Classes polynomiales du problème CSP : entre théorie et pratique

Cyril Terrioux

AIX-MARSEILLE UNIVERSITÉ

FACULTÉ DES SCIENCES

LABORATOIRE DES SCIENCES DE L'INFORMATION
ET DES SYSTÈMES - UMR CNRS 7296

Habilitation à Diriger des Recherches

Discipline : Informatique

Cyril TERRIOUX

Classes polynomiales du problème CSP : entre théorie et pratique

Soutenue le 14 décembre 2016 devant le jury composé de :

M. Christian BESSIÈRE	CNRS	Président
M. Philippe JÉGOU	Aix-Marseille Université	Tuteur
M. Christophe LECOUTRE	Université d'Artois	Rapporteur
M. Thomas SCHIEX	INRA	Rapporteur
Mme Christine SOLNON	INSA Lyon	Rapportrice
M. Ioan TODINCA	Université d'Orléans	Examinateur

Remerciements

L'écriture de cette habilitation est, pour moi, l'occasion d'exprimer toute ma gratitude auprès des personnes qui ont contribué, d'une manière ou d'une autre, à mon travail.

Pour commencer, je souhaite adresser tous mes remerciements à Christophe LECOUTRE, Thomas SCHIEX et Christine SOLNON d'avoir accepté de rapporter ce mémoire d'habilitation et à Christian BESSIÈRE et Ioan TODINCA d'avoir accepté de participer au jury.

Je remercie tout particulièrement Philippe JÉGOU, qui par sa sagesse et sa vision, a toujours su me guider dans les différentes missions qui incombent à un enseignant-chercheur.

Les travaux présentés ici sont essentiellement le fruit d'un travail collectif et d'échanges. Je profite donc de ces quelques lignes pour remercier tous les collègues (des doctorants aux chercheurs les plus confirmés) avec qui j'ai eu le plaisir de collaborer ou d'échanger des idées.

Je remercie bien sûr les membres de mon équipe de recherche autant d'un point de vue scientifique que pour l'atmosphère conviviale qu'il y règne. Un grand merci aussi à tous les personnels administratifs et techniques (et en particulier ceux du LSIS) qui nous facilitent la vie au quotidien et sans qui tout serait plus difficile.

Enfin, je tiens bien sûr à remercier toute ma famille notamment mes parents, mes grands-parents, mon épouse et nos deux enfants, pour leur soutien et leur patience.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Liste des symboles

Classes polynomiales

α-$ACYCLIC$	Ensemble des instances ayant un hypergraphe de contraintes α-acyclique 33
β-$ACYCLIC$	Ensemble des instances ayant un hypergraphe de contraintes β-acyclique 33
$BBTW_b$	Ensemble des instances binaires ayant une BT-largeur au plus égale à b et satisfaisant la $\max(2, b+1)$-cohérence forte et m-wBTP pour une certaine valeur de m 46
$BHTW_k$	Ensemble des instances dont l'hypergraphe de contraintes a une largeur hyperarborescente bornée par la constante k 34
BTP	Ensemble des instances binaires satisfaisant la propriété BTP 35
BTW_k	Ensemble des instances dont l'hypergraphe de contraintes a une largeur arborescente bornée par la constante k 34
BV	Ensemble des instances binaires bivalentes 37
CCM	Ensemble des instances binaires dont le complémentaire de la microstructure est triangulée .. 35
CM	Ensemble des instances binaires ayant une microstructure triangulée 35
CSG^k	Ensemble des instances binaires ayant une microstructure CSG^k 41
$DBTP$	Ensemble des instances satisfaisant la propriété DBTP 47
DR-k	Ensemble des instances binaires ayant un rang directionnel au plus égal à k et satisfaisant la $(k+1)$-cohérence forte 36
ETP-SPC	Ensemble des instances binaires satisfaisant la chemin-cohérence forte et la propriété ETP ... 45
$IFUN$	Ensemble des instances incrémentalement fonctionnelles 37
k-BTP-sC	Ensemble des instances binaires satisfaisant la k-cohérence forte et la propriété k-BTP 46
MC	Ensemble des instances max-closed 32
PM	Ensemble des instances binaires ayant une microstructure parfaite 35
RC	Ensemble des instances binaires convexes par rangée 32
RRM	Ensemble des instances binaires renommables monotones à droite 32
$TREE$	Ensemble des instances binaires ayant un graphe de contraintes acyclique 32
$WBTP$	Ensemble des instances binaires satisfaisant la propriété WBTP 37
ZUT	Ensemble des instances binaires 0-1-tous 31

Symboles

Δ	Ensemble de décisions 28

$\mathcal{A}$	Affectation	23
$\mu(P)$	Microstructure de l'instance binaire P	22
$\mu_{DR}(P)$	DR-microstructure de l'instance P	41
$\mu_{DSR}(P)$	DSR-microstructure de l'instance P	42
$\mu_{HT}(P)$	HT-microstructure de l'instance P	42
$\mu_{ME}(P)$	ME-microstructure de l'instance P	42
$\omega_{\#}(G)$	Nombre de cliques maximales dans le graphe G	40
Σ	Suite de décisions	30
a	Arité d'une instance CSP	22
C	Ensemble de contraintes d'une instance CSP	21
c_{ij}	Contrainte binaire liant x_i et x_j	22
d	Taille du plus grand domaine	24
d_x	Domaine associé à la variable x	21
e	Nombre de contraintes d'une instance CSP	21
E_i	Un cluster d'une décomposition arborescente	33
h	Largeur d'une décomposition hyperarborescente d'un (hyper)graphe	34
hw	Largeur hyperarborescente d'un (hyper)graphe	34
n	Nombre de variables d'une instance CSP	21
P	Une instance CSP	21
$Pos(\Sigma)$	Ensemble des décisions positives de Σ	30
r	Taille de la plus grande relation	25
$R(c)$	Relation associée à la contrainte c	21
S	Taille d'une instance CSP	60
s	Taille de la plus grande intersection entre deux clusters d'une décomposition arborescente	57
$S(c)$	Portée de la contrainte c	21
w	Largeur arborescente d'un (hyper)graphe	33
w^+	Largeur d'une décomposition arborescente	57
X	Ensemble de variables d'une instance CSP	21

Introduction

Ce mémoire présente les différentes activités que j'ai pu mener tant au niveau de la recherche que de l'enseignement. Mes activités de recherche concernent principalement le problème de satisfaction de contraintes (CSP). Le problème CSP (Montanari, 1974) constitue une branche de l'Intelligence Artificielle qui possède des liens forts avec le problème SAT, la théorie des graphes et des hypergraphes ou encore la théorie des bases de données relationnelles. Dans le sillage du problème CSP, la programmation par contraintes a vu le jour il y a une quarantaine d'années et a connu depuis un essor important. Elle offre un cadre de travail puissant permettant de modéliser toute sorte de problèmes tout en mettant à disposition des outils efficaces pour les résoudre. Notamment, cette dernière décennie a vu l'apparition de nombreux solveurs complets, qui, bien qu'ayant une complexité temporelle exponentielle dans le pire des cas, ont fait montre d'une efficacité remarquable et ont ainsi rendu possible la résolution d'instances de taille importante et de nature relativement diverse. Cependant, en dépit de la qualité de ces résultats pratiques, la plupart des solveurs n'exploitent pas explicitement les classes polynomiales du problème CSP. De nombreuses classes polynomiales ont pourtant déjà été identifiées et de nouvelles le sont tous les ans. Toutefois, la grande majorité des travaux réalisés autour des classes polynomiales reste d'ordre théorique. Une des raisons à cela est liée aux algorithmes de résolution et de reconnaissance de ces classes polynomiales. Il s'avère que ces algorithmes sont bien trop souvent spécifiques à chaque classe polynomiale, rendant ainsi difficile, voire impossible, toute intégration dans les solveurs. Une autre raison concerne les restrictions imposées au niveau de la sémantique ou de la structure des instances pour garantir la polynomialité. Ces restrictions sont souvent considérées comme trop fortes pour permettre l'existence d'instances réalistes appartenant à des classes polynomiales. Les classes polynomiales, d'une part, et les solveurs, d'autre part, semblent donc constituer deux mondes bien différents. D'une certaine manière, mes travaux de recherche se situent à la frontière de ces deux mondes et visent à les rapprocher. En effet, dans ces travaux, au-delà des résultats théoriques, les classes polynomiales sont soit exploitées pour tenter d'expliquer la remarquable efficacité des solveurs, soit pour définir de nouveaux algorithmes de résolution.

Récemment, la classe polynomiale BTP (Cooper et al., 2008, 2010) a ouvert une nouvelle voie de recherche concernant les classes polynomiales hybrides. Un de ses principaux atouts réside dans la possibilité de résoudre ses instances avec des algorithmes de résolution classiques présents dans la plupart des solveurs sans pour autant avoir besoin de mettre en œuvre un quelconque traitement spécifique. Cette capture implicite des classes polynomiales peut alors, dans certains cas, permettre d'expliquer pourquoi les solveurs sont si efficaces sur certaines instances. Une partie de mes travaux est donc consacrée à proposer de nouvelles classes polynomiales pouvant, sous certaines conditions, être implicitement capturées par des solveurs.

Les classes polynomiales structurelles, quant à elles, semblent être les plus prometteuses quand il s'agit de vouloir proposer des algorithmes de résolution généraux. Je me suis particulièrement intéressé à l'une d'elles qui repose sur la notion de décomposition arborescente de graphes (Robertson et Seymour, 1986) introduite en théorie des graphes. Les travaux qui en découlent ont conduit à la définition de plusieurs algorithmes de résolution généraux et à leur étude tant du point de vue théorique que du point de vue pratique. Au niveau théorique, ces algorithmes obtiennent des bornes de complexité en temps parmi les meilleures existantes tandis qu'ils peuvent s'avérer relativement efficaces en pratique sur des instances possédant une structure adéquate. Certains de ces travaux ont ensuite été étendus à des domaines proches comme l'optimisation sous contraintes et le problème SAT.

La première partie de ce manuscrit présente une synthèse de mes travaux de recherche concernant l'exploitation des classes polynomiales d'un point de vue théorique et/ou pratique. Aussi, je ne décrirai pas, dans ce qui suit, les travaux relatifs à la résolution de CSP par des approches coopératives qui constitue la première partie de ma thèse (voir (Terrioux, 2001a,b, 2002) pour plus de détails). Cette première partie est divisée en trois chapitres. Le

premier consiste en un bref état de l'art introduisant un certain nombre de notions utiles pour la compréhension des deux chapitres suivants. Le second chapitre détaille les travaux accomplis concernant les classes polynomiales hybrides tandis que le troisième décrit les travaux réalisés autour des classes polynomiales structurelles. La seconde partie est consacrée à un curriculum vitae détaillé décrivant mes activités liées à la recherche (avec notamment la liste des encadrements réalisés et celle de mes publications) ou à l'enseignement ainsi que les différentes charges ou responsabilités assurées actuellement ou par le passé. La troisième partie présente une sélection de publications représentatives des travaux de recherche menés.

