exister.

($\Leftarrow$) Nous utilisons ici le théorème 6.10. Étant donné un CSP β-acyclique (X, D, C) d'hypergraphe H qui admet un ordre $\prec$ correct sur les contraintes, nous allons montrer que :

(1) Une hyperarête $S(c_i)$ est vide ssi H sans $S(c_i)$ admet un ordre et est β-acyclique.

(2) Un sommet x de H est un point de type "nest" tel que H admet un ordre ssi H sans x admet un ordre et est β-acyclique.

On voit immédiatement que la propriété est vérifiée par l'application de la règle (1). Considérons donc la règle (2). Supposons que pour un hypergraphe H nous disposons d'un ordre $\prec$. Aussi, $\forall c_i, c_j, c_k \in C$ tel que $c_i \prec c_j \prec c_k$, nous avons $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ ou $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$. Nous avons 5 cas à considérer :

1. $x \notin S(c_i) \cup S(c_j) \cup S(c_k)$: après la suppression de x, ni $S(c_i) \cap S(c_k)$, ni $S(c_j) \cap S(c_k)$ n'ont changé. La propriété est donc vérifiée.
2. $x \in S(c_i) \cap S(c_j) \cap S(c_k)$: après la suppression de x, ce sommet disparaît de chaque intersection $S(c_i) \cap S(c_k)$ et $S(c_j) \cap S(c_k)$, et ainsi, la propriété est vérifiée.
3. x n'appartient qu'à un seul des ensembles $S(c_i)$ ou $S(c_j)$ ou $S(c_k)$: donc x n'appartient à aucune intersection, et donc la propriété est vérifiée après la suppression.
4. $x \in S(c_i) \cap S(c_j)$ et $x \notin S(c_k)$: donc x n'appartient ni à $S(c_i) \cap S(c_k)$, ni à $S(c_j) \cap S(c_k)$, et donc la propriété est vérifiée après la suppression.
5. $x \in S(c_i) \cap S(c_k)$ et $x \notin S(c_j)$ (ou symétriquement $x \in S(c_j) \cap S(c_k)$ et $x \notin S(c_i)$) : donc, avant la suppression, nous avions nécessairement $S(c_i) \cap S(c_k) \not\subseteq S(c_j) \cap S(c_k)$ et $S(c_j) \cap S(c_k) \subsetneq S(c_i) \cap S(c_k)$. Ainsi, après la suppression de x, nous avons au moins $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$.

Donc, nous avons montré que si nous pouvons réduire tout hypergraphe, ce qui ne contredit pas la propriété sur l'ordre, et donc, nécessairement, l'hypergraphe d'origine est β-acyclique et admet un ordre approprié sur les contraintes. □

Nous pouvons noter que ce théorème explique pourquoi la condition énoncée dans le lemme 4.6 de [Cooper et al., 2010] (lemme 6.7 dans ce manuscrit) est vérifiée indépendamment de la portée des contraintes.

Lemme 6.7. *Soit P un CSP (d'arité quelconque) ayant un ensemble des contraintes $\{c_1, ..., c_e\}$. Le dual de P satisfait BTP par rapport à un ordre $<$ sur les contraintes, si et seulement si, $\forall c_i, c_j$ et c_k avec $i < j < k$, on a :*
$(S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)) \vee (S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k))$

Ce lemme et le théorème précédent nous permettent d'obtenir le théorème 6.12 où β-$ACYCLIC$ est l'ensemble des CSP pour lesquels l'(hyper)graphe de contraintes est β-acyclique.

Théorème 6.12. $TREE \subsetneq \beta\text{-}ACYCLIC \subsetneq DBTP$.

Toutefois, l'équivalence proposée dans le lemme 4.6 [Cooper et al., 2010] est seulement vérifiée dans le sens inverse (ce qui ne compromet pas la preuve du théorème 6.12). Pour s'en rendre compte, il suffit de considérer un CSP binaire avec 3 variables monovalentes (une seule valeur par domaine) mutuellement connectées (chaque contrainte admet l'unique tuple possible). Son dual satisfait BTP bien que le graphe de contraintes ne soit pas β-acyclique. Cet exemple de CSP peut bien entendu être généralisé à des tailles de domaines et à un nombre de variables quelconques en reproduisant un motif identique.

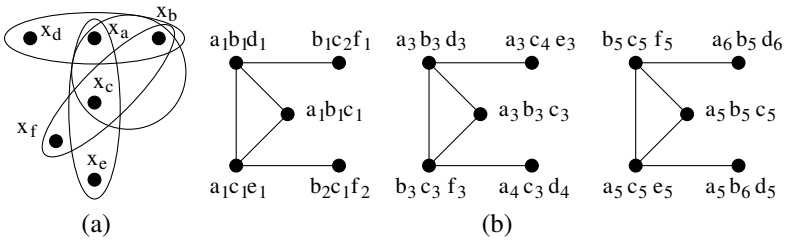

FIGURE 6.6 – Un CSP α-acyclique (a) qui ne vérifie pas DBTP (b).

Nous montrons maintenant que si α-$ACYCLIC$ est l'ensemble des CSP dont l'(hyper)graphe de contraintes est α-acyclique, alors les ensembles α-$ACYCLIC$ et $DBTP$ sont incomparables.

Considérons un CSP possédant six variables $x_a, \ldots x_f$ et quatre contraintes dont les portées sont respectivement $\{x_a, x_b, x_c\}$, $\{x_a, x_b, x_d\}$, $\{x_a, x_c, x_e\}$ et $\{x_b, x_c, x_f\}$. La figure 6.6 présente son hypergraphe de contraintes (a) et la microstructure de son dual (b). Nous pouvons constater que ce CSP est α-acyclique mais qu'elle ne vérifie pas DBTP puisqu'aucun ordre approprié sur les contraintes ne peut exister. De plus, il est bien connu que β-$ACYCLIC \subsetneq \alpha$-$ACYCLIC$ [Beeri et al., 1983]. Donc, nous obtenons le théorème suivant :

Théorème 6.13. *α-$ACYCLIC \cap DBTP \neq \emptyset$ et α-$ACYCLIC \perp DBTP$.*

Dans cette partie, nous avons donc globalement établi le résultat suivant :

Théorème 6.14. *$BTP \cap DBTP \neq \emptyset$ et $BTP \perp DBTP$.*

Pour le cas non-binaire, la figure 6.7 montre les différentes relations qui existent entre DBTP, BTP et certaines de leurs classes cachées.

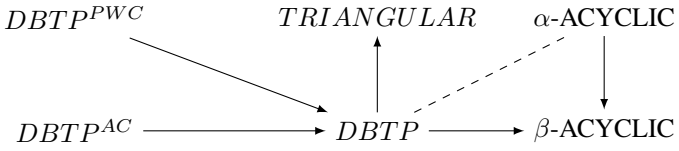

FIGURE 6.7 – Relation entre DBTP et ses classes cachées dans le cas non-binaire.

Pour le cas binaire, la figure 6.8 montre également les différentes relations qui existent entre DBTP, BTP et certaines de leurs classes cachées.

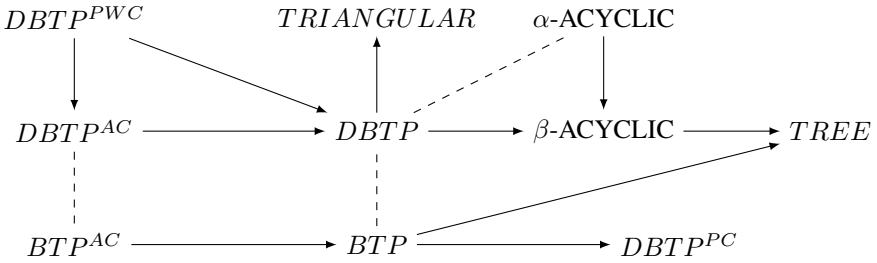

FIGURE 6.8 – Relation entre DBTP, BTP et leurs classes cachées dans le cas binaire.

Dans la partie suivante, nous étudierons le lien entre DBTP et quelques autres classes polynomiales.

6.4 DBTP vs quelques classes polynomiales

Dans cette section, nous comparerons DBTP à certaines classes polynomiales connues introduites pour les CSP binaires et n-aires.

6.4.1 Cas des CSP binaires

Comme $DBTP$ et BTP sont deux classes différentes, nous nous concentrons d'abord sur certaines classes polynomiales incluses dans BTP. Ces classes s'appuient sur des restrictions de langages de contraintes.

Le théorème suivant montre que les classes polynomiales RC (row-convex), ZOA (0-1-tous) et RRM (renamable right monotone) partagent des CSP avec DBTP mais sont cependant différentes.

Théorème 6.15. *$DBTP \cap RC \neq \emptyset$ et $DBTP \perp RC$.*
$DBTP \cap ZOA \neq \emptyset$ et $DBTP \perp ZOA$.
$DBTP \cap RRM \neq \emptyset$ et $DBTP \perp RRM$.

Preuve : Si l'on considère le CSP binaire de la figure 6.9(a), il est *0-1-tous*, *convexe par rangée*, et *renommable monotone à droite* par rapport aux ordres lexicographiques sur les valeurs et les variables. Cependant, comme le montre la figure 6.9(b), ce CSP n'est pas DBTP à cause des triangles cassés suivants :

- $((a_1, b_2), (b_2, c_2), (a_2, c_2), (a_2, b_1))$ sur c_{ab},
- $((a_1, c_2), (a_1, b_2), (b_2, c_1), (a_2, c_1))$ sur c_{ac} et
- $((b_1, c_2), (a_2, b_1), (a_2, c_2), (a_2, c_1))$ sur c_{bc}.