Première partie

Synthèse des travaux de recherche

Chapitre 1

Le problème de satisfaction de contraintes

Sommaire

Ce premier chapitre rappelle les principales notions qui seront utilisées dans les deux chapitres suivants. Il n'a donc pas vocation à être exhaustif. Le lecteur intéressé pourra se référer à (Apt, 2003; Dechter, 2003; Rossi et al., 2006; Lecoutre, 2009) pour des introductions plus complètes au cadre des problèmes de satisfaction de contraintes et de la programmation par contraintes. Tout d'abord, nous présentons le formalisme introduit par Montanari (Montanari, 1974) dans la section 1.1. Ensuite, nous nous intéressons à la résolution de ce problème. D'une part, dans la section 1.2, nous abordons les principales méthodes et techniques couramment employées pour résoudre ce problème dans le cas général. D'autre part, dans la section 1.3, nous décrivons quelques classes polynomiales de ce problème.

1.1 Le problème CSP

Un problème de satisfaction de contraintes (CSP) se définit par la donnée d'un ensemble de variables et d'un ensemble de contraintes. Chaque variable peut prendre une valeur choisie dans le domaine fini qui lui est associé tandis que les contraintes définissent les combinaisons de valeurs autorisées pour les variables. Plus formellement :

Définition 1.1 (Problème de satisfaction de contraintes (CSP) (Montanari, 1974)) *Une instance CSP est un triplet* (X, D, C)*, où* $X = \{x_1, \ldots, x_n\}$ *est un ensemble de* n *variables,* $D = \{d_{x_1}, \ldots, d_{x_n}\}$ *est un ensemble de domaines finis, à raison d'un domaine par variable, et* $C = \{c_1, \ldots, c_e\}$ *est un ensemble fini de* e *contraintes. Chaque contrainte* c_i *est un couple* $(S(c_i), R(c_i))$*, où* $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ *est la* portée *de* c_i*, et* $R(c_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ *est sa* relation de compatibilité.

Une des forces de ce formalisme réside dans les multiples possibilités existantes pour définir les relations associées aux contraintes. En effet, ces relations peuvent être données en extension, en intention ou à l'aide de motifs. En *extension*, elles sont définies en énumérant l'ensemble des tuples autorisés ou des tuples interdits.

En *intention*, elles sont décrites par des prédicats. Quant aux motifs, ils permettent de représenter des relations sémantiques indépendantes de l'arité de la contrainte. Les contraintes dont les relations sont définies par des motifs sont appelées *contraintes globales*.

Une instance est dite *normalisée* s'il n'existe pas deux contraintes ayant la même portée. Par la suite, nous ne considérons que des instances normalisées. L'*arité* de c_i est définie par le nombre de variables impliquées par la contrainte, c'est-à-dire $|S(c_i)|$. Une contrainte est dite *unaire* si elle est d'arité un, *binaire* si elle est d'arité deux, *n-aire* ou *non binaire* sinon. Une instance est dite *binaire* si toutes ses contraintes sont unaires ou binaires, *n-aire* ou *non binaire* sinon. On dit qu'elle est d'arité a si a est l'arité maximale de ses constraintes. Dans le cas particulier des instances binaires, nous notons c_{ij} la contrainte liant x_i et x_j. De plus, pour certaines définitions relationnelles, en l'absence d'une contrainte c_{ij} explicitement définie dans C, on considérera la contrainte c_{ij} comme une contrainte *universelle* (c'est-à-dire une contrainte qui autorise toutes les combinaisons de valeurs et donc pour laquelle on a $R(c_{ij}) = d_{x_i} \times d_{x_j}$).

Exemple 1.1 *Considérons l'instance CSP binaire $P = (X, D, C)$ avec :*

- $X = \{x_1, x_2, x_3, x_4\}$,
- $D = \{d_{x_1}, d_{x_2}, d_{x_3}, d_{x_4}\}$, *avec* $d_{x_1} = d_{x_2} = d_{x_3} = d_{x_4} = \{1, 2, 3\}$,
- $C = \{c_{12}, c_{13}, c_{23}, c_{24}, c_{34}\}$ *avec :*
  - $c_{12} = (S(c_{12}), R(c_{12}))$ *où* $S(c_{12}) = \{x_1, x_2\}$ *et* $R(c_{12}) = \{(v_1, v_2) \in d_{x_1} \times d_{x_2} | v_1 = v_2\}$.
  - $c_{13} = (S(c_{13}), R(c_{13}))$ *où* $S(c_{13}) = \{x_1, x_3\}$ *et* $R(c_{13}) = \{(v_1, v_3) \in d_{x_1} \times d_{x_3} | v_1 = v_3\}$.
  - $c_{23} = (S(c_{23}), R(c_{23}))$ *où* $S(c_{23}) = \{x_2, x_3\}$ *et* $R(c_{23}) = \{(v_2, v_3) \in d_{x_2} \times d_{x_3} | v_2 = v_3\}$.
  - $c_{24} = (S(c_{24}), R(c_{24}))$ *où* $S(c_{24}) = \{x_2, x_4\}$ *et* $R(c_{24}) = \{(v_2, v_4) \in d_{x_2} \times d_{x_4} | v_2 > v_4\}$.
  - $c_{34} = (S(c_{34}), R(c_{34}))$ *où* $S(c_{34}) = \{x_3, x_4\}$ *et* $R(c_{34}) = \{(v_3, v_4) \in d_{x_3} \times d_{x_4} | v_3 > v_4\}$.

La contrainte c_{14} est une contrainte universelle.

La structure d'une instance CSP est représentée par un hypergraphe (un graphe dans le cas binaire), appelé *hypergraphe de contraintes*, dont les sommets correspondent aux variables et les hyperarêtes aux portées des contraintes. Pour simplifier les notations, dans la suite, nous notons l'hypergraphe de contraintes $(X, \{S(c_1), \ldots, S(c_e)\})$ par (X, C). L'emploi d'un (hyper)graphe pour représenter la structure conduit naturellement à exploiter fréquemment une partie de la terminologie définie en théorie des (hyper)graphes, comme les notions de voisinage ou de degré.

Exemple 1.2 *La figure 1.1 présente le graphe de contraintes associé à l'instance CSP de l'exemple 1.1.*

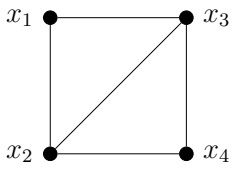


FIGURE 1.1 – Le graphe de contraintes associé à l'instance CSP de l'exemple 1.1

Si l'hypergraphe de contraintes permet de représenter les interactions entre les variables via les contraintes, il est également possible, dans le cas des instances binaires, de décrire sous forme d'un graphe les interactions entre les valeurs via les relations. Ce graphe, dont les sommets sont des couples variable-valeur et dont les arêtes expriment la compatibilité des valeurs, est appelé *microstructure* de l'instance.

Définition 1.2 (Microstructure (Jégou, 1993a)) *Soit une instance CSP binaire $P = (X, D, C)$.*
La microstructure de P (notée $\mu(P)$) est le graphe $(X_{\mu(P)}, E_{\mu(P)})$ avec :

- $X_{\mu(P)} = \{(x_i, v_i) \mid x_i \in X \text{ et } v_i \in d_{x_i}\}$
- $E_{\mu(P)} = \{((x_i, v_i), (x_j, v_j)) \mid i \neq j, \text{ soit } c_{ij} \notin C, \text{ soit } (c_{ij} \in C \text{ et } (v_i, v_j) \in R(c_{ij}))\}$.

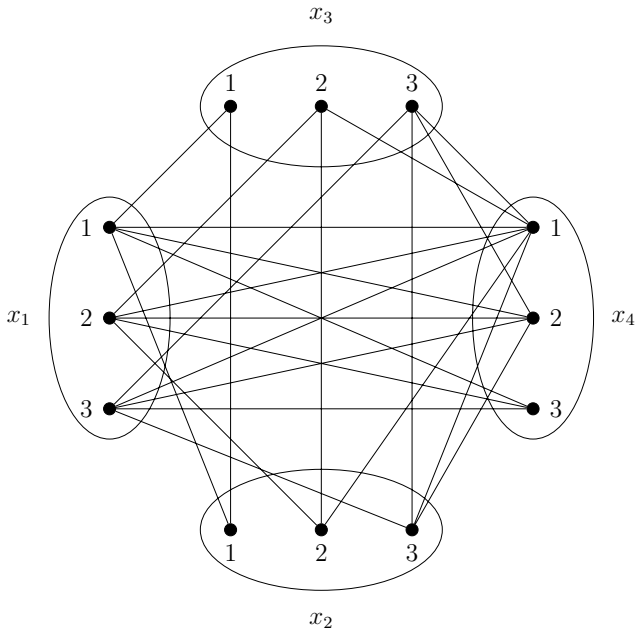


FIGURE 1.2 – La microstructure de l'instance CSP de l'exemple 1.1.

Exemple 1.3 *La figure 1.2 présente la microstructure de l'instance CSP de l'exemple 1.1.*

Une *affectation* ou *instanciation* sur un sous-ensemble $Y = \{x_{i_1}, \ldots, x_{i_k}\}$ de X est un k-uplet $(v_{i_1}, \ldots, v_{i_k})$ de $d_{x_{i_1}} \times \ldots \times d_{x_{i_k}}$, que nous noterons également $\langle x_{i_1} = v_{i_1}, \ldots, x_{i_k} = v_{i_k} \rangle$. Une affectation est dite *complète* si elle porte sur toutes les variables de l'instance, *partielle* sinon. Une affectation $\mathcal{A}$ sur un sous-ensemble Y de X est dite *cohérente* si elle ne viole aucune contrainte, c'est-à-dire si $\forall c \in C, S(c) \subseteq Y, \mathcal{A}[S(c)] \in R(c)$ avec $\mathcal{A}[Y']$ la restriction de A aux variables de Y'. Elle sera dite *incohérente* sinon. Une affectation partielle cohérente est appelée *solution partielle*. Une *solution* est une affectation complète cohérente, c'est-à-dire une affectation de chacune des variables avec une valeur prise dans leur domaine respectif qui satisfait toutes les contraintes. Une instance est dite *satisfiable* ou *satisfaisable* si elle possède au moins une solution, *insatisfiable* ou *insatisfaisable* sinon. Nous pouvons maintenant rappeler la définition du problème de décision CSP (au sens de la théorie de la complexité (Garey et Johnson, 1979)) :

Définition 1.3 (Problème de décision CSP)

- *Donnée : une instance CSP* (X, D, C),
- *Question : l'instance CSP* (X, D, C) *possède-t-elle une solution ?*

Ce problème de décision est connu pour être NP-complet. Aussi, pour le résoudre, différentes voies ont été explorées. Une première consiste à proposer des algorithmes de résolution généraux dont la complexité en temps sera, dans le pire des cas, exponentielle. Une seconde repose sur l'exploitation des classes polynomiales du problème CSP.

1.2 Résolution dans le cas général

Dans le cas général, on distingue souvent trois catégories d'algorithmes pour la résolution du problème CSP :

- les algorithmes de recherche énumératifs [1],

1. Le terme ≪ énumératif ≫ pouvant avoir plusieurs interprétations, nous entendons ici énumération de toutes les affectations possibles, et non de toutes les solutions.

- les algorithmes reposant sur la programmation dynamique (Bertelè et Brioschi, 1972),
- les algorithmes de recherche locale (Hoos et Stützle, 2004; Hoos et Tsang, 2006).

Les deux premières approches conduisent à des algorithmes dits *complets* dans la mesure où ils garantissent de répondre positivement à la question d'existence d'une solution, s'il en existe une, et de répondre négativement dans le cas contraire. Par contre, si les algorithmes de recherche locale sont souvent très efficaces en pratique pour trouver des solutions, ils ne permettent pas de conclure à l'absence de solution, et pour cette raison, sont généralement qualifiés d'*incomplets*. Dans cette section, nous nous concentrons sur les algorithmes de recherche énumératifs. Ceux-ci reposant généralement sur un savant dosage entre recherche et simplification du problème, nous présentons, dans la suite de cette section, les principales notions de cohérence exploitées pour simplifier les instances, puis quelques-uns des algorithmes de résolution de l'état de l'art. L'approche par programmation dynamique sera évoquée ultérieurement dans le chapitre 3, tandis que les algorithmes de recherche locale ne seront pas abordés car hors du champ de ce manuscrit.

1.2.1 Cohérences locales et filtrages

Résoudre une instance CSP quelconque conduit souvent à explorer un espace de recherche de taille potentiellement exponentielle. Aussi, simplifier l'instance en amont ou durant la résolution peut s'avérer fondamental d'un point de vue pratique. De nombreuses notions de cohérence ont été définies dans la littérature (Apt, 2003; Dechter, 2003; Rossi et al., 2006; Lecoutre, 2009). Généralement, ces cohérences sont définies par des propriétés que doit satisfaire localement l'instance CSP considérée et sont accompagnées d'algorithmes de filtrages qui visent à interdire les combinaisons de valeurs ne satisfaisant pas ces propriétés. Une des plus simples et des plus usitées en pratique est la cohérence d'arc (AC (Mackworth, 1977)). Elle consiste à assurer la compatibilité de chaque valeur de chaque variable x avec les contraintes auxquelles la variable x participe. Formellement :

Définition 1.4 (Cohérence d'arc (Mackworth, 1977)) *Soit une instance CSP $P = (X, D, C)$.*

- *Une valeur v de d_x est* arc-cohérente *si pour chaque contrainte c telle que $x \in S(c)$, il existe un tuple $t \in R(c)$ tel que $t[\{x\}] = v$ et $\forall y \in S(c),\ t[\{y\}] \in d_y$ (t est alors appelé* support *de v vis-à-vis de c).*
- *Une variable x est* arc-cohérente *si chaque valeur v de d_x est arc-cohérente.*
- *P est dite* arc-cohérente *si chaque variable de P est arc-cohérente.*

Le filtrage par cohérence d'arc consiste à supprimer toutes les valeurs ne satisfaisant pas l'arc-cohérence. De nombreux algorithmes de filtrage par arc-cohérence ont été proposés parmi lesquels nous pouvons citer AC3 (Mackworth, 1977), AC4 (Mohr et Henderson, 1986), AC6 (Bessière, 1994), AC8 (Chmeiss et Jégou, 1998), AC3.1/AC2001 (Bessière et al., 2005) ou AC3^{rm} (Lecoutre et al., 2008). Ces algorithmes se distinguent les uns des autres par les mécanismes et les structures de données plus ou moins complexes qu'ils mettent en œuvre pour établir la cohérence d'arc. Il est résulte des complexités en temps et en espace différentes. Par exemple, dans le cas des instances binaires, les algorithmes AC6 et AC2001 ont une complexité optimale en temps en $O(ed^2)$ pour une complexité en espace en $O(ed)$ avec d la taille du plus grand domaine. D'un point de vue pratique, le filtrage par cohérence d'arc constitue souvent un bon compromis entre temps d'exécution et capacité à supprimer des valeurs ne participant pas à des solutions. Son emploi est donc des plus fréquents en prétraitement comme à chaque étape de la recherche.

Par ailleurs, de manière plus générale, la plupart des cohérences employées en pratique s'intéressent à la suppression de valeurs ne pouvant pas participer à une solution. De ce fait, elles sont souvent appelées *cohérences de domaine*. La cohérence d'arc en est un exemple, mais d'autres formes plus fortes ont été proposées (notamment SAC (Debruyne et Bessière, 1997a), Max-RPC (Debruyne et Bessière, 1997b) ou PIC (Freuder et Elfe, 1996)). Elles ont l'avantage de supprimer souvent plus de valeurs que la cohérence d'arc (au détriment d'une complexité en temps et d'un temps d'exécution plus importants) sans pour autant altérer la structure de l'instance considérée.

Au-delà de la suppression de valeurs, il est possible de supprimer des combinaisons de valeurs ne participant pas à des solutions. C'est le cas, par exemple, de la cohérence de chemin (Montanari, 1974), définie pour les instances binaires, qui interdit certains couples de valeurs.

Définition 1.5 (Cohérence de chemin (Montanari, 1974)) *Soit une instance binaire $P = (X, D, C)$.*

- *Un couple de variables* (x, y) *est* chemin-cohérent *si* $\forall (v, w) \in R(c_{xy}), \exists u \in d_z, (v, u) \in R(c_{xz})$ *et* $(w, u) \in R(c_{yz})$.
- P *est dite* chemin-cohérente *si tout couple* (x, y) *de variables est chemin-cohérent.*

Le filtrage par cohérence de chemin permet donc de supprimer certains couples de valeurs des relations de certaines contraintes. Il est possible que certains de ces couples soient issus d'une contrainte universelle. Dans ce cas-là, la contrainte est alors explicitement ajoutée au problème avec une relation autorisant tous les couples sauf ceux interdits par la cohérence de chemin. Ainsi, la cohérence de chemin est susceptible d'altérer la structure du problème. Comme pour la cohérence d'arc, de nombreux algorithmes ont été proposés (par exemple (Mackworth, 1977; Han et Lee, 1988; Chmeiss et Jégou, 1998; Bessière et al., 2005; Lecoutre et al., 2007a,b, 2011)). La complexité en temps de ces algorithmes (au moins en $O(n^3d^3)$) explique que la cohérence de chemin ne soit utilisée principalement qu'en prétraitement.

Freuder (Freuder, 1978) a proposé des formes de cohérence encore plus fortes :

Définition 1.6 (k-cohérence et k-cohérence forte (Freuder, 1978)) *Une instance CSP est dite* k-cohérente *si toute affectation cohérente de* $k-1$ *variables peut être étendue en une instanciation cohérente sur* k *variables. Une instance CSP vérifie la* k-cohérente forte *si elle vérifie la* i*-cohérence pour tout* i *inférieur ou égal à* k.

Dans le cas des instances CSP binaires, la cohérence d'arc et la cohérence de chemin correspondent respectivement à la 2-cohérence et à la 3-cohérence. Un algorithme optimal pour établir la k-cohérence est présenté dans (Cooper, 1989). Sa complexité en temps est en $O(n^k d^k)$, ce qui explique que la k-cohérence soit peu usitée en pratique pour des valeurs de k supérieures à 3. De plus, à partir de k égal à 3, la k-cohérence est susceptible de générer de nouvelles contraintes d'arité $k-1$ et donc de modifier la structure de l'instance. En particulier, une instance initialement binaire peut ne plus l'être après application de la k-cohérence pour k supérieur strictement à 3. À partir de de la cohérence forte, il est possible de définir la cohérence globale :

Définition 1.7 (Cohérence globale) *Une instance CSP est dite* globalement cohérente *si elle est fortement* n*-cohérente.*

La cohérence globale constitue ainsi la forme la plus puissante de cohérence. Son intérêt principal réside dans le fait que toute instance globalement cohérente peut être résolue en temps polynomial (Freuder, 1982).

Par ailleurs, la k-cohérence forte n'est pas la seule cohérence susceptible de supprimer des tuples. En effet, d'autres cohérences supprimant des tuples ont été définies, comme par exemple l'inter-cohérence (PWC (Janssen et al., 1989)).

Définition 1.8 (Inter-cohérence (Janssen et al., 1989)) *Une instance CSP* $P = (X, D, C)$ *est* intercohérente *(pairwise-consistent en anglais) si* $\forall\ 1 \leq i \leq e$, $R(c_i) \neq \emptyset$ *et* $\forall\ 1 \leq i < j \leq e$, $R(c_i)[S(c_i) \cap S(c_j)] = R(c_j)[S(c_i) \cap S(c_j)]$ *avec* $R(c)[Y]$ *la projection de* $R(c)$ *sur les variables de* Y.

De plus, à l'image de la généralisation de la cohérence d'arc par la k-cohérence, il est possible de généraliser cette cohérence en considérant k contraintes :

Définition 1.9 (Hyper-k-cohérence et hyper-k-cohérence forte (Jégou, 1993b)) *Une instance CSP* $P = (X, D, C)$ *est dite* hyper-k-cohérente *si pour tout sous-ensemble de* k *contraintes, on a :*

$$\bowtie_{i=1}^{k-1} R(c_i)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)] \subseteq R(c_k)[(\bigcup_{i=1}^{k-1} S(c_i)) \cap S(c_k)]$$

P *vérifie l'*hyper-k-cohérente forte *si elle est hyper-*i*-cohérente pour tout* i *inférieur ou égal à* k.

Appliquer l'hyper-k-cohérence s'effectue en $O(r^k)$ avec r la taille maximale des relations associées aux contraintes. Aussi, il semble difficile d'exploiter en pratique un tel niveau de cohérence, hormis pour $k = 2$, c'est-à-dire l'inter-cohérence.

Enfin, dans une certaine mesure, les algorithmes de filtrage peuvent être vus comme des méthodes de résolution. Toutefois, il s'agit alors de méthodes de résolution incomplètes car elles ne peuvent résoudre que certaines catégories d'instances. Nous reviendrons sur ce point dans la section 1.3. Dans le cas général, la résolution s'effectue souvent par le biais d'algorithmes énumératifs dont nous rappelons les concepts dans la sous-section suivante.

1.2.2 Algorithmes de résolution énumératifs

D'un point de vue pratique, les algorithmes de résolution énumératifs constituent, à l'heure actuelle, une des meilleures alternatives pour la résolution du problème CSP. Ils réalisent une exploration systématique de l'espace de recherche via un parcours en profondeur d'abord. Pour cela, ils essaient de construire progressivement une solution à partir de zéro en prenant, à chaque étape, une nouvelle décision et en testant si la suite de décisions obtenue respecte un certain niveau de cohérence.