Inversement, tout CSP non-binaire DBTP ne peut appartenir à RC, ZOA ou RRM.

Afin de prouver que $DBTP$ intersecte RC, ZOA et RRM, il suffit de considérer un CSP binaire monovalent à trois variables et trois contraintes qui est cohérent. En effet, ce type de CSP satisfait à la fois $DBTP$, RC, ZOA et RRM car ces classes polynomiales (sauf DBTP) couvrent seulement les CSP binaires. □

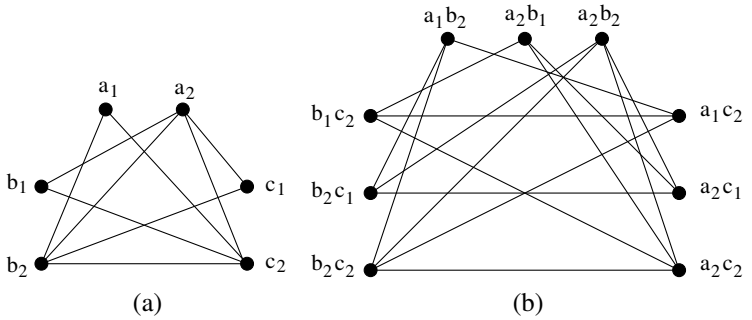

FIGURE 6.9 – Un CSP appartenant à RC, ZOA et RRM (a) mais qui ne vérifie pas DBTP (b).

Certaines classes traitables sont relatives à la microstructure, c'est le cas de CM (chordal microstructure), CCM (complement chordal microstructure) et PM (perfect microstructure). Nous montrerons que ces classes sont différentes de DBTP.

Théorème 6.16. *$CM \cap DBTP \neq \emptyset$ et $CM \perp DBTP$.*

Preuve : Tout CSP binaire cohérent et monovalent a une microstructure triangulée et est DBTP. Donc, l'intersection entre les deux classes n'est pas vide. Si on considère le CSP décrit par la figure 6.9(a), il a une microstructure triangulée mais il n'est pas DBTP. Réciproquement, tout CSP DBTP non-binaire n'appartient pas à CM. □

Théorème 6.17. *$CCM \cap DBTP \neq \emptyset$ et $CCM \perp DBTP$.*

Preuve : Tout CSP binaire cohérent et monovalent a un complément de microstructure triangulé et est DBTP. Donc, l'intersection entre les deux classes n'est pas vide. Si on considère le CSP décrit par la figure 6.9(a), il a un complément de microstructure triangulé mais il n'est pas DBTP. Réciproquement, tout CSP satisfaisant DBTP non-binaire n'appartient pas à CCM. □

Les graphes triangulés sont aussi parfaits, donc nous pouvons dériver un résultat similaire pour la classe des CSP ayant une microstructure parfaite [Salamon and Jeavons, 2008].

Théorème 6.18. *$PM \cap DBTP \neq \emptyset$ et $PM \perp DBTP$.*

La classe suivante repose sur le nombre de cliques maximales figurant dans la microstructure :

Théorème 6.19. *$CL \cap DBTP \neq \emptyset$ et $CL \perp DBTP$.*

Preuve : Tout CSP binaire monovalent et cohérent possède une seule clique maximale et il est DBTP. Donc, l'intersection n'est pas vide.

Considérons maintenant l'exemple de la figure 6.3(b), ce CSP n'est pas DBTP (présence d'un triangle cassé sur chaque contrainte) mais il a un nombre polynomial de cliques maximales dans sa microstructure (3 cliques maximales de taille 3). Inversement, tout CSP non-binaire satisfaisant $DBTP$ ne peut appartenir à CL. □

En ce qui concerne les classes basées sur des structures restreintes, nous avons prouvé dans le théorème 6.12 que $TREE < DBTP$.

6.4.2 CSP d'arité quelconque

Nous considérons d'abord certaines classes polynomiales connues basées sur des restrictions de langages de contraintes comme la classe MC (max-closed) et $IFUN$ (incrementally functional).

Théorème 6.20. *$MC \cap DBTP \neq \emptyset$ et $MC \perp DBTP$.*

Preuve : La preuve de $MC \cap DBTP \neq \emptyset$ et de $MC \not\subseteq DBTP$ est similaire à celle du théorème 6.15. En ce qui concerne $DBTP \not\subseteq MC$, tout CSP ayant deux variables et une contrainte binaire est DBTP mais pas nécessairement *max-closed* (voir figure 2.16). □

Théorème 6.21. *$IFUN \cap DBTP \neq \emptyset$ et $IFUN \perp DBTP$.*

Preuve : Afin de prouver que l'intersection n'est pas vide, nous considérons un CSP avec quatre variables monovalentes $x_1, \ldots, x_4$ et trois contraintes ternaires c_1, c_2 et c_3 telles que $S(c_1) = \{x_1, x_2, x_3\}$, $R(c_1) = \{(v_1, v_2, v_3)\}$, $S(c_2) = \{x_1, x_2, x_4\}$, $R(c_2) = \{(v_1, v_2, v_4)\}$, $S(c_3) = \{x_2, x_3, x_4\}$ et $R(c_3) = \{(v_2, v_3, v_4)\}$. Ce CSP est *incrémentalement fonctionnel* (en utilisant la numérotation des variables comme ordre) et DBTP. Le CSP présenté dans la figure 6.10 est *incrémentalement fonctionnel* mais pas DBTP. Inversement, tout CSP satisfaisant DBTP ayant plusieurs solutions ne peut pas être *incrémentalement fonctionnel*. □

Nous comparons maintenant DBTP avec la classe polynomiale des CSP ayant un nombre polynomial de cliques maximales dans la DR-microstructure (DCL).

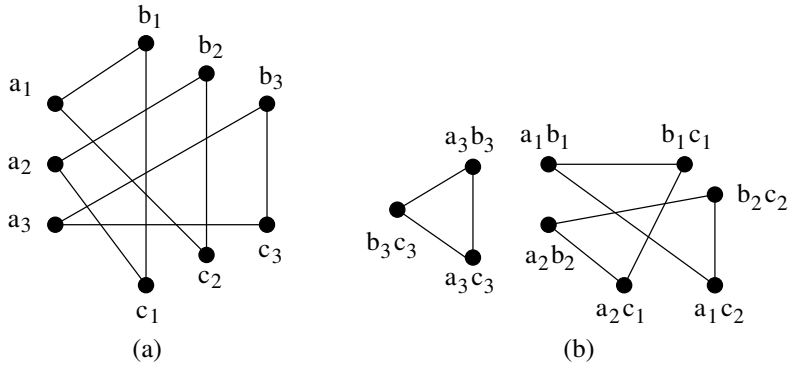

FIGURE 6.10 – Un CSP *incrémentalement fonctionnel* (a) mais qui ne vérifie pas DBTP (b).

Théorème 6.22. $DCL \cap DBTP \neq \emptyset$ *et* $DCL \perp DBTP$.

Preuve : Considérons le premier CSP défini dans la preuve du théorème 6.21. La microstructure de son CSP dual possède une seule clique maximale et le CSP est DBTP. Ainsi, l'intersection n'est pas vide. En ce qui concerne l'exemple représenté dans la figure 6.10, elle possède un nombre polynomial de cliques maximales dans la microstructure de son CSP dual mais n'est pas DBTP. Inversement, un CSP binaire dont le graphe de contraintes est une étoile et pour lequel chaque domaine dispose de plusieurs valeurs est DBTP mais possède un nombre de cliques maximales dans sa microstructure duale non borné polynomialement. □

Maintenant, nous introduisons une nouvelle classe polynomiale basée sur une restriction du langage de contraintes.

Définition 6.2 (Triangulaire)**.** *Un* $P = (X, D, C)$ *est dit* ***triangulaire*** *par rapport à un ordre sur les contraintes* $\prec$ *ssi* $\forall c_i, c_j, c_k, c_i \prec c_j \prec c_k$, $\forall t_i \in R(c_i), t_j \in R(c_j), t_k \in R(c_k)$, *si*
- $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$ *et*
- $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$.

alors, $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$.
Nous notons TR *l'ensemble de ces CSP.*

Théorème 6.23. *Si un CSP* P *est triangulaire par rapport à un ordre* $\prec$*, alors* P *vérifie DBTP par rapport à* $\prec$.

Preuve : Supposons que P soit triangulaire mais pas DBTP. Alors, il existe trois contraintes c_i, c_j et c_k, $c_i \prec c_j \prec c_k$, $t_i \in R(c_i)$, $t_j \in R(c_j)$ et $t_k, t'_k \in R(c_k)$ telles que $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$, $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$, $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$, $t'_k[S(c_i) \cap S(c_k)] \neq t_i[S(c_i) \cap S(c_k)]$ et $t_j[S(c_j) \cap S(c_k)] \neq t_k[S(c_j) \cap S(c_k)]$. Comme P est triangulaire par rapport à l'ordre $\prec$, nous devons avoir $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$ et $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$, ce qui n'est pas possible puisque P ne vérifie pas DBTP. □

Théorème 6.24. $TR \subsetneq DBTP$.