L'algorithme Backtrack (noté BT) constitue l'algorithme de base. Partant d'une affectation partielle $\mathcal{A}$ (initialement vide), BT choisit d'abord une variable x parmi les variables non instanciées puis une valeur v dans le domaine d_x de x. Ensuite, il teste si l'affectation $\mathcal{A}$ augmentée de l'affectation $x = v$ (c'est-à-dire $\mathcal{A} \cup \langle x = v \rangle$) correspond à une affectation cohérente (c'est-à-dire ne violant aucune contrainte). Si c'est le cas, la recherche se poursuit en choisissant une nouvelle variable parmi les variables non instanciées. Dans le cas contraire, il remet en cause le choix de la valeur v en choisissant une nouvelle valeur v' de d_x et teste l'affectation $\mathcal{A} \cup \langle x = v' \rangle$. Si toutes les valeurs de d_x ont été testées sans succès, cela signifie qu'aucune extension de l'affectation $\mathcal{A}$ ne peut conduire à une solution. Aussi, BT remet en cause le choix de valeur pour la variable instanciée juste avant x. Ce faisant, il effectue, ce qu'on appelle un *retour en arrière chronologique*. Au final, en procédant ainsi, soit BT parvient à instancier chacune des variables de façon cohérente, trouvant par la même une solution, soit, au fil du temps, il finit par explorer entièrement l'espace de recherche sans succès, prouvant ainsi l'absence de solution pour l'instance considérée. Sa complexité en temps est en $O(e.a.d^n)$ pour une instance d'arité a ayant n variables avec au plus d valeurs par domaine et e contraintes et en supposant que le test d'appartenance d'un tuple à une relation s'effectue en $O(a)$.

Hormis pour des instances très particulières, cet algorithme s'avère inefficace en pratique. Aussi, de multiples voies ont été explorées pour améliorer les performances des algorithmes énumératifs et les rendre opérationnels sur des instances du monde réel dont la taille peut être significative. Il en découle qu'à l'heure actuelle, les solveurs sont généralement caractérisés par les choix qu'ils font concernant notamment :

- le type de décisions prises,
- l'emploi de cohérence locale,
- le type de retour en arrière,
- l'enregistrement de nouvelles informations,
- le choix de la prochaine variable ou valeur à instancier,
- l'exploitation de redémarrage.

Il est à noter que ces choix sont tout aussi stratégiques les uns que les autres. Aussi, la définition de solveurs efficaces nécessite souvent de les combiner savamment.

1.2.2.1 Type de décisions prises et stratégie de branchement

Principalement, les décisions prises peuvent être de deux types :

- des *décisions positives* $x_i = v_i$ qui affectent la valeur v_i à la variable x_i,
- des *décisions négatives* $x_i \neq v_i$ qui assurent que x_i ne pourra pas être affectée à la valeur v_i.

Notons qu'une suite de décisions positives correspond alors tout simplement à la notion d'affectation.

Certains algorithmes (par exemple (Sabin et Freuder, 1994; Lecoutre et al., 2007c)) exploitent les deux types de décisions simultanément. Ainsi, en cas d'échec (c'est-à-dire si aucune solution n'est trouvée) pour une décision donnée, ils vont prendre la décision opposée, à savoir la décision $x_i \neq v_i$ si la décision initiale était $x_i = v_i$, $x_i = v_i$ sinon. Il en résulte que la trace d'exécution de ces algorithmes décrit un arbre (appelé *arbre de recherche*) binaire. Ces algorithmes sont alors dits *à branchement binaire*.

Par ailleurs, d'autres algorithmes (comme l'algorithme Backtrack) n'exploitent que des décisions positives. Ainsi, en cas d'échec pour une décision donnée $x_i = v_i$, ils vont prendre une décision $x_i = v'_i$ en essayant une nouvelle valeur v'_i prise dans le domaine de x_i, et cela jusqu'à trouver une solution ou avoir essayé toutes les valeurs du domaine. Dans ce cas-là, la trace d'exécution est représentée par un arbre non binaire (ou encore d-aire en faisant allusion au paramètre d représentant la taille du plus grand domaine). De tels algorithmes sont alors appelés *à branchement non binaire*.

Les algorithmes exploitant un branchement binaire semblent plus généraux dans la mesure où ils peuvent simuler le comportement de leur pendant avec un branchement non binaire. D'un point de vue théorique, il a d'ailleurs été montré que lorsqu'il s'agit de prouver l'absence de solution, les algorithmes exploitant un branchement binaire sont plus efficaces que leur équivalent utilisant un branchement non binaire (Mitchell, 2003; Hwang et Mitchell, 2005). Cependant, d'un point de vue pratique, il semble plus difficile d'être aussi catégorique. En effet, l'efficacité des solveurs dépend de plusieurs choix stratégiques et peut varier fortement selon les instances que l'on souhaite résoudre.

1.2.2.2 Emploi de cohérence locale

Compte tenu de la NP-complétude du problème CSP, il est fondamental de détecter au plus tôt les incohérences, et ainsi d'éviter de visiter inutilement certaines parties de l'espace de recherche. Une solution pour atteindre ce but réside dans l'emploi d'une certaine forme de cohérence locale. Bien souvent, il s'agit alors d'appliquer, après chaque décision, un algorithme de filtrage afin de maintenir en chaque nœud de l'arbre de recherche le niveau de cohérence locale souhaité. Cet algorithme de filtrage est susceptible d'ajouter de nouvelles contraintes à l'instance en cours de résolution. Ces nouvelles contraintes peuvent être unaires (correspondant alors à des suppressions de valeurs dans les domaines) ou d'arité supérieure selon le niveau de cohérence retenu. Dans tous les cas, elles permettent d'expliciter certaines conséquences des décisions prises tout au long de la branche courante de l'arbre de recherche (à savoir la branche allant de la racine jusqu'au nœud courant). Ainsi, elles ne sont valides que pour la branche courante. Lors d'un changement de branche (c'est-à-dire lorsque la dernière décision est remise en cause), il est nécessaire de supprimer toutes les contraintes ajoutées suite à cette décision. Aussi, restaurer l'instance dans sa configuration précédente peut nécessiter des mécanismes et des structures de données plus ou moins complexes, et ainsi engendrer un coût supplémentaire non négligeable. Par conséquent, l'emploi d'une cohérence locale à chaque étape de la recherche peut conduire à diminuer significativement la taille de l'arbre de recherche, mais, en contre-partie, peut également augmenter de manière considérable le coût en chaque nœud de l'arbre. Aussi, le choix du niveau de cohérence locale requiert généralement de trouver un compromis adéquat entre pouvoir de filtrage et coût du filtrage et de la restauration.

Forward-Checking (noté FC (Haralick et Elliot, 1980)) a été un des premiers algorithmes à employer une cohérence locale pour résoudre des instances binaires. La cohérence en question est une forme allégée de la cohérence d'arc. Plus précisement, elle consiste à ne considérer la cohérence d'arc que pour un sous-ensemble de contraintes, à savoir les contraintes n'ayant plus qu'une seule variable non instanciée dans leur portée. Plusieurs généralisations (notées nFC_i, pour i de 0 à 5) aux instances n-aires ont été, par la suite, proposées selon le sous-ensemble de contraintes considéré et la manière dont la cohérence d'arc est appliquée sur ce sous-ensemble (Hentenryck, 1989; Bessière et al., 2002). La complexité en temps de FC est en $O(e.d^n)$ dans le cas binaire et celle de nFC_5, la version non binaire exploitant le filtrage le plus puissant, en $O(e.a.r.d^n)$.

Ensuite, l'étape suivante a naturellement consisté à maintenir, en chaque nœud, la cohérence d'arc. Deux algorithmes ont été définis dans ce but : RFL (pour Real Full Look-ahead (Nadel, 1988), voir l'algorithme 1.1) qui exploite un branchement non binaire et MAC (pour Maintaining Arc Consistency (Sabin et Freuder, 1994, 1997), voir l'algorithme 1.2) qui repose sur un branchement binaire. Il est à noter que, dans la littérature, la distinction entre MAC et RFL n'est pas toujours faite. Ainsi, il n'est pas rare que le terme MAC soit employé pour faire référence à RFL. D'un point de vue pratique, FC a longtemps été considéré comme plus performant que MAC ou RFL. Cependant, les améliorations des algorithmes de filtrages par cohérence d'arc et l'emploi de structures de données moins coûteuses ont permis d'inverser cette tendance. Il en résulte que la cohérence d'arc est désormais le niveau minimum de cohérence à maintenir. RFL et MAC ont une complexité en temps en $O(e.d^{n+1})$ dans le cas binaire et en $O(e.a.r.d^n)$ dans le cas non binaire.

Au-delà de la cohérence d'arc, d'autres cohérences de domaine (notamment SAC dans (Lecoutre et Prosser, 2006; Lecoutre, 2009), PIC dans (Debruyne, 2000), RPC et Max-RPC dans (Stergiou, 2015)) ont été exploitées. L'emploi de cohérence plus forte (comme SPC par exemple) est bien sûr possible mais ne semble pas pertinent pour toutes les instances pour des raisons évidentes de coût.

1.2.2.3 Retour en arrière et enregistrement d'informations

Durant la recherche, tout algorithme énumératif est amené à remettre en cause certaines décisions prises concernant la branche courante aussitôt qu'il détecte l'impossibilité pour cette branche de conduire à une so-

Algorithme 1.1 : RFL(P, Σ, V)

Entrées : $P = (X, D, C)$: CSP ; Σ : suite de décisions positives, V : ensemble de variables
Résultat : $vrai$ si Σ peut être étendue de façon cohérente aux variables de V, $faux$ sinon

```
si V = ∅ alors
    retourner vrai
sinon
    Choisir une variable x ∈ V
    d'_x ← d_x
    tant que d'_x ≠ ∅ faire
        Choisir une valeur v ∈ d'_x
        d'_x ← d'_x \ {v}
        si AC (P,Σ ∪ ⟨x = v⟩) ∧ RFL(P, Σ ∪ ⟨x = v⟩, V\{x}) alors retourner vrai
    retourner faux
```

Algorithme 1.2 : MAC (P, Σ, V)

Entrées : $P = (X, D, C)$: CSP ; Σ : suite de décisions, V : ensemble de variables
Résultat : $vrai$ si Σ peut être étendue de façon cohérente aux variables de V, $faux$ sinon

```
si V = ∅ alors
    retourner vrai
sinon
    Choisir une variable x ∈ V
    Choisir une valeur v ∈ d_x
    d_x ← d_x \ {v}
    si AC (P,Σ ∪ ⟨x = v⟩) ∧ MAC(P, Σ ∪ ⟨x = v⟩, V\{x}) alors retourner vrai
    sinon
        si AC (P,Σ ∪ ⟨x ≠ v⟩) alors
            retourner MAC(P,Σ ∪ ⟨x ≠ v⟩,V)
        sinon retourner faux
```

lution (un tel cas de figure est appelé *impasse*). Lorsqu'une impasse survient, la première possibilité explorée consiste à modifier la dernière décision selon la stratégie de branchement choisie. Si toutes les décisions possibles pour la stratégie de branchement sélectionnée ont été envisagées sans succès, l'algorithme va devoir remettre en cause les décisions prises auparavant. À ce stade-là, on peut distinguer principalement deux possibilités :

- effectuer un retour en arrière chronologique : l'algorithme remet alors en cause la décision précédant la décision courante sans pour autant avoir de garantie que cette décision soit réellement impliquée dans l'échec rencontré.
- effectuer un retour en arrière *non chronologique* (aussi dit *retour en arrière intelligent* ou *saut en arrière* (Stallman et Sussman, 1977; Gaschnig, 1979; Dechter, 1990; Prosser, 1993; Ginsberg, 1993; Prosser, 1995; Chen, 2000; Jussien et al., 2000)) : il s'agit d'analyser plus ou moins finement les causes de l'échec afin de remettre en cause une décision directement à l'origine de l'échec rencontré.

Exploiter des retours en arrière intelligents permet généralement d'éviter certaines redondances dans la recherche et donc d'explorer inutilement certainement parties de l'espace de recherche.

Une autre alternative pour éviter des redondances consiste à apprendre de ses échecs pour éviter de les reproduire. Dans les faits, elle se traduit souvent par la mémorisation et l'exploitation d'informations explicitées durant la recherche. Parfois, l'information apprise peut traduire la possibilité d'étendre, de façon cohérente, un ensemble de décisions donné sur un certain sous-problème (comme, par exemple, dans (Bayardo et Miranker, 1996; Baget et Tognetti, 2001)). Cependant, dans la grande majorité des travaux (notamment (Stallman et Sussman, 1977; Dechter, 1986; Frost et Dechter, 1994; Schiex et Verfaillie, 1993, 1994; Ginsberg, 1993; Bayardo et Miranker, 1996; Jussien et al., 2000; Katsirelos et Bacchus, 2003, 2005; Lecoutre et al., 2007c)), ces informations représentent plutôt des explorations infructueuses et correspondent alors à la notion de nogood :

Définition 1.10 (Nogood (Lecoutre et al., 2007c)) *Etant donnés une instance CSP $P = (X, D, C)$ et un ensemble de décisions Δ, $P_{|\Delta}$ est l'instance CSP (X, D', C) induite par Δ avec $D' = (d'_{x_1}, \ldots, d'_{x_n})$ tel que pour chaque décision positive $x_i = v_i$ de Δ, $d'_{x_i} = \{v_i\}$ et pour chaque décision négative $x_i \neq v_i$, $d'_{x_i} = d_{x_i} \backslash \{v_i\}$. Pour le cas où x_i n'apparaît pas dans Δ, on a $d'_{x_i} = d_{x_i}$.*
Δ est un nogood *de P si $P_{|\Delta}$ n'a pas de solution.*

Certains nogoods sont triviaux et n'ont que peu d'intérêt. C'est le cas, par exemple, des nogoods dont certaines décisions violent directement une contrainte. D'autres sont le fruit d'une exploration infructueuse d'une partie plus ou moins vaste de l'espace de recherche et, en tant que tel, sont plus intéressants pour éviter de reproduire plusieurs fois la même recherche. Généralement, la découverte de nogoods s'effectue en analysant les causes de l'échec. Aussi, il est très fréquent que l'exploitation de nogoods soit couplée avec l'utilisation de retours en arrière intelligents. Leur mémorisation s'effectue souvent sous la forme de nouvelles contraintes (ou du durcissement de contraintes existantes) qui seront ensuite exploitées de la même manière que les contraintes initiales.

D'un point de vue pratique, une bonne gestion des informations apprises est primordiale, à la fois pour la mémorisation et pour l'exploitation. En effet, la quantité d'informations apprises peut vite devenir importante (par exemple, le nombre de nogoods peut être exponentiel). Aussi, certains travaux (par exemple (Dechter, 1986; Schiex et Verfaillie, 1993, 1994; Frost et Dechter, 1994; Bayardo et Miranker, 1996)) ont restreint l'enregistrement de nogoods afin de maîtriser le coût de leur gestion et l'espace mémoire consommé. La maîtrise du coût passe aussi par l'emploi de structures de données adaptées. C'est notamment le cas dans (Lecoutre et al., 2007c) où l'ensemble des nogoods est représenté sous la forme d'une contrainte globale exploitant la notion de « watched literals » (Moskewicz et al., 2001) introduite dans le cadre de la résolution du problème SAT (satisfiabilité d'une formule booléenne (Biere et al., 2009)).

Enfin, les techniques de retours en arrière intelligents et d'enregistrement d'information peuvent être couplées avec un maintien d'un certain niveau de cohérence locale et conduire à des méthodes de résolution plus ou moins performantes (Prosser, 1993; Schiex et Verfaillie, 1994; Bessière et Régin, 1996; Jussien et al., 2000; Lecoutre et al., 2007c).

1.2.2.4 Choix de la prochaine variable

Pour résoudre une instance CSP, les algorithmes énumératifs ont besoin de choisir une variable à instancier parmi les variables non instanciées et ensuite une valeur dans le domaine de la variable sélectionnée. Le choix de la prochaine variable est de loin le plus important du fait de son influence prépondérante sur la taille de l'arbre de recherche. Le choix de la prochaine valeur a été moins étudié dans la littérature car ayant un impact moindre. Aussi, dans la suite, nous concentrons nos rappels sur le choix de la prochaine variable à instancier.

Trouver un ordre sur les variables conduisant un algorithme énumératif à visiter un arbre de recherche le plus petit possible est une tâche des plus difficiles (Liberatore, 2000). Aussi, la construction de l'ordre est généralement accomplie à l'aide d'heuristiques dont le coût doit être le plus raisonnable possible. De nombreuses heuristiques ont été proposées dans la littérature. Elles sont souvent divisées en deux catégories :

- les heuristiques dites *statiques* pour lesquelles l'ordre est calculé en amont de la résolution et n'évolue pas au cours de celle-ci,
- les heuristiques dites *dynamiques* pour lesquelles l'ordre est calculé notamment en fonction de l'état courant de la recherche et peut ainsi évoluer durant la résolution.

En règle générale, les heuristiques dynamiques sont réputées pour être plus efficaces que les heuristiques statiques (Dechter et Meiri, 1994). Aussi, nous nous intéressons ici aux heuristiques dynamiques généralistes (c'est-à-dire qui ne sont pas spécifiques à une famille d'instances donnée). Depuis la première d'entre elles, l'heuristique `dom` (Golomb et Baumert, 1965) qui choisit, comme prochaine variable, la variable non instanciée ayant le plus petit domaine, de nombreuses améliorations ont été proposées (par exemple (Brélaz, 1979; Bessière et Régin, 1996; Bessière et al., 2001)). Parmi elles, l'heuristique `dom/deg` (Bessière et Régin, 1996) est une des plus simples et des plus efficaces. Elle sélectionne comme prochaine variable celle minimisant le rapport taille courante du domaine sur le degré courant de la variable (c'est-à-dire le nombre de contraintes auxquelles la variable participe). À l'heure actuelle, les heuristiques dites *adaptatives* (Geelen, 1992; Refalo, 2004; Boussemart et al., 2004) sont parmi les toutes meilleures heuristiques généralistes. Elles considèrent non seulement des informations concernant l'état courant de la recherche, mais aussi des informations concernant les états précédents.

Par exemple, l'une d'entre elles, l'heuristique `dom/wdeg` (Boussemart et al., 2004), associe un poids à chaque contrainte (intialement égal à un) et augmente le poids d'une contrainte de un à chaque fois que l'utilisation de cette contrainte, lors d'un filtrage, aboutit à rendre vide un domaine. Elle choisit, comme prochaine variable à instancier, la variable x minimisant le rapport taille courante du domaine de x sur la somme des poids des contraintes impliquant x et au moins une autre variable non instanciée. L'idée sous-jacente est d'essayer d'identifier les parties difficiles de l'instance à résoudre et de les étudier en priorité.

D'une manière ou d'une autre, toutes ces heuristiques vérifient le principe du *first-fail* (Haralick et Elliot, 1980) qui préconise de rencontrer les échecs le plus tôt possible dans la recherche afin de maximiser les chances de pouvoir mener celle-ci à son terme.

1.2.2.5 Exploitation des techniques de redémarrage

En pratique, les algorithmes énumératifs font souvent preuve d'une grande variabilité dans leur comportement d'une instance à l'autre, mais aussi pour une même instance. En particulier, il n'est pas rare qu'une modification mineure dans l'ordre d'affectation des variables permette de réduire significativement la taille de l'arbre de recherche ou au contraire de la faire croître sensiblement. Une des explications possibles réside dans les mauvais choix effectués par l'heuristique de choix de variables qui, en fonction de leur nombre et de leur localisation, sont plus ou moins pénalisants. C'est notamment pour pallier ce défaut potentiel que les techniques de redémarrage ont été introduites (Harvey, 1995; Gomes et al., 2000). L'idée est de stopper la recherche en cours quand celle-ci ne semble pas suffisamment progresser pour ensuite la relancer tout en garantissant d'explorer l'espace de recherche différemment. Plusieurs questions se posent alors :

- quand interrompre la recherche ?
- comment garantir d'explorer différemment l'arbre de recherche ?
- comment garantir la complétude et la terminaison de la recherche ?

Pour la première question, la solution consiste généralement à considérer une certaine mesure (comme le nombre de retours en arrière, le nombre de nœuds visités ou le temps de calcul) et à stopper la recherche dès que la valeur de cette mesure dépasse une limite donnée. Ensuite, pour garantir d'explorer différemment l'arbre de recherche, plusieurs alternatives existent et peuvent être combinées. Les plus courantes procèdent en introduisant une part d'aléa (par exemple en cassant alétoirement les égalités au niveau de l'heuristique de choix de variables) ou en mémorisant des informations concernant la recherche effectuée (comme des nogoods). Enfin, l'assurance qu'un algorithme exploitant des techniques de redémarrage soit complet et termine repose bien souvent sur le fait que la taille de l'espace de recherche restant à explorer décroît strictement au fur et à mesure des redémarrages. Pour atteindre ce but, une première possibilité est de procéder à un enregistrement de nogoods illimité. Chaque nogood mémorisé caractérise alors une partie de l'espace de recherche déjà visitée et garantit qu'elle ne sera plus explorée. Une seconde possibilité, qui peut être utilisée conjointement avec la première, consiste à faire croître au cours du temps la limite à partir de laquelle un redémarrage survient. Elle exploite alors la notion de *stratégie de redémarrage*, qui est une suite d'entiers positifs indiquant les valeurs de cette limite en fonction du nombre de redémarrages déjà effectués. Plusieurs stratégies ont été définies dont une basée sur la suite de Luby (Luby et al., 1993) et une basée sur une suite géométrique (Walsh, 1999).

D'un point de vue pratique, les techniques de redémarrage ont joué un rôle non négligeable dans l'efficacité des solveurs CDCL (pour Conflict-Driven Clause Learning (Marques Silva et al., 2009)) dans le cadre de la résolution du problème SAT. Leur exploitation pour la résolution du problème CSP est plus récente. Nous présentons maintenant l'algorithme MAC+RST+NG (Lecoutre et al., 2007c) que nous exploiterons ensuite dans le chapitre 3. Cet algorithme est, dans les faits, une version de MAC exploitant conjointement des redémarrages et des nogoods. Une fois choisies une variable x_i et une valeur v_i, il commence par considérer la décision positive $x_i = v_i$ avant de s'intéresser à la décision négative $x_i \neq v_i$ en cas d'échec. Ce faisant, il peut exploiter des *nld-nogoods* pour caractériser la partie de l'espace de recherche déjà visitée.