Preuve : Le théorème 6.23 montre que $TR < DBTP$. Le CSP présenté dans la figure 6.11 vérifie DBTP (il existe deux ordre pour que ce CSP satisfasse DBTP $c_{ab} < c_{bc} < c_{ac}$ et $c_{bc} < c_{ab} < c_{ac}$) mais n'est pas triangulaire. □

En ce qui concerne les classes basées sur des restrictions de structures, nous avons prouvé dans les théorèmes 6.12 et 6.13 que $\beta\text{-}ACYCLIC \subsetneq DBTP$, $\alpha\text{-}ACYCLIC \cap DBTP \neq \emptyset$ et $\alpha\text{-}ACYCLIC \perp DBTP$.

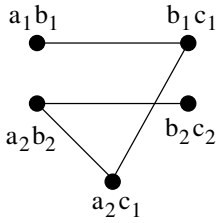

FIGURE 6.11 – Un CSP vérifiant DBTP mais qui n'est pas triangulaire.

Une autre classe polynomiale importante basée sur une restriction de structure porte sur la largeur d'arbre.

Théorème 6.25. $BTW_1 \subsetneq DBTP$.
Pour $k > 1$, $BTW_k \cap DBTP \neq \emptyset$ et $BTW_k \perp DBTP$.

Preuve : Il est bien connu que BTW_1 est l'ensemble de CSP binaires ayant une structure arborescente. D'après le théorème 6.9, nous avons $BTW_1 \subsetneq DBTP$.

Pour $k > 1$, comme $BTW_1 \subsetneq BTW_k$, l'intersection $BTW_k \cap DBTP$ est non vide. Maintenant, nous considérons un CSP, ayant n variables avec $n \geq 3$, dont la largeur arborescent est bornée par une constante $k \geq 2$ et qui contient un sous-problème illustré par la figure 6.10(a). CeCSP a une largeur arborescente bornée mais ne satisfait pas DBTP. Réciproquement, tout CSP ayant n variables et une contrainte d'arité n est DBTP mais possède une largeur arborescente non bornée. Par conséquent, $BTW_k \perp DBTP$. □

6.5 (D)BTP et l'(Hyper-)k-Cohérence

Dans cette partie, nous étudions les liens existants entre (D)BTP et l'(hyper-)k-cohérence directionnelle. Cette étude nous semble naturelle dans la mesure où, comme évoqué dans [Cooper et al., 2010], BTP est une propriété moins forte que l'hyper-3-cohérence [Jégou, 1993b].

Comme l'hyper-3-cohérence qui implique BTP, est trop forte, nous définissons une forme relâchée en prenant en compte un ordre sur les contraintes :

Définition 6.3 (Hyper-k-Cohérence Directionnelle)**.** *Étant donnés un CSP $P = (X, D, C)$, un ordre sur les contraintes $\prec$ et un entier k tel que $1 \leq k \leq e$, P vérifie l'**hyper-k-cohérence directionnelle** si pour tout sous-ensemble de k contraintes tel que $c_1 \prec c_2 \prec \cdots \prec c_{k-1} \prec c_k$, on a*

$$\bowtie_{i=1}^{k-1} R(c_i)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)] \subseteq R(c_k)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)]$$

Théorème 6.26. *Si un CSP P vérifie DBTP par rapport à un ordre $\prec$ et qu'il est hyper-k-cohérent directionnel par rapport à $\prec$ pour $2 \leq k < e$, alors P est hyper-$(k+1)$-cohérent directionnel par rapport à $\prec$.*

Preuve : Supposons que P ne soit pas hyper-$(k+1)$-cohérent. Alors, il existe un sous-ensemble de $k+1$ contraintes tel que $c_1 \prec \cdots \prec c_k \prec c_{k+1}$, $\exists (t_1, \ldots, t_k) \in R(c_1) \times \ldots \times R(c_k), \forall t_{k+1} \in R(c_{k+1})$, $\bowtie_{i=1}^{k} \; t_i[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})] \neq t_{k+1}[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})]$. Considérons les k sous-ensembles de k contraintes $\{c_1, \ldots c_{j-1}, c_{j+1}, \ldots c_k, c_{k+1}\}$, pour $1 \leq j \leq k$. Comme P est hyper-k-cohérent directionnel, pour $1 \leq j \leq k$, il existe un tuple t_{k+1}^j of $R(c_{k+1})$ tel

que $\bowtie_{i=1,i\neq j}^{k} t_i[(\bigcup_{i=1,i\neq j}^{k} S(c_i)) \cap S(c_{k+1})] = t_{k+1}^{j}[(\bigcup_{i=1,i\neq j}^{k} S(c_i)) \cap S(c_{k+1})]$. Considérons $1 \leq j < j' \leq k$. Nous avons deux cas :

(1) $t_{k+1}^{j} = t_{k+1}^{j'}$. Alors
– $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] = t_{k+1}^{j}[S(c_{j'}) \cap S(c_{k+1})]$ et
– $t_j[S(c_j) \cap S(c_{k+1})] = t_{k+1}^{j'}[S(c_j) \cap S(c_{k+1})]$.
Donc $\bowtie_{i=1}^{k} t_i[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})] = t_{k+1}^{j}[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})]$ et nous avons une contradiction.

(2) $t_{k+1}^{j} \neq t_{k+1}^{j'}$. Nous avons :
– $t_j[S(c_j) \cap S(c_{j'})] = t_{j'}[S(c_j) \cap S(c_{j'})]$,
– $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] = t_{k+1}^{j}[S(c_{j'}) \cap S(c_{k+1})]$,
– $t_j[S(c_j) \cap S(c_{k+1})] = t_{k+1}^{j'}[S(c_j) \cap S(c_{k+1})]$,
– $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] \neq t_{k+1}^{j'}[S(c_{j'}) \cap S(c_{k+1})]$ et
– $t_j[S(c_j) \cap S(c_{k+1})] \neq t_{k+1}^{j}[S(c_j) \cap S(c_{k+1})]$.
Par conséquent P ne vérifie pas DBTP par rapport à $\prec$ et nous avons encore une contradiction.

Donc, P est hyper-$(k+1)$-cohérent directionnel par rapport à l'ordre $\prec$. □

Comme l'inter-cohérence correspond à l'hyper-2-cohérence, nous pouvons en déduire ce corollaire :

Corollaire 6.1. *Si un CSP P vérifie DBTP par rapport à un ordre $\prec$ sur les contraintes et qu'il est inter-cohérent directionnel par rapport à $\prec$, alors P est hyper-$(k+1)$-cohérent directionnel par rapport à $\prec$ pour $1 \leq k < e$.*

Donc, un CSP qui est à la fois DBTP et inter-cohérent directionnel par rapport à un ordre donné sur les contraintes est cohérent. Il s'ensuit également qu'un tel CSP vérifie l'hyper-3-cohérence directionnel.

De plus, comme l'hyper-$(k+1)$-cohérence correspond à la k-cohérence sur le problème dual, nous pouvons également en déduire le théorème et le corollaire suivants en appliquant un raisonnement similaire dans le cas de BTP et des CSP binaires.

Théorème 6.27. *Si un CSP binaire P vérifie BTP par rapport à un ordre $<$ sur les variables et qu'il est k-cohérent directionnel par rapport à $<$ pour $2 \leq k < n$, alors P est $(k+1)$-cohérent directionnel par rapport à $<$.*

Corollaire 6.2. *Si un CSP binaire P vérifie BTP par rapport à un ordre $<$ sur les variables et qu'il est k-cohérent directionnel par rapport à $<$, alors P est $(k+1)$-cohérent directionnel par rapport à $<$ pour $1 \leq k < n$.*

Comme conséquence, un CSP binaire qui est à la fois BTP et DAC (Directionnel Arc cohérent [Wallace, 1994]) par rapport à un ordre donné sur les variables est cohérent.

En ce qui concerne le positionnement de l'hyper-3-cohérence directionnelle par rapport à BTP, nous pouvons prouver que l'hyper-3-cohérence n'implique pas nécessairement BTP. Par exemple, nous pouvons considérer un CSP binaire avec six variables $\{x_a, x_b, \ldots, x_f\}$. Nous définissons ce CSP en reproduisant plusieurs fois un même motif tel que chaque valeur apparaissant dans une occurrence du motif n'apparaisse dans aucune autre de ses occurrences. Ce motif consiste en un triangle cassé sur une variable z pour un triplet (x, y, z) (i.e. ce qui impose la condition $z < max(x, y)$ sur $<$) et chaque valeur des variables x, y et z est liée à une valeur donnée d'une variable qui n'est pas impliquée dans ce triplet. Nous reproduisons ce motif six fois de telles sorte que les conditions suivantes soient vérifiées :

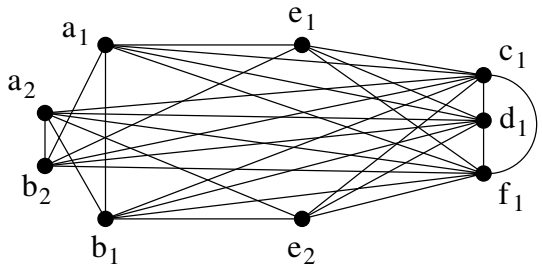

FIGURE 6.12 – Partie d'un CSP vérifiant l'hyper-3-cohérence directionnelle mais pas BTP.

- $x_a < max(x_b, x_c)$,
- $x_b < max(x_d, x_e)$,
- $x_c < max(x_e, x_f)$,
- $x_d < max(x_a, x_b)$,
- $x_e < max(x_a, x_b)$ et
- $x_f < max(x_a, x_b)$.

La figure 6.12 présente ce motif pour les triplets (x_a, x_b, x_e), un triangle cassé sur x_e (correspondant à la condition $x_e < max(x_a, x_b)$) et sur les variables indépendantes x_c, x_d et x_f. En faisant cela, la microstructure de notre CSP binaire possède six composantes connexes. On peut noter que ce CSP n'est pas BTP parce que les six conditions rendent impossibles la construction d'un ordre approprié sur les variables. Néanmoins, il est hyper-3-cohérent directionnel ainsi qu'arc-cohérent.

6.6 DBTP d'un point de vue pratique

Dans cette section, nous allons étudier DBTP d'un point de vue pratique. D'abord, nous montrerons que certains benchmarks utilisés souvent pour tester les solveurs sont DBTP. Ensuite, nous montrerons que ces benchmarks peuvent être résolus efficacement par des algorithmes comme MAC et RFL.

6.6.1 DBTP dans les benchmarks

Vérifier si un CSP P satisfait la propriété DBTP peut être réalisé en testant l'existence d'un ordre $\prec$ sur les contraintes pour lequel P satisfait DBTP. Pour cela, nous rappelons que la preuve du lemme 6.2 propose une méthode qui consiste à calculer le dual de P et tester si un ordre sur les contraintes pour lequel le dual de P satisfait BTP existe comme dans [Cooper et al., 2010]. Cependant, en pratique, la construction du dual peut engendrer un coût prohibitif en mémoire surtout quand la contrainte admet un nombre important de tuples. De plus, si les contraintes sont exprimées en intension, le calcul du CSP dual exigera souvent un espace mémoire plus grand que celui des CSP dont les contraintes sont définies en extension.

Heureusement, nous pouvons éviter de construire le dual en le considérant d'une manière virtuelle et en exploitant la caractérisation de DBTP énoncée dans le théorème 6.1. En effet, dans cette caractérisation, nous avons besoin de manipuler seulement les interactions entre les tuples d'un triplet de contraintes. Ces interactions sont directement exprimées dans le CSP dual mais elles peuvent être déduites à la volée en tenant compte de l'intersection entre les tuples de contraintes différents. La deuxième étape peut être réalisée en construisant et en résolvant le CSP max-closed avec une variable o_i (dont le domaine $\{1, \ldots, e\}$) pour chaque contrainte c_i de P et en imposant

Benchmark	n	e	d
hanoi/normalized-hanoi-3_ext	6	5	27
hanoi/normalized-hanoi-4_ext	14	13	81
hanoi/normalized-hanoi-5_ext	30	29	243
hanoi/normalized-hanoi-6_ext	62	61	729
hanoi/normalized-hanoi-7_ext	126	125	2 187
rlfapGraphsMod/normalized-graph12-w0	680	340	44
rlfapGraphsMod/normalized-graph13-w0	916	458	44

TABLE 6.1 – Liste des benchmarks binaires qui ont été détectés comme étant DBTP.

une contrainte de type $o_k < max(o_i, o_j)$ pour tous les triplets de contraintes (c_i, c_j, c_k) tels que $\exists t_i \in R(c_i)$, $t_j \in R(c_j)$ et $t_k, t'_k \in R(c_k)$,
- $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$,
- $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$,
- $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$,
- $t'_k[S(c_i) \cap S(c_k)] \neq t_i[S(c_i) \cap S(c_k)]$ et
- $t_j[S(c_j) \cap S(c_k)] \neq t_k[S(c_j) \cap S(c_k)]$.

Nous nous intéressons ici aux 7272 benchmarks de la compétition CSP 2008. Dans notre expérimentation, nous avons exclu les benchmarks contenant des contraintes globales car notre bibliothèque CSP ne les prend pas en compte. Pour le reste des benchmarks, et pour des raisons de temps, nous avons seulement considéré 2530 benchmarkss dont les contraintes sont binaires ou non-binaires exprimées en extension (par la liste des tuples autorisés ou interdits) ou en intension.

Si nous voulons tester directement l'existence de DBTP, seuls les CSP arborescents ou β-acycliques sont DBTP.

En fait, nous trouvons 7 benchmarks binaires (voir table 6.1) dont le graphe de contraintes est arborescent (par exemple tous les benchmarks de la famille `hanoi`) et 23 benchmarks non-binaires dont l'hypergraphe de contraintes est β-acyclique (voir table 6.2).

L'absence des CSP dans lesquelles la propriété DBTP réside est due dans certains cas à la présence des valeurs inutiles (incohérentes) dans les domaines ou des tuples inutiles dans les relations. Pour éviter ce problème, une solution consiste à simplifier le CSP en appliquant certains niveaux de cohérence comme la cohérence d'arc. Le choix de la cohérence d'arc peut se justifier de ce niveau de cohérence par l'exploitation de la plupart des solveurs.

En appliquant la cohérence d'arc, 362 ont été trivialement détectées comme étant $DBTP^{AC}$ parce qu'elles étaient arc-incohérentes et avaient des domaines tous vides (nous supposons que le processus de filtrage ne s'arrête pas quand un domaine devient vide). Nous avons trouvé également 103 benchmarks qui sont à la fois arc-cohérents et $DBTP^{AC}$. Pour les CSP binaires, 36 benchmark satisfont $DBTP^{AC}$. Nous pouvons noter que tous les benchmark des familles `domino` et `hanoi` sont $DBTP^{AC}$. En plus, les benchmarks des familles `hanoi` et `rlfapGraphsMod` sont DBTP car leur graphe de contraintes est arborescent.

Finalement, nous avons observé que les cinq premiers benchmarks de la table 6.1 sont BTP et les deux dernières (de la table 6.1) avec tous les benchmarks de la table 6.3 sont BTP^{AC}. Par contre, tous les benchmarks de la compétition qui satisfont BTP ne sont pas forcément DBTP. Par exemple, le benchmark `fapp/fapp17/normalized-fapp17-0300-10` satisfait BTP mais il n'est pas DBTP. En regardant les benchmarks non-binaires, nous pouvons facilement constater que les benchmarks de la famille `mknap` sont trivialement DBTP vu que chacun d'entre eux contient seulement une contrainte. Nous pouvons aussi observer que plus que 33% des benchmarks de la famille `primes-*` sont $DBTP^{AC}$ et que 10 benchmarks de cette famille le sont car elles ont

Benchmark	n	e	d	r
mknap/normalized-mknap-1-0	6	1	2	6
mknap/normalized-mknap-1-2	15	1	2	15
mknap/normalized-mknap-1-3	20	1	2	20
mknap/normalized-mknap-1-4	28	1	2	28
mknap/normalized-mknap-1-5	39	1	2	39
mknap/normalized-mknap-1-6	50	1	2	50
primes-10/normalized-primes-10-20-2-1	100	20	28	3
primes-10/normalized-primes-10-20-3-1	100	20	28	4
primes-15/normalized-primes-15-20-2-1	100	20	46	3
primes-15/normalized-primes-15-20-3-1	100	20	46	4
primes-20/normalized-primes-20-20-2-1	100	20	70	3
primes-20/normalized-primes-20-20-3-1	100	20	70	4
primes-25/normalized-primes-25-20-2-1	100	20	96	3
primes-25/normalized-primes-25-20-3-1	100	20	96	4
primes-30/normalized-primes-30-20-2-1	100	20	112	3
primes-30/normalized-primes-30-20-3-1	100	20	112	4
pseudo/mps/normalized-mps-sentoy	60	1	2	60
pseudo/mpsReduced/normalized-mps-red-blend2	2 944	196	2	2 659
pseudo/mpsReduced/normalized-mps-red-est	146	4	2	74
pseudo/mpsReduced/normalized-mps-red-markshare1	230	6	2	80
pseudo/mpsReduced/normalized-mps-red-markshare1-1	280	11	2	130
pseudo/mpsReduced/normalized-mps-red-markshare2	270	7	2	90
pseudo/mpsReduced/normalized-mps-red-markshare2-1	330	13	2	150

TABLE 6.2 – Liste des benchmarks non-binaires qui ont été détectées comme étant DBTP.