Définition 1.11 (Nld-nogood (Lecoutre et al., 2007c)) *Soit $\Sigma = \langle \delta_1, \ldots, \delta_k \rangle$ une suite de décisions prises le long d'une branche de l'arbre de recherche développé durant la résolution d'une instance CSP P. Pour toute sous-suite $\Sigma' = \langle \delta_1, \ldots, \delta_\ell \rangle$ préfixe de Σ telle que δ_ℓ est une décision négative, l'ensemble $Pos(\Sigma') \cup \{\neg\delta_\ell\}$ est un nogood (appelé un* nld-nogood réduit*) de P avec $\neg\delta_\ell$ la décision positive correspondant à δ_ℓ et $Pos(\Sigma')$ l'ensemble des décisions positives de Σ'.*

Des nld-nogoods réduits ne sont produits qu'au moment où survient un redémarrage en considérant la branche courante. Leur enregistrement garantit alors de ne pas revisiter le même espace de recherche par la suite. Leur représentation sous la forme d'une contrainte globale exploitant la notion de « watched literals » (Moskewicz et al., 2001) permet de les exploiter efficacement et à moindre coût dans le cadre d'un algorithme maintenant la cohérence d'arc comme MAC. En pratique, pour certains types d'instances, MAC+RST+NG s'avère plus efficace que MAC.

1.3 Classes polynomiales

La NP-complétude d'un problème de décision donné n'exclut pas pour autant la possibilité de résoudre certaines de ses instances en temps polynomial. Le problème CSP ne fait pas exception. Nous proposons ici une définition de la notion de *classe polynomiale* pour le problème CSP :

Définition 1.12 (Classe polynomiale) *Une* classe polynomiale *du problème CSP est un ensemble d'instances pour lequel il existe un algorithme capable de résoudre chacune de ses instances en temps polynomial.*

Dans cette définition, seule la polynomialité de la résolution est considérée. Toutefois, dans certains travaux (par exemple (Gottlob et al., 2000)), la polynomialité de la reconnaissance (c'est-à-dire décider si une instance appartient à un ensemble d'instances donné) est également requise pour définir une classe polynomiale. Dans les faits, la nécessité d'avoir une reconnaissance en temps polynomial dépend grandement de l'utilisation faite des classes polynomiales. Par exemple, construire un solveur capable d'exploiter tel ou tel algorithme en fonction de la classe à laquelle appartient l'instance à résoudre implique forcément d'être en mesure de reconnaître efficacement les instances de chaque classe considérée. Par contre, comme nous le verrons par la suite, certains algorithmes généraux comme MAC ou RFL sont capables de résoudre en temps polynomial les instances de certaines classes polynomiales sans pour autant mettre un œuvre de traitement spécifique. Dans ce cas, la reconnaissance n'est pas nécessaire et sa difficulté n'a donc aucune incidence.

Dans la littérature, de nombreuses classes polynomiales du problème CSP ont été proposées. Leur définition s'effectue généralement en posant des conditions restrictives. Ces conditions portent souvent sur les relations associées aux contraintes ou sur la structure de l'hypergraphe de contraintes. Les rappels ci-dessous ne sont pas exhaustifs (voir, par exemple, (Gottlob et al., 2000; Dechter, 2006; Cohen et Jeavons, 2006; Carbonnel et Cooper, 2016) pour plus de détails) et se focalisent sur les classes qui seront utiles pour la suite du propos.

1.3.1 Classes polynomiales relationnelles

Les classes polynomiales dites *relationnelles* sont définies en imposant des conditions restrictives sur les relations. Par exemple, dans la classe polynomiale constituée des instances Zéro-Un-Tous (Cooper et al., 1994), la restriction porte sur le nombre de supports que peut avoir chaque valeur vis-à-vis de chaque contrainte :

Définition 1.13 (Classe ZUT (Cooper et al., 1994)) *Une instance CSP binaire $P = (X, D, C)$ est dite* 0-1-tous *(ou* 0-1-all*) si pour chaque contrainte c_{ij} de C, pour chaque valeur $v_i \in d_{x_i}$, c_{ij} satisfait une des conditions suivantes :*

- *(ZÉRO) pour chaque valeur $v_j \in d_{x_j}$, $(v_i, v_j) \notin R(c_{ij})$,*
- *(UN) il existe une unique valeur $v_j \in d_{x_j}$ telle que $(v_i, v_j) \in R(c_{ij})$,*
- *(TOUS) pour chaque valeur $v_j \in d_{x_j}$, $(v_i, v_j) \in R(c_{ij})$.*

Nous notons ZUT l'ensemble des instances qui sont 0-1-tous.

La cohérence de chemin est une procédure de décision pour les instances de cette classe. Toutefois, il est possible de les résoudre encore plus efficacement en utilisant un algorithme dédié (Cooper et al., 1994; Zhang et al., 1999).

Les instances convexes par rangée (van Beek et Dechter, 1995) constituent également une classe polynomiale pour laquelle la cohérence de chemin est une procédure de décision.

Définition 1.14 (Classe des instances convexes par rangée (van Beek et Dechter, 1995)) *Une instance CSP binaire $P = (X, D, C)$ est dite* convexe par rangée *(row-convex en anglais) par rapport à un ordre $<$ sur les variables et 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 certaines valeurs a_j, $b_j \in d_{x_j}$ où $[a_j..b_j]$ représente l'ensemble des valeurs de d_{x_j} comprises entre a_j et b_j par rapport à l'ordre sur les valeurs.*
Nous notons RC l'ensemble des instances convexes par rangée.

Cette classe a fait l'objet de plusieurs généralisations ou raffinements (par exemple (van Beek et Dechter, 1995; Deville et al., 1999)), mais ceux-ci ne seront pas abordés ici.

Une façon de caractériser les classes polynomiales relationnelles est d'exploiter la notion d'ensemble clos. C'est notamment le cas de la classe Max-closed (Jeavons et Cooper, 1995) :

Définition 1.15 (Classe Max-closed (Jeavons et Cooper, 1995)) *Sous l'hypothèse où chaque domaine est ordonné, une contrainte c est dite* max-closed *si $\forall (v_1, v_2, \ldots, v_{|S(c)|}), (v'_1, v'_2, \ldots, v'_{|S(c)|}) \in R(c)$, $(max(v_1, v'_1), max(v_2, v'_2), \ldots, max(v_{|S(c)|}, v'_{|S(c)|})) \in R(c)$.*
Une instance CSP $P = (X, D, C)$ est dite max-closed *si chacune de ses contraintes est max-closed.*
Nous notons MC l'ensemble des instances max-closed.

Pour cette classe, la cohérence d'arc est une procédure de décision. En effet, si l'application d'un filtrage par cohérence d'arc ne rend vide aucun domaine, alors l'instance possède au moins une solution. Cette classe possède plusieurs particularités. D'un point de vue théorique, il s'agit de la première classe définie en terme de polymorphisme (Jeavons et al., 1997; Cohen et Jeavons, 2006). Sur un plan pratique, les contraintes max-closed incluent toutes les contraintes de base sur les entiers naturels du langage de programmation CHIP (Dincbas et al., 1988) et ont été exploitées dans le cadre du développement d'un outil industriel (Lesaint et al., 1998).

La cohérence d'arc est également une procédure de décision pour la classe suivante introduite dans (Cooper et al., 2010) :

Définition 1.16 (Classe Renommable monotone à droite (Cooper et al., 2010)) *Une instance CSP binaire $P = (X, D, C)$ est dite* renommable monotone à droite *(renamable right monotone en anglais) par rapport à un ordre $<$ sur les variables si, pour $2 \leq j \leq n$, chaque domaine d_{x_j} peut être ordonné par $\preccurlyeq_j$ 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 $v_j \preccurlyeq_j v'_j$ alors $(v_i, v'_j) \in R(c_{ij})$.*
Nous notons RRM l'ensemble de ces instances.

Notons que, pour toutes les classes présentées ci-dessus, la reconnaissance s'effectue en temps polynomial.

1.3.2 Classes polynomiales structurelles

Nous nous intéressons, à présent, aux classes polynomiales *structurelles*, c'est-à-dire aux classes dont la définition repose sur des restrictions imposées à la structure de l'instance (en l'occurrence son hypergraphe de contraintes).

Une des toutes premières classes polynomiales structurelles proposées est celles des instances binaires ayant un graphe de contraintes acyclique.

Définition 1.17 (Classe $TREE$) *La classe $TREE$ est l'ensemble des instances CSP binaires ayant un graphe de contraintes acyclique.*

Pour cette classe, une nouvelle fois, la cohérence d'arc est une procédure de décision (Freuder, 1982). Cette classe peut être généralisée de plusieurs manières différentes. La première consiste à généraliser la notion d'acyclicité dans le cadre des instances d'arité quelconque. Autrement dit, il s'agit alors de considérer la notion d'acyclicité dans les hypergraphes. Cependant, à la différence des graphes pour lesquels la notion d'acyclicité est unique, il existe plusieurs définitions dans le cas des hypergraphes. Nous en rappelons ici deux que nous utiliserons par la suite :

Définition 1.18 (α-acyclicité (Beeri et al., 1983)) *Un hypergraphe (X, C) est* α-acyclique *s'il existe un ordre $(c_1, \ldots, c_e)$ tel que $\forall k, 1 < k \leq e, \exists j < k, (S(c_k) \cap \bigcup_{i=1}^{k-1} S(c_i)) \subseteq S(c_j)$.*

Définition 1.19 (β-acyclicité (Graham, 1979)) *Une séquence $(c_1, ..., c_m, c_{m+1})$ avec $m \geq 3$, telle que les hyperarêtes c_1,..., c_m sont distinctes et $c_1 = c_{m+1}$, est un cycle de Graham si chaque $\Delta_i = S(c_i) \cap S(c_{i+1})$ $(1 \leq i \leq m)$ n'est pas vide, et chaque fois que $i \neq j$, Δ_i et Δ_j sont incomparables (i.e. $\Delta_i \not\subseteq \Delta_j$ et $\Delta_j \not\subseteq \Delta_i$). Un hypergraphe $H = (X, C)$ est* β-acyclique *s'il ne possède pas de cycle de Graham.*

Ces deux notions d'acyclicité ne sont pas indépendantes l'une de l'autre puisque tout hypergraphe β-acyclique est α-acyclique. Elles conduisent à définir deux classes polynomiales, la première incluant donc la seconde :

Définition 1.20 (Classe α-$ACYCLIC$) *La classe α-$ACYCLIC$ est l'ensemble des instances CSP ayant un hypergraphe de contraintes α-acyclique.*

Définition 1.21 (Classe β-$ACYCLIC$) *La classe β-$ACYCLIC$ est l'ensemble des instances CSP ayant un hypergraphe de contraintes β-acyclique.*

L'inter-cohérence est une procédure de décision pour les instances de ces deux classes (Janssen et al., 1989).