Benchmark	n	e	d
domino/normalized-domino-100-100	100	100	100
domino/normalized-domino-100-200	100	100	200
domino/normalized-domino-100-300	100	100	300
domino/normalized-domino-300-100	300	300	100
domino/normalized-domino-300-200	300	300	200
domino/normalized-domino-300-300	300	300	300
domino/normalized-domino-500-100	500	500	100
domino/normalized-domino-500-200	500	500	200
domino/normalized-domino-500-300	500	500	300
domino/normalized-domino-500-500	500	500	500
domino/normalized-domino-800-100	800	800	100
domino/normalized-domino-800-200	800	800	200
domino/normalized-domino-800-300	800	800	300
domino/normalized-domino-800-500	800	800	500
domino/normalized-domino-800-800	800	800	800
domino/normalized-domino-1000-100	1 000	1 000	100
domino/normalized-domino-1000-200	1 000	1 000	200
domino/normalized-domino-1000-300	1 000	1 000	300
domino/normalized-domino-1000-500	1 000	1 000	500
domino/normalized-domino-1000-800	1 000	1 000	800
domino/normalized-domino-1000-1000	1 000	1 000	1 000
domino/normalized-domino-2000-2000	2 000	2 000	2 000
domino/normalized-domino-3000-3000	3 000	3 000	3 000
domino/normalized-domino-5000-5000	5 000	5 000	5 000
marc/normalized-large-80-sat_ext	80	3 160	80
marc/normalized-large-84-sat_ext	84	3 486	84
marc/normalized-large-88-sat_ext	88	3 828	88
marc/normalized-large-92-sat_ext	92	4 186	92
marc/normalized-large-96-sat_ext	96	4 560	96

TABLE 6.3 – Liste des benchmarks binaires qui ont été détectés comme étant $DBTP^{AC}$.

un hypergraphe de contraintes β-acyclique. Par conséquent, le reste des benchmarks de cette famille satisfait $DBTP^{AC}$ grâce au contenu de leurs relations. Donc, la famille `primes-*` illustre parfaitement le fait que $DBTP$ est une classe hybride.

La table 6.5 présente la liste des familles dans lesquelles aucun benchmark ne satisfait DBTP, il s'agit de 1640 benchmarks. Les autres benchmarks qui ne sont pas DBTP appartiennent à des familles dans lesquelles il existe au moins un benchmark qui satisfait DBTP ou bien qui a un état inconnu (pour dépassement du temps prévu pour le test).

Pour conclure, les benchmarks qui sont $DBTP$ (ou $DBTP^{AC}$) présentent 4% des benchmarks considérés et 18% si nous considérons les benchmarks arc-incohérents et qui sont trivialement $DBTP^{AC}$. Ceci prouve que cette classe polynomiale est exploitable en pratique et que les benchmarks DBTP peuvent expliquer leur efficacité de résolution (voir la sous-section suivante).

6.6.2 Liens avec la résolution

Généralement, les CSP appartenant à une classe polynomiale sont résolues en temps polynomial par un algorithme dédié à cette classe. Ici, notre but n'est pas de présenter un algorithme de résolution spécifique à DBTP, mais d'exploiter la propriété DBTP pour expliquer pourquoi certains CSP sont résolus en temps polynomial par des algorithmes classiques comme MAC [Sabin and Freuder, 1994] ou RFL [Nadel, 1988]. Nous nous focalisons évidemment sur les CSP qui sont DBTP et arc-cohérents.

Pour les CSP binaires qui sont $DBTP^{AC}$ et arc-cohérents, le théorème 6.4 énonce que MAC résout ces benchmarks en temps polynomial. En pratique, MAC s'avère très efficace vu qu'il résout tous les benchmarks de la table 6.1 sans faire de retour-arrière.

Pour les CSP non-binaires, les 54 benchmarks satisfaisant DBTP de la famille `primes-*` sont aussi résolues par MAC sans backtrack (à l'exception de deux benchmarks qui exigent un seul backtrack). Pour les 10 benchmarks, la cardinalité maximale des intersections entre les portées des contraintes ne dépasse pas 1. Donc, le théorème 6.5 est appliqué, ce qui explique la résolution efficace de ces benchmarks par MAC. Par contre, ce théorème concerne seulement le cas où la cohérence d'arc du CSP engendre son inter-cohérence. D'autres cas pourraient exister et dépendraient des caractéristiques des relations. Pour 38 benchmarks, l'application de la cohérence d'arc implique l'inter-cohérence à chaque niveau de recherche et donc MAC a pu les résoudre en temps polynomial comme énoncé dans le théorème 6.4. Pour 6 benchmarks, l'application de la cohérence d'arc n'entraine pas l'inter-cohérence du CSP. Donc, l'appartenance à DBTP ne suffit pas comme raison pour expliquer sa résolution efficace. Le benchmark `pseudo/mps/normalized-mps-sentoy` et les trois premiers benchmarks de la famille `mknap` sont résolus efficacement sans backtrack. Comme elles ont une seule contrainte, alors les conditions du théorème 6.4 sont vérifiées, ce qui explique l'efficacité de la résolution. Par contre, pour les autres benchmarks de la famille `mknap` ou les benchmarks de la famille `pseudo/mpsReduced`, MAC ne réussit pas à les résoudre efficacement bien qu'ils soient $DBTP^{AC}$. Ceci est dû au fait que les relations sont données en intension (et donc ne respecte pas notre hypothèse de travail) et au fait que leurs contraintes ont une arité importante. Si la violation de cette hypothèse n'a souvent pas de conséquence sur l'efficacité de MAC vis-à-vis des benchmarks $DBTP$ (tous les benchmarks de la table 6.2 et la table 6.4 ont des contraintes définies en intension), ce n'est pas le cas ici.

Finalement, nous notons que nous pouvons avoir les mêmes résultats en utilisant RFL au lieu de MAC et que les théorèmes 6.4 et 6.5 restent aussi vrais pour RFL.

Benchmark	n	e	d	r
primes-10/normalized-primes-10-40-2-1	100	40	28	3
primes-10/normalized-primes-10-40-3-1	100	40	28	4
primes-10/normalized-primes-10-60-2-1	100	60	28	3
primes-10/normalized-primes-10-60-2-3	100	60	28	5
primes-10/normalized-primes-10-60-2-5	100	60	28	7
primes-10/normalized-primes-10-60-3-1	100	60	28	4
primes-10/normalized-primes-10-80-2-1	100	80	28	3
primes-10/normalized-primes-10-80-2-3	100	80	28	5
primes-10/normalized-primes-10-80-2-5	100	80	28	7
primes-10/normalized-primes-10-80-3-1	100	80	28	4
primes-10/normalized-primes-10-80-3-3	100	80	28	6
primes-15/normalized-primes-15-40-2-1	100	40	46	3
primes-15/normalized-primes-15-40-2-3	100	40	46	5
primes-15/normalized-primes-15-40-3-1	100	40	46	4
primes-15/normalized-primes-15-60-2-1	100	60	46	3
primes-15/normalized-primes-15-60-2-3	100	60	46	5
primes-15/normalized-primes-15-60-3-1	100	60	46	4
primes-15/normalized-primes-15-60-3-3	100	60	46	6
primes-15/normalized-primes-15-80-2-1	100	80	46	3
primes-15/normalized-primes-15-80-2-3	100	80	46	5
primes-15/normalized-primes-15-80-3-1	100	80	46	4
primes-15/normalized-primes-15-80-3-3	100	80	46	6
primes-20/normalized-primes-20-40-2-1	100	40	70	3
primes-20/normalized-primes-20-40-3-1	100	40	70	4
primes-20/normalized-primes-20-60-2-1	100	60	70	3
primes-20/normalized-primes-20-60-2-3	100	60	70	5
primes-20/normalized-primes-20-60-3-1	100	60	70	4
primes-20/normalized-primes-20-80-2-1	100	80	70	3
primes-20/normalized-primes-20-80-2-3	100	80	70	5
primes-20/normalized-primes-20-80-3-1	100	80	70	4
primes-25/normalized-primes-25-40-2-1	100	40	96	3
primes-25/normalized-primes-25-40-3-1	100	40	96	4
primes-25/normalized-primes-25-60-2-1	100	60	96	3
primes-25/normalized-primes-25-60-2-3	100	60	96	5
primes-25/normalized-primes-25-60-3-1	100	60	96	4
primes-25/normalized-primes-25-80-2-1	100	80	96	3
primes-25/normalized-primes-25-80-2-3	100	80	96	5
primes-25/normalized-primes-25-80-3-1	100	80	96	4
primes-30/normalized-primes-30-40-2-1	100	40	112	3
primes-30/normalized-primes-30-40-3-1	100	40	112	4
primes-30/normalized-primes-30-60-2-1	100	60	112	3
primes-30/normalized-primes-30-60-3-1	100	60	112	4
primes-30/normalized-primes-30-80-2-1	100	80	112	3
primes-30/normalized-primes-30-80-3-1	100	80	112	4

TABLE 6.4 – Liste des benchmarks non-binaires qui ont été détectées comme étant $DBTP^{AC}$.

binaire		non-binaire
BH-4-4	frb50-23	aim-100
bqwh-15-106	frb56-25	aim-200
bqwh-18-141	frb59-26	aim-50
coloring	geom	dubois
composed-25-1-2	graphColoring/mug	pret
composed-25-1-25	graphColoring/sgb/book	pseudo/aim
composed-25-1-40	graphColoring/sgb/games	
composed-25-1-80	QCP-10	
composed-25-10-20	QCP-15	
composed-75-1-2	QCP-20	
composed-75-1-25	QWH-10	
composed-75-1-40	QWH-15	
composed-75-1-80	QWH-20	
ehi-85	rand-2-30-15-fcd	
ehi-90	rand-2-40-19	
fapp/fapp17	rand-2-40-19-fcd	
fapp/fapp18	Rand-2-50-23	
fapp/fapp19	rand-2-50-23-fcd	
fapp/fapp20	tightness0.1	
frb30-15	tightness0.2	
frb35-17	tightness0.35	
frb40-19	tightness0.5	
frb45-21		

TABLE 6.5 – Liste des familles de benchmarks binaires et non-binaires pour lesquelles aucun benchmark n'est DBTP.

6.7 Conclusion

Dans ce chapitre, nous avons étendu une classe polynomiale hybride (BTP) dont les CSP y appartenant peuvent être résolues en temps polynomial par des algorithmes de type MAC en utilisant les différentes microstructures présentées dans le chapitre précédent. Nous avons montré que DBTP paraît être la meilleure extension de BTP en terme de simplicité et de résultats similaires à BTP. Nous avons prouvé que DBTP est incomparable avec plusieurs classes polynomiales connues (notamment BTP) et qu'elle inclut des classes polynomiales à la fois structurelles et relationnelles (à savoir les CSP β-acycliques et les CSP Triangulaires). Ensuite, nous avons mis en évidence les liens existants entre (D)BTP et la k-cohérence ainsi que l'hyper-k-cohérence directionnelle. D'un point de vue pratique, nous avons montré comment DBTP peut être utilisé pour expliquer l'efficacité des solveurs vis-à-vis de ces CSP.

Une première extension devrait consister en l'étude des liens entre DBTP et d'autres classes polynomiales non prises en compte dans cette contribution. Puis, dans le même esprit, nous essayerons de l'étendre au problème SAT et de tester si nous aurions des résultats aussi intéressants. Nous pourrions aussi explorer la possibilité de définir d'autres extensions de BTP en nous basant sur les autres microstructures de CSP non-binaires présentées dans le chapitre 3. Pour cela, il faudra étudier leurs propriétés d'une manière profonde, comme ce que nous avons fait pour BTP, et voir si des liens entre ces extensions pourraient exister. Quelques résultats préliminaires sur ces extensions ont été envisagés dans [El Mouelhi et al., 2013b].

Conclusion

Dans cette thèse, nous nous sommes intéressés à l'étude des classes polynomiales pour le problème de satisfaction des contraintes d'arité quelconque. Nous avons principalement comme objectif de combler le fossé qui existe entre une étude théorique assez riche et des résultats pratiques efficaces des solveurs. En effet, nous avons dans un premier temps proposé des classes polynomiales qui peuvent être exploitées efficacement par les algorithmes de résolution classiques à base de backtrack. Dans un second temps, nous avons réalisé une étude expérimentale, dans le cas binaire, sur la propriété BTP et dans le cas non-binaire sur son extension DBTP, ce qui a permis de déduire que BTP est une propriété intéressante pour la résolution et la simplification des problèmes réels.

Nos contributions peuvent être partagées en deux parties (comme elles étaient présentées dans ce manuscrit). Une première partie (partie 2) qui porte sur l'extension de la notion de microstructure aux CSP d'arité quelconque et leur exploitation pour l'étude des classes polynomiales. Cette partie est composée de deux chapitres (chapitre 3 et chapitre 4). Dans le chapitre 3, nous avons proposé des extensions de la notion de microstructure aux CSP d'arité quelconque. Les différentes microstructures que nous avons introduites sont des graphes et sont basées sur les codages binaires des CSP non-binaires à savoir le codage dual, le codage par variables cachées et le codage double. Le choix des graphes plutôt que des hypergraphes (comme le suggérait Cohen dans Cohen [2003]) peut se justifier par plusieurs arguments dont la simplicité et surtout l'existence d'une littérature relativement riche comparativement à celle existant pour les hypergraphes. Nous avons montré comment ces microstructures peuvent être utilisées pour étendre certaines classes polynomiales comme Zéro/Un/Tous pour obtenir des résultats assez similaires au cas binaire. La poursuite de ce travail nécessite d'étudier les extensions de certaines autres classes polynomiales, rappelées dans l'état de l'art, en utilisant plus particulièrement ces microstructures. Cela pourrait permettre de confirmer qu'une de ces microstructures peut être la plus adéquate pour l'étude des classes polynomiales en particulier et le problème CSP en général. Sinon, il serait intéressant de trouver une vraie généralisation de la notion de microstructure qui sera basée sur les graphes.

Dans le chapitre 4, nous avons présenté une nouvelle évaluation de la complexité des algorithmes Backtrack, Forward Checking et Real Full Lookahead. Cette nouvelle analyse consiste à exprimer la complexité de ces algorithmes en fonction du nombre de cliques maximales contenues dans la microstructure. Nous avons, tout d'abord, montré que les CSP binaires ayant un nombre polynomial de cliques maximales dans leur microstructure peuvent être résolus en temps polynomial par les algorithmes précédents. Ensuite, nous avons étendu les résultats obtenus pour le cas binaire aux CSP non-binaires en utilisant une microstructure introduite dans le chapitre 3. En théorie de graphes, il existe certains types de graphes ayant un nombre polynomial de cliques maximales comme les graphes planaires, les graphes triangulés, les graphes plongeables dans une surface et les graphes CSG^k qui sont une généralisation des graphes triangulés. Les travaux futurs devraient étudier d'autres paramètres graphiques (ou non graphiques) qui permettraient de mieux analyser la complexité des algorithmes de résolution. Il serait également nécessaire de valider cette

étude par une partie expérimentale complète. Aussi, il serait intéressant de proposer de nouvelles classes de graphes qui respecteraient les conditions énoncées dans ce chapitre et qui permettraient de mieux capturer les problèmes du monde réel. En ce qui concerne MAC dont le comportement est différent de celui des algorithmes BT, FC et RFL, il faudrait vérifier si un résultat similaire à celui de ces algorithmes pourrait être prouvé.

La seconde partie de nos contributions (partie 3 de la thèse) porte sur l'exploitation pratique de la propriété BTP. Cette partie se décline en deux chapitres (chapitre 5 et chapitre 6). Dans le chapitre 5, nous avons tout d'abord introduit le concept de classe polynomiale cachée. Ce concept est basé sur la transformation d'un CSP par l'ajout ou la suppression des valeurs, variables ou contraintes. Des techniques très utilisées en pratique permettent de réaliser ce genre de transformation, par exemple les filtrage par cohérence locale. Nous avons mis l'accent dans cette étude sur la classe polynomiale BTP et nous avons montré que plusieurs benchmarks de la compétition CSP 2008 sont captés par cette classe après application d'un filtrage comme AC, PC, SAC, NIC, PIC, RPC, SPC... Ces classes polynomiales sont donc appelées classes cachées puisque le CSP de départ n'y appartient pas. Ensuite, nous avons étudié la propriété BTP mais cette fois pour fusionner certaines valeurs dans les domaines des variables d'un CSP binaire. En fait, nous avons prouvé que tout couple de valeurs d'une même variable n'appartenant à aucun triangle cassé peut être fusionné en préservant la satisfiabilité du CSP. Nous avons testé cette nouvelle méthode de fusion sur les benchmarks et nous avons obtenu des résultats assez intéressants en termes de pourcentage de valeurs supprimées. Une première perspective consiste à appliquer l'étude sur les classes cachées sur d'autres classes polynomiales en utilisant d'autres moyens de transformation ou en rendant la définition de transformation moins restrictive. Une deuxième perspective est constituée par l'étude d'une généralisation de la fusion par BTP aux CSP d'arité quelconque voire même au problème SAT. Aussi, il faudrait voir si un résultat théorique sur l'ordre des variables pourrait être déduit afin de confirmer l'unicité du résultat final obtenu par la fusion par BTP. Une troisième perspective pourrait soit étudier l'influence de fixer un ordre (sur les variables) sur le nombre total de valeurs supprimées, soit proposer une version plus allégée (voire même différente) de la définition actuelle qui permettrait de supprimer plus de valeurs en temps plus optimal.

Le chapitre 6 est consacré à l'extension de la propriété BTP aux CSP d'arité quelconque. Notre extension, appelée DBTP, permet d'obtenir des résultats similaires aux résultats obtenus pour le cas binaire. Par exemple, BTP permet de capturer les CSP binaires arborescents et notre extension permet de capturer tout CSP β-acyclique d'arité quelconque. Comme pour BTP, les CSP appartenant à cette nouvelle classe peuvent être résolus efficacement par des algorithmes de type MAC et RFL. Nous avons également établi le lien entre (D)BTP et la cohérence globale. Après, nous avons montré que DBTP est différente de BTP et des autres classes polynomiales connues dans l'état de l'art. Enfin, nous avons mis en évidence l'existence de cette propriété en pratique puisque certains benchmarks satisfont ses conditions. Les benchmarks appartenant à cette classe peuvent être résolus en temps polynomial par une version généralisée de MAC ou de RFL et pour certains sans retour arrière. Une première extension de ce travail peut être réalisée en envisageant une autre extension de BTP aux CSP d'arité quelconque en utilisant d'autres microstructures n-aires. Il serait également intéressant d'étudier une forme allégée de BTP, qui conserverait l'essentiel de ses propriétés (contrairement à min-of-max extendable), et qui permettrait ainsi de capturer plus de benchmarks et de justifier, encore plus, l'efficacité des solveurs. La poursuite de ce travail nécessite d'identifier les conditions pour lesquelles des algorithmes non étudiés dans ce chapitre (comme BT, FC et autres) seront polynomiaux sur des CSP satisfaisant BTP ou DBTP.