La seconde généralisation s'intéresse à mesurer le degré de cyclicité dans l'hypergraphe de contraintes en terme de *largeur*. Dans ce but, plusieurs concepts ont été introduits conduisant ainsi à la définition de plusieurs largeurs. Le premier concept repose sur la notion de décomposition arborescente de graphes :

Définition 1.22 (Décomposition arborescente (Robertson et Seymour, 1986)) *Une* décomposition arborescente *d'un graphe $G = (X, C)$ est un couple (E, T) où $T = (I, F)$ est un arbre (I est l'ensemble des nœuds de T et F l'ensemble de ses arêtes) et $E = \{E_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque sous-ensemble (appelé cluster) E_i est un nœud T et vérifie :*

(i) $\bigcup_{i \in I} E_i = X$,

(ii) pour chaque arête $\{x, y\} \in C$, il existe $i \in I$ avec $\{x, y\} \subseteq E_i$, et

(iii) pour tout $i, j, k \in I$, si k est un chemin de i à j dans T, alors $E_i \cap E_j \subseteq E_k$.

La largeur d'une décomposition arborescente est égale à $max_{i \in I}|E_i| - 1$. La largeur arborescente dite tree-width *w de G est la largeur minimale pour toutes les décompositions arborescentes de G.*

Exemple 1.4 *La figure 1.3 présente un graphe de contraintes (a) et une décomposition arborescente de largeur 3 de ce graphe (b).*

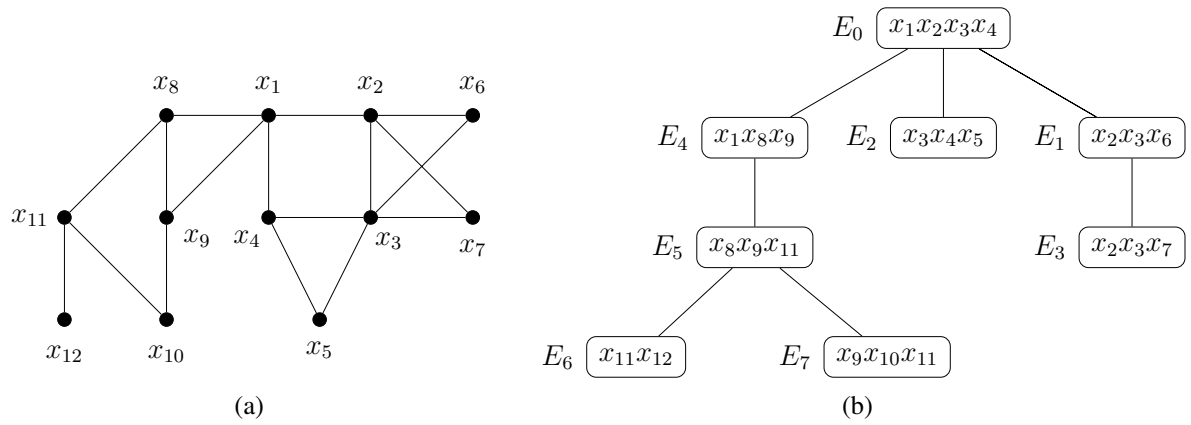


FIGURE 1.3 – Un graphe de contraintes (a) et une décomposition arborescente de largeur 3 de ce graphe (b).

Cette notion peut être étendue aux hypergraphes en considérant leur 2-section [2]. Elle conduit à la définition de la classe polynomiale suivante :

2. La 2-section d'un hypergraphe (X, C) est le graphe (X, C') tel que $C' = \{\{x, y\} | \exists c \in C, \{x, y\} \subseteq c\}$ (Berge, 1973).

Définition 1.23 (Classe BTW_k) *Soit un entier k. La classe BTW_k est l'ensemble des instances dont l'hypergraphe de contraintes a une largeur arborescente bornée par k.*

Notons que BTW_1 est l'ensemble des instances ayant un graphe acyclique. Toute instance de la classe BTW_k peut être résolue en temps polynomial en $O(n.d^{k+1})$ (Freuder, 1990) en utilisant, par exemple, la cohérence adaptative ou la méthode Tree-Clustering (Dechter et Pearl, 1989).

Si la décomposition arborescente vise à regrouper ensemble certaines variables pour garantir la polynomialité de la résolution, il est possible de parvenir au même résultat en regroupant les contraintes à travers, par exemple, la notion de décomposition hyperarborescente :

Définition 1.24 (Décomposition hyperarborescente (Gottlob et al., 1999b, 2000)) *Etant donné un hypergraphe $H = (X, C)$, un* hyperarbre pour l'hypergraphe *H est un triplet (T, χ, λ) où $T = (N, F)$ est un arbre enraciné, et χ et λ sont des fonctions d'étiquetage qui associent à chaque sommet $p \in N$ deux ensembles $\chi(p) \subseteq X$ et $\lambda(p) \subseteq C$. Si $T' = (N', F')$ est un sous-arbre de T, on note $\chi(T') = \bigcup_{v \in N'} \chi(v)$. L'ensemble de sommets N de T est noté $sommets(T)$, et la racine de T $racine(T)$. De plus, pour chaque $p \in N$, T_p désigne le sous-arbre de T enraciné en p.*
Une décomposition en hyperarbre, ou décomposition hyperarborescente *de H est un hyperarbre $HD = (T, \chi, \lambda)$ pour H qui satisfait les conditions suivantes :*

1. *pour chaque arête $c \in C, \exists p \in sommets(T)$ tel que $c \subseteq \chi(p)$,*
2. *pour chaque sommet $x \in X$, l'ensemble $\{p \in sommets(T) : x \in \chi(p)\}$ induit un sous-arbre (connexe) de T,*
3. *pour chaque $p \in sommets(T), \chi(p) \subseteq \bigcup_{c \in \lambda(p)} c$,*
4. *pour chaque $p \in sommets(T), \bigcup_{c \in \lambda(p)} c \cap \chi(T_p) \subseteq \chi(p)$.*

Une arête $c \in C$ est fortement couverte dans HD s'il existe $p \in sommets(T)$ tel que $c \subseteq \chi(p)$ et $c \in \lambda(p)$. Une décomposition en hyperarbre HD est une décomposition complète *de H si chaque arête de H est fortement couverte dans HD. La* largeur *h d'une décomposition en hyperarbre $HD = (T, \chi, \lambda)$ est $max_{p \in sommets(T)} |\lambda(p)|$. La* largeur d'hyperarbre *ou* largeur hyperarborescente *hw de H est la largeur minimum parmi toutes ses décompositions en hyperarbre.*

Sous l'hypothèse où, pour chaque contrainte, la relation associée est définie en extension, cette notion conduit naturellement à définir la classe polynomiale suivante :

Définition 1.25 (Classe $BHTW_k$) *Soit un entier k. La classe $BHTW_k$ est l'ensemble des instances dont l'hypergraphe de contraintes a une largeur hyperarborescente bornée par k.*

Gottlob et al. ont montré que toute instance appartenant à la classe BTW_k pour un entier k donné appartient à la classe $BHTW_{k+\delta}$ pour un certain entier δ (Gottlob et al., 2000). Plus récemment, Grohe a établi que, pour les instances ayant une arité bornée par une constante, la seule raison d'être traitable en temps polynomial est d'avoir une largeur arborescente bornée si $FPT \neq W[1]$ et que la classe de structures soit récursivement énumérable (Grohe, 2007). Notons qu'il est possible de déterminer en temps polynomial s'il existe une décomposition (hyper)-arborescente de largeur k (Gottlob et al., 2000). Par contre, déterminer la largeur (hyper)-arborescente d'un (hyper)graphe constitue un problème NP-difficile (Arnborg et al., 1987; Gottlob et al., 2000).

Enfin, d'autres concepts, et donc d'autres largeurs (voir par exemple (Gottlob et al., 2000; Grohe et Marx, 2014)), ont été exploités pour définir des classes polynomiales et ont principalement un intérêt théorique, en particulier quand l'arité maximale des contraintes n'est pas bornée par une constante.

1.3.3 Classes polynomiales hybrides

Les classes polynomiales *hybrides* sont des classes polynomiales pour lesquelles les restrictions ne concernent pas uniquement les relations ou la structure. Dans le cas des instances binaires, elles reposent souvent sur des propriétés particulières de la microstructure. En effet, la microstructure tient compte à la fois des interactions entre les variables via les contraintes et des interactions des valeurs des variables via les relations. Ainsi, elle peut

permettre d'exploiter des propriétés structurelles ou relationnelles et même au-delà. De plus, il existe une transformation polynomiale transformant les instances binaires du problème CSP en instances du problème CLIQUE. Elle s'appuie sur le théorème suivant qui établit la correspondance entre solutions et cliques maximales de taille n dans la microstructure :

Théorème 1.1 ((Jégou, 1993a)) *Soit une instance CSP binaire P.*
Une affectation $(v_1, \ldots, v_n)$ de X est une solution de P ssi $\{(x_1, v_1), \ldots, (x_n, v_n)\}$ est une n-clique de $\mu(P)$.

Bien que le problème CLIQUE soit NP-complet, il est tout de même possible d'exploiter ce résultat via les classes polynomiales du problème CLIQUE, comme, par exemple, la classe des graphes triangulés (Berge, 1960; Golumbic, 1980). Un graphe est dit *triangulé* ou *chordal* s'il n'existe pas de cycle de longueur supérieure ou égale à quatre sans corde (c'est-à-dire d'arête entre deux sommets non consécutifs dans le cycle). Le calcul de toutes les cliques maximales d'un graphe triangulé pouvant s'effectuer en temps linéaire (Gavril, 1972), il est possible de définir deux classes polynomiales pour le problème CSP :

Définition 1.26 (Classe des instances ayant une microstructure triangulée (Jégou, 1993a))
La classe CM est définie par l'ensemble des instances binaires ayant une microstructure triangulée.

Définition 1.27 (Classe des instances ayant un complémentaire de microstructure triangulé (Cohen, 2003))
Le complémentaire *d'un graphe $G = (X, E)$ est le graphe (X, E') avec $E' = \{\{x, y\} | x, y \in X \ t.q. \ \{x, y\} \notin E\}$.*
La classe CCM est définie par l'ensemble des instances binaires dont le complémentaire de la microstructure est triangulé.

La classe CM peut être généralisée en considérant la notion de graphe parfait (les graphes triangulés étant des graphes parfaits). Un graphe est dit *parfait* s'il ne comporte ni un cycle, ni le complémentaire d'un cycle de longueur impaire et supérieure ou égale à cinq (Berge, 1961; Chudnovsky et al., 2006). En exploitant des résultats récents de théorie des graphes (Chudnovsky et al., 2005, 2006), Salamon et Jeavons (Salamon et Jeavons, 2008) ont défini la classe des instances binaires ayant une microstructure parfaite :

Définition 1.28 (Classe des instances ayant une microstructure parfaite (Salamon et Jeavons, 2008))
La classe PM est définie par l'ensemble des instances binaires ayant une microstructure parfaite.

La reconnaissance des instances de ces classes peut s'effectuer en temps polynomial (Tarjan et Yannakakis, 1984; Chudnovsky et al., 2005).

Nous rappelons, à présent, une classe polynomiale hybride qui joue, à l'heure actuelle, un rôle important et qui a suscité et suscite encore de nombreux travaux. Elle consiste à interdire l'existence de *triangle cassé* dans la microstructure.