Enfin, dans cette thèse nous avons introduit des nouveaux concepts qui ont été utiles d'une part pour étudier des classes polynomiales pouvant être exploitées en pratique par les algorithmes de l'état de l'art, et d'autre part pour expliquer l'efficacité des solveurs vis-à-vis de certains problèmes réels. Les travaux que nous avons présenté permet d'amoindrir le fossé entre théorie et pratique

mais d'autres pistes devraient être explorées pour continuer ce que nous avons commencé.

Bibliographie

CSPLib : A problem library for constraints. http://www.csplib.org.

F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *AAAI/IAAI*, pages 310–318, 1998.

R. Barták and R. Erben. A new algorithm for singleton arc consistency. In *FLAIRS Conference*, pages 257–262, 2004.

C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30 :479–513, 1983.

C. Berge. *Graphes*. Gauthier-Villars, 1987a.

C. Berge. *Hypergraphes – Combinatoire des ensembles finis*. Gauthier-Villars, 1987b.

P. Berlandier. A symbiotic approach to arc and path consistency checking. In *EPIA*, pages 107–114, 1995.

P. Bernstein and N. Goodman. The power of natural semijoins. *SIAM J. Comput.*, 10-4 :751–771, 1981.

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

C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65 :179–190, 1994.

C. Bessière. *Constraint Propagation*. Elsevier, 2006.

C. Bessière and R. Debruyne. Proceedings ecai'04 workshop on modelling and solving problems with constraints. pages 20–29, 2004.

C. Bessière and R. Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In *IJCAI*, pages 54–59, 2005.

C. Bessière and J.-C. Régin. MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ ?) on Hard Problems. In *Proceedings of CP'96*, pages 61–75, 1996.

C. Bessière, E. C. Freuder, and J.-C. Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artif. Intell.*, 107(1) :125–148, 1999.

C. Bessière, J.-C. Régin, R. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2) :165–185, 2005.

C. Bessiere, S. Cardon, R. Debruyne, and C. Lecoutre. Efficient algorithms for singleton arc consistency. *Constraints*, 16(1) :25–53, 2011.

F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, pages 146–150, 2004.

N. Chandrasekharan and S. S. Iyengar. Nc algorithms for recognizing chordal graphs and *k* trees. *IEEE Trans. Computers*, 37(10) :1178–1183, 1988.

K. C. K. Cheng and R. H. C. Yap. Maintaining generalized arc consistency on ad-hoc n-ary boolean constraints. In *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, pages 78–82, 2006.

A. Chmeiss. Sur la consistance de chemin et ses formes partielles. In *Actes du Congrès AFCET-RFIA 96*, pages 212–219, Rennes, France, 1996.

A. Chmeiss and P. Jégou. A generalization of chordal graphs and the maximum clique problem. *Information Processing Letters*, 62 :111–120, 1997.

A. Chmeiss and P. Jégou. Décomposition : Vers une érosion du pic de difficulté ? In $4^{ème}$ *Journées Nationales Résolution Pratique des Problèmes NP-Complets*, pages 21–29. JNPC 98, 1998.

A. Chmeiss and L. Sais. About the use of local consistency in solving csps. In *ICTAI*, pages 104–107, 2000.

M. Chudnovsky, G. Cornuéjols, X. Liu, P. D. Seymour, and K. Vuskovic. Recognizing berge graphs. *Combinatorica*, 25(2) :143–186, 2005.

M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *ANNALS OF MATHEMATICS*, 164 :51–229, 2006.

D. A. Cohen. A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *CP*, pages 807–811, 2003.

D. A. Cohen, M. C. Cooper, M. J. Green, and D. Marx. On guaranteeing polynomially bounded search tree size. In *CP*, pages 160–171, 2011.

D. A. Cohen, M. C. Cooper, G. Escamocher, and S. Zivny. Variable elimination in binary csp via forbidden patterns. In *IJCAI*, 2013.

M. Cooper. An Optimal k-Consistency Algorithm. *Artificial Intelligence*, 41, 1989.

M. Cooper, D. Cohen, and P. Jeavons. Characterising Tractable Constraints. *Artificial Intelligence*, 65(2) :347–361, 1994.

M. Cooper, P. Jeavons, and A. Salamon. Generalizing constraint satisfaction on trees : hybrid tractability and variable elimination. *Artificial Intelligence*, 174 :570–584, 2010.

M. C. Cooper. Fundamental properties of neighbourhood substitution in constraint satisfaction problems. *Artif. Intell.*, 90(1-2) :1–24, 1997.

M. C. Cooper, A. El Mouelhi, C. Terrioux, and B. Zanuttini. On broken triangles. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 9–24, 2014.

D. G. Corneil. Lexicographic breadth first search - a survey. In *WG*, pages 1–19, 2004.

A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126 :5–41, 2001.

A. Darwiche and M. Hopkins. Using recursive decomposition to construct elimination orders, jointrees and dtrees. In *Trends in Artificial Intelligence, Lecture notes in AI, 2143*, pages 180–191. Springer-Verlag, 2001.

A. Darwiche and K. Pipatsrisawat. Complete algorithms. In *Handbook of Satisfiability*, pages 99–130. 2009.

P. David. When functional and bijective constraints make a CSP polynomial. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 224–231, 1993.

R. Debruyne. A property of path inverse consistency leading to an optimal PIC algorithm. In *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, pages 88–92, 2000.

R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problems. In *Proceedings of IJCAI 97*, pages 412–417. IJCAI 97, 1997.

R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *CP*, pages 312–326, 1997.

R. Debruyne and C. Bessière. Domain Filtering Consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

R. Dechter. Enhancement Schemes for Constraint Processing : Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41 :273–312, 1990.

R. Dechter. Constraint Networks. In *Encyclopedia of Artificial Intelligence*, volume 1, pages 276–285. John Wiley & Sons, Inc., second edition, 1992.

R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks and their AND/OR search space. In *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, Banff, Canada, July 7-11, 2004*, pages 120–129, 2004.

R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. In *Proceedings of the tenth International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 271–277, Detroit, Michigan, August 1989.

R. Dechter and J. Pearl. The Cycle-cutset method for Improving Search Performance in AI Applications. In *Proceedings of the third IEEE on Artificial Intelligence Applications*, pages 224–230, 1987a.

R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34 :1–38, 1987b.

R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38 :353–366, 1989.

J. Doyle. A truth maintenance system. *Artif. Intell.*, 12(3) :231–272, 1979.

V. Dujmovic, G. Fijavz, G. Joret, T. Sulanke, and D. R. Wood. On the maximum number of cliques in a graph embedded in a surface. *European J. Combinatorics*, 32(8) :1244–1252, 2011.

D. Duris. Some characterizations of γ and β-acyclicity of hypergraphs. *Information Processing Letters*, 112 (16) :617–620, 2012.

A. El Mouelhi, P. Jégou, C. Terrioux, and B. Zanuttini. On the efficiency of backtracking algorithms for binary constraint satisfaction problems. In *ISAIM*, January 2012.

A. El Mouelhi, P. Jégou, and C. Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. In *ICTAI*, pages 947–954, 2013a.

A. El Mouelhi, P. Jégou, and C. Terrioux. Microstructures for csps with constraints of arbitrary arity. In *SARA*, 2013b.

A. El Mouelhi, P. Jégou, C. Terrioux, and B. Zanuttini. Some new tractable classes of csps and their relations with backtracking algorithms. In *CPAIOR*, pages 61–76, 2013c.

A. El Mouelhi, P. Jégou, and C. Terrioux. Hidden Tractable Classes. In *ICTAI*, 2014.

A. El Mouelhi, P. Jégou, and C. Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. 2015.

R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3) : 514–550, 1983.

H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction : A framework for decision problems under incomplete knowledge. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1.*, pages 175–180, 1996.

E. C. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1) :24–32, 1982.

E. C. Freuder. A Sufficient Condition for Backtrack-Bounded Search. *JACM*, 32 :755–761, 1985.

E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *AAAI*, pages 227–233, 1991.

E. C. Freuder and C. D. Elfe. Neighborhood inverse consistency preprocessing. In *AAAI/IAAI, Vol. 1*, pages 202–208, 1996.

D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 572–578, Montréal, Canada, 1995.

D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3) :835–855, 1965.

F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1 (2) :180–187, 1972.

P. A. Geelen. Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In *ECAI*, pages 31–35, 1992.

M. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1 :25–46, 1993.

S. W. Golomb and L. D. Baumert. Backtrack programming. *J. ACM*, 12(4) :516–524, 1965.

G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. In *IJCAI 99*, pages 394–399, 1999.

G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

M. H. Graham. On the universal relation. Technical report, University of Toronto, 1979.

C. Han and C. Lee. Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence*, 36 :125–130, 1988.

R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313, 1980.

P. V. Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artif. Intell.*, 57(2-3) :291–321, 1992.

P. Janssen, P. Jégou, B. Nouguier, and M. Vilarem. A filtering process for general constraint satisfaction problems : achieving pairwise-consistency using an associated binary representation. In *Proceedings of IEEE Workshop on Tools for Artificial Intelligence*, pages 420–427, 1989.

P. Jeavons and M. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2) : 327–339, 1995.

P. Jégou. Cyclic-Clustering : a compromise between Tree-Clustering and the Cycle-Cutset method for improving search efficiency. In *Proceedings of European Conference on Artificial Intelligence (ECAI-90)*, pages 369–371, 1990.

P. Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution – Propagation de contraintes dans les réseau dynamiques*. PhD thesis, Université des Sciences et Techniques du Languedoc, January 1991.

P. Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *AAAI*, pages 731–736, 1993a.

P. Jégou. On the Consistency of General Constraint-Satisfaction Problems. In *AAAI*, pages 114–119, 1993b.

P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proc. of ECAI*, pages 196–200, 2004.

P. Jégou and C. Terrioux. Combining restarts, nogoods and decompositions for solving csps. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 465–470, 2014.

N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP'2000)*, pages 249–261, 2000.

S. Karakashian, R. J. Woodward, B. Y. Choueiry, S. Prestwhich, and E. C. Freuder. A partial taxonomy of substitutability and interchangeability. *CoRR*, abs/1010.4609, 2010.

H. A. Kautz, A. Sabharwal, and B. Selman. Incomplete algorithms. In *Handbook of Satisfiability*, pages 185–203. 2009.

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 47 :498–519, 1998.

C. Lecoutre. *Constraint Networks - Techniques and Algorithms*. ISTE/Wiley, 2009.

C. Lecoutre and F. Hemery. Une étude des supports résiduels pour laconsistance d'arc. In *Actes de la 3ème Journées Francophones de la Programmation par Contraintes*, pages 267–276. JFPC'06, 2006.

C. Lecoutre and P. Prosser. Maintaining Singleton Arc Consistency. In *3rd International Workshop on Constraint Propagation And Implementation held with CP'06*, pages 47–61. CPAI'06, 2006.

C. Lecoutre and J. Vion. Enforcing Arc Consistency using Bitwise Operations. In *Constraint Programming Letters*, pages 21–35. CPL, 2008.

C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In *CP*, pages 480–494, 2003.

C. Lecoutre, S. Cardon, and J. Vion. Path consistency by dual consistency. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, pages 438–452, 2007a.

C. Lecoutre, S. Cardon, and J. Vion. Conservative dual consistency. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 237–242, 2007b.

C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood recording from restarts. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 131–136, 2007c.

C. Lecoutre, S. Cardon, and J. Vion. Second-order consistencies. *J. Artif. Intell. Res. (JAIR)*, 40 : 175–219, 2011.

J. Lehel. A characterization of totally balanced hypergraphs. *Discrete Mathematics*, 57(1-2) :59–65, 1985.

C. Likitvivatanavong and R. H. C. Yap. Many-to-many interchangeable sets of values in csps. In *SAC*, pages 86–91, 2013.

A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8 :99–118, 1977.

A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artif. Intell.*, 25(1) :65–74, 1985.

J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. pages 229–250, 1979.

R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28, 1986.

R. Mohr and G. Masini. Good old discrete relaxation. In *ECAI*, pages 651–656, 1988.

U. Montanari. Networks of Constraints : Fundamental Properties and Applications to Picture Processing. *Artificial Intelligence*, 7 :95–132, 1974.

J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3 :23–28, 1965.

B. Nadel. *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*, pages 287–342. In *Search in Artificial Intelligence*. Springer-Verlag, 1988.

S. N. Ndiaye. 2-triangulation de micro-structure pour la résolution de CSP. In *3ème Manifestation des Jeunes Chercheurs en Sciences et Technologies de l'Information et de la Communication (Majecstic'2005)*, pages 180–187, Nov. 2005.

A. Paparrizou and K. Stergiou. Extending generalized arc consistency. In *Artificial Intelligence : Theories and Applications - 7th Hellenic Conference on AI, SETN 2012, Lamia, Greece, May 28-31, 2012. Proceedings*, pages 174–181, 2012.

C. Peirce, C. Hartshorne, and P. Weiss. *Collected Papers of Charles Sanders Peirce*. Number vol. 3. Harvard University Press, 1933.

P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9 :268–299, 1993.

P. Prosser. Mac-cbj maintaining arc consistency with conflict direct backjumping. In *Technical Report Research Report*, 1995.

A. Rauzy. Polynomial restrictions of SAT : What can be done with an efficient implementation of the Davis and Putnam's procedure. In U. Montanari and F. Rossi, editors, *Proc. International Conference on Principles of Constraint Programming (CP 1995)*, pages 515—532. Springer Verlag, 1995.

P. Refalo. Impact-based search strategies for constraint programming. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, pages 557–571, 2004.

J.-C. Régin. A filtering algorithm for constraints of difference in csps. In *AAAI*, pages 362–367, 1994.

N. Robertson and P. Seymour. Graph minors II : Algorithmic aspects of treewidth. *Algorithms*, 7 : 309–322, 1986.

B. Rosgen and L. Stewart. Complexity results on graphs with few cliques. *Discrete Mathematics and Theoretical Computer Science*, 9 :127–136, 2007.

F. Rossi, C. J. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *ECAI*, pages 550–556, 1990.

F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI*, pages 125–129, 1994.

A. Salamon and P. Jeavons. Perfect Constraints Are Tractable. In *Proceedings of CP*, pages 524–528, 2008.

T. Schiex and G. Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In *Proceedings of the 5^{th} IEEE International Conference on Tools with Artificial Intelligence*, Nov. 1993.

M. Singh. Path Consistency Revisited. In *Proceedings of the 5th International Conference on Tools for Artificial Intelligence*, pages 318–325, 1995.

B. M. Smith. The brélaz heuristic and optimal static orderings. In *CP*, pages 405–418, 1999.

B. M. Smith, K. Stergiou, and T. Walsh. Using auxiliary variables and implied constraints to model non-binary problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 182–187, 2000.

R. Stallman and G. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9 :135–196, 1977.

K. Stergiou and T. Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In *AAAI*, pages 163–168, 1999.

E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

P. van Beek and R. Dechter. On the minimality and decomposability of row-convex constraint networks. *J. ACM*, 42(3) :543–561, 1995.

G. Verfaillie and T. Schiex. Dynamic backtracking for dynamic csps. In *Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, 1994.

R. Wallace. Directed arc consistency preprocessing. In *Proceedings of the ECAI-94 Workshop on Constraint Processing*, volume LNCS 923, pages 121–137, 1994.

R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.

D. R. Wood. On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23 : 337–352, June 2007.

Y. Zhang and F. S. Bao. A survey of tree convex sets test. *CoRR*, 2009.

Y. Zhang and E. C. Freuder. Properties of tree convex constraints. *Artif. Intell.*, 172(12-13) :1605–1612, 2008.

Y. Zhang and R. H. C. Yap. Consistency and set intersection. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 263–270, 2003.

Y. Zhang, R. H. C. Yap, and J. Jaffar. Functional elimination and 0/1/all constraints. In *AAAI/IAAI*, pages 175–180, 1999.

Classes Polynomiales pour CSP : de la Théorie à la Pratique.

Résumé :

Les travaux de recherche de cette thèse portent sur l'analyse des classes polynomiales et leur applicabilité pour la résolution pratique de problèmes de satisfaction de contraintes (CSP) à domaines finis. Les travaux s'articulent en deux parties principales. Dans la première partie, nous proposons dans un premier temps plusieurs formes de microstructures pour les CSP d'arité quelconque. Il s'agit là d'outils théoriques qui doivent faciliter l'étude des classes polynomiales quand les contraintes ne sont pas uniquement binaires. Dans un second temps, nous proposons de nouvelles classes polynomiales de CSP dont la mise en évidence doit permettre d'une part de contribuer à l'explication, pour partie, de l'efficacité pratique des solveurs de l'état de l'art, mais aussi, d'offrir des perspectives d'intégration simple dans ces solveurs. Cela passe notamment par la définition de nouvelles classes polynomiales traitables directement par les solveurs, sans recours comme c'est le cas traditionnellement, à des algorithmes ad hoc. Ces nouvelles classes polynomiales sont liées notamment au nombre de cliques maximales présentes dans les microstructures de CSP binaires ou non binaires. Dans la dernière partie, nous nous intéressons à la présence d'instances appartenant à des classes polynomiales dans les benchmarks classiques utilisés par la communauté CP. Nous étudions en particulier la classe BTP et une extension de BTP que nous proposons pour les contraintes d'arité quelconque. Nous introduisons pour cela un cadre formel et nous développons le concept de classe polynomiale cachée que nous exploitons ensuite d'un point de vue expérimental.

Mots clés : CSP, classes polynomiales, résolution pratique.

Tractable Classes of CSP : from Theory to Practice.

Abstract :

The research of this thesis focuses on the analysis of polynomial classes and their practical exploitation for solving constraint satisfaction problem (CSP) with finite domains. Our work is organized into two main parts. In the first part, we propose several types of microstructures for CSPs with arbitrary arity. This is a theoretical tools designed to facilitate the study of polynomial classes when the constraints are non-binary. After that, we propose a new polynomial classes of CSPs whose the highlighting should allow on the one hand to explain the effectiveness of solvers of the state of the art and on the second hand to provide the opportunities for easy integration in these solvers. These would include the definition of new polynomial classes, tractable by solvers without using of an ad hoc algorithms as in the traditional case. These new tractable classes are related to the number of maximal cliques in the microstructure of binary or non-binary CSP. In the last part, we focus on the presence of instances belonging to polynomial classes in classical benchmarks used by the CP community. We study in particular the BTP property and its extension DBTP to CSP with arbitrary arity. We introduce a formal framework and we develop the concept of hidden tractable class that we exploit from an experimental point of view.

Keywords : CSP, tractable classes, practical solving.