Définition 1.29 (Classe Broken Triangle Property (Cooper et al., 2008, 2010)) *Soient une instance CSP binaire P et un ordre $<$ sur les variables de P.*

- *Une paire de valeurs $v'_k, v''_k \in D(x_k)$ satisfait BTP si pour chaque couple de variables (x_i, x_j) 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* $(v_j, v'_k) \in R(C_{jk})$.
- *Une variable x_k satisfait BTP si chaque paire de valeurs de d_{x_k} satisfait BTP.*
- *P satisfait BTP par rapport à l'ordre $<$ si chacune de ses variables satisfait BTP.*

On note BTP la classe constituée des instances binaires satisfaisant la propriété BTP.

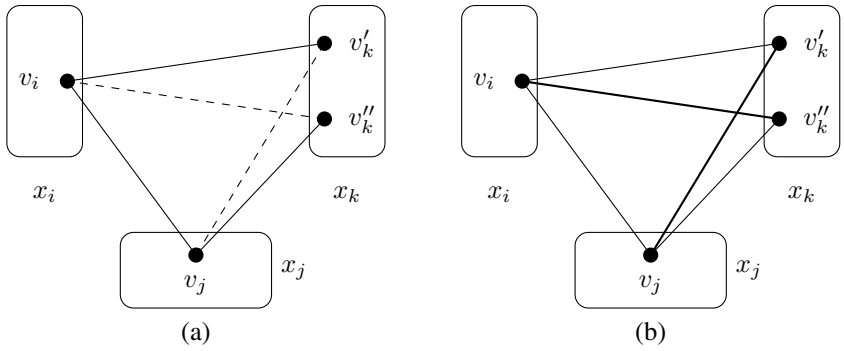


FIGURE 1.4 – (a) Un triangle cassé (v_i, v_j, v'_k, v''_k). (b) Un triangle non cassé (v_i, v_j, v'_k, v''_k) et satisfaisant donc la propriété BTP.

Cette définition peut se représenter graphiquement grâce à la microstructure de l'instance considérée comme indiqué dans la figure 1.4. Par la suite, l'absence d'arête et les arêtes en pointillés représenteront des tuples interdits.

La microstructure de la figure 1.4(a) correspond à une instance ne respectant pas la propriété BTP par rapport à l'ordre $x_i < x_j < x_k$ car les tuples (v_j, v'_k) et (v_i, v''_k) ne sont pas autorisés. Dans ce cas, (v_i, v_j, v'_k, v''_k) constitue un *triangle cassé* sur les valeurs v'_k et v''_k. La présence de ce triangle cassé conduit à dire qu'il existe un triangle cassé sur x_k par rapport à x_i et x_j. Par contre, si $(v_i, v''_k) \in R(c_{ik})$ ou $(v_j, v'_k) \in R(c_{jk})$, la propriété BTP est bien satisfaite comme l'illustre la microstructure représentée à la figure 1.4(b).

Un triangle cassé peut être vu comme la raison potentielle d'un retour en arrière lors d'une recherche énumérative. Ainsi, interdire tous les triangles cassés rencontrés selon un ordre donné sur les variables permet de garantir la polynomialité d'une résolution accomplie selon cet ordre. La cohérence d'arc est alors une procédure de décision (Cooper et al., 2008, 2010). De plus, une des propriétés remarquables de cette classe réside dans la capacité de l'algorithme MAC à résoudre ses instances en temps polynomial sans pour autant avoir connaissance de l'ordre sur les variables (Cooper et al., 2010). Ce résultat est en fait vrai pour tout algorithme (comme RFL) maintenant une cohérence de domaine au moins égale à la cohérence d'arc. Il est d'importance car, même si l'existence d'un ordre convenable sur les variables peut être établie en temps polynomial (Cooper et al., 2008, 2010), ne pas avoir à le calculer explicitement permet une exploitation implicite de cette classe par tout solveur maintenant en chaque nœud une cohérence de domaine au moins égale à la cohérence d'arc. D'autre part, cette classe généralise des classes polynomiales existantes comme les classes $TREE$ et RRM.

La classe BTP a fait l'objet de différentes généralisations ou extensions, aboutissant ainsi à de nouvelles classes polynomiales. Nous décrivons ci-dessous deux classes proposées par Naanaa (Naanaa, 2013, 2016). D'autres seront détaillées dans le chapitre 2.

Au préalable, nous rappelons la notion de famille indépendante. Étant donnés un ensemble fini E et une famille finie $\{E_i\}_{i \in I}$ de sous-ensembles de E, la famille $\{E_i\}_{i \in I}$ est dite *indépendante* si et seulement si pour tout $J \subsetneq I$, $\bigcap_{i \in I} E_i \subsetneq \bigcap_{j \in J} E_j$.

Définition 1.30 (Classe Rang Directionnel (Naanaa, 2013)) *Soit une instance CSP binaire P dont les variables sont totalement ordonnées par $<$. Le* rang directionnel *de la variable x_m est la taille k de la plus grande affectation cohérente $(a_1, \ldots, a_k)$ à un ensemble de variables $x_{i_1}, \ldots, x_{i_k}$ (avec $i_1 < \ldots < i_k < m$) telle que la famille des ensembles $\{R(c_{i_j m})[a_j]\}_{j=1,\ldots,k}$ est indépendante.*
Le rang directionnel *de P (par rapport à l'ordre $<$ de ses variables) est le rang directionnel maximal sur toutes ses variables.*
Étant donné un entier k, on note DR-k l'ensemble des instances ayant un rang directionnel au plus égal à k et satisfaisant la $(k+1)$-cohérence forte.

Naanaa a montré que toute instance binaire ayant un rang directionnel au plus égal à k et vérifiant la $(k+1)$-cohérence forte est globalement cohérente (Naanaa, 2013). De plus, cette famille de classes généralise plusieurs

classes existantes, notamment la classe BTP. En effet, les instances satisfaisant la propriété BTP ont un rang directionel au plus égal à un.

Plus récemment, Naanaa a considéré une autre alternative pour généraliser la classe BTP en autorisant la présence de certains triangles cassés. En contre-partie, la propriété WBTP garantit, pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) conservé, l'existence d'une valeur $v_k \in d_{x_k}$ compatible avec v_i et v_j. Il en résulte que, comme pour BTP, la cohérence d'arc est, une fois encore, une procédure de décision.

Définition 1.31 (Classe Weak-BTP (Naanaa, 2016)) *Une instance CSP binaire dotée d'un ordre $<$ sur ses variables satisfait la propriété WBTP (Weak Broken Triangle Property) si pour tout triplet de variables $x_i < x_j < x_k$ et pour tout $v_i \in d_{x_i}$, $v_j \in d_{x_j}$ tels que $(v_i, v_j) \in R(c_{ij})$, il existe une variable $x_\ell < x_k$ telle que lorsque $v_\ell \in d_{x_\ell}$ est compatible avec v_i et v_j, alors nous avons $\forall v_k \in d_{x_k}$, $(v_\ell, v_k) \in R(c_{\ell k}) \Rightarrow ((v_i, v_k) \in R(c_{ik}) \wedge (v_j, v_k) \in R(c_{jk}))$.*
On note $WBTP$ l'ensemble des instances satisfaisant la propriété WBTP.

Enfin, nous pouvons noter que le test d'appartenance aux classes DR-k (pour un entier k fixé) et $WBTP$ est réalisable en temps polynomial (Naanaa, 2013, 2016).

1.3.4 Autres classes polynomiales

Il existe d'autres voies pour garantir la polynomialité de la résolution. L'une consiste à autoriser ou à interdire un faible nombre de tuples pour chaque relation de sorte que chaque sous-problème possède un nombre polynomial de solutions. Par exemple, pour les instances *incrémentalement fonctionnelles* (Cohen et al., 2011), ce nombre de solutions est limité à un.

Définition 1.32 (Classe des instances incrémentalement fonctionnelle (Cohen et al., 2011)) *Une instance CSP P est dite* incrémentalement fonctionnelle *s'il existe un ordre $<$ sur les variables tel que pour $1 \leq i < n$, chaque solution de $P[\{x_1, \ldots, x_i\}]$ s'étend en au plus une solution de $P[\{x_1, \ldots, x_{i+1}\}]$ où, pour tout sous-ensemble X' de X, $P[X']$ représente l'instance CSP (X', D', C') avec $D' = \{d_{x_i} | x_i \in X'\}$ et $C' = \{c' | c \in C \text{ t.q. } S(c) \cap X' \neq \emptyset, S(c') = S(c) \cap X' \text{ et } R(c') = \{t[S(c')] | t \in R(c)\}\}$.*
Nous notons $IFUN$ l'ensemble de ces instances.

La résolution s'effectue sans retour arrière avec n'importe quel algorithme énumératif si ce dernier a connaissance de l'ordre. Notons que cet ordre est calculable en temps polynomial.

Une autre possibilité est d'exploiter un niveau de cohérence forte adapté en fonction de la taille du plus grand domaine et de l'arité maximale des contraintes de l'instance :

Théorème 1.2 ((Dechter, 1992)) *Soit une instance CSP P d'arité a ayant au maximum d valeurs par domaine. Si P est fortement $(d(a-1)+1)$-cohérente, alors P est globalement cohérente.*

Il est possible d'utiliser ce théorème pour définir des classes polynomiales. Toutefois, ce résultat n'est guère exploitable en pratique que dans le cas des instances binaires *monovalentes* ou *bivalentes* (c'est-à-dire ayant au plus une ou deux valeurs respectivement par domaine). En effet, au-delà, pour un nombre de valeurs supérieur ou pour une arité maximale plus grande, le coût de la cohérence forte devient prohibitif.

Définition 1.33 (Classe des instances binaires bivalentes) *La classe BV est définie par l'ensemble des instances binaires dont les variables ont au plus deux valeurs dans leur domaine.*

1.4 Conclusion

La résolution du problème CSP peut s'effectuer notamment en exploitant des classes polynomiales ou par le biais d'algorithmes énumératifs. Concernant les classes polynomiales, il s'agit principalement de travaux théoriques, même si de rares cas d'utilisations pratiques existent (par exemple (Lesaint et al., 1998; Purvis et Jeavons, 1999; Dincbas et al., 1988)). À l'opposé, les solveurs actuels basés sur des algorithmes énumératifs, dont la complexité temporelle est exponentielle dans le pire des cas, parviennent à faire preuve d'une remarquable efficacité

en pratique sans toutefois exploiter explicitement les classes polynomiales existantes. Cependant, certains algorithmes comme MAC ou RFL sont en mesure d'exploiter certaines classes polynomiales (Petke et Jeavons, 2009; Cooper et al., 2010).

Dans la suite de ce manuscrit, nous nous intéressons aux classes polynomiales hybrides, dans le chapitre 2, et aux classes polynomiales structurelles dans le chapitre 3. Dans les deux cas, notre objectif ne se limite pas seulement à définir ou étudier des classes polynomiales d'un point de vue théorique. Notre démarche consiste plutôt à tisser des liens entre la théorie (et ses classes polynomiales) et la pratique (avec ses solveurs à l'efficacité si remarquable). Par exemple, dans le cadre des classes polynomiales hybrides, nous proposons une nouvelle évaluation de la complexité des algorithmes comme FC ou RFL qui nous permet de définir de nouvelles classes polynomiales explicitement capturées par ces algorithmes. Nous poursuivons un objectif similaire en définissant la notion de classes polynomiales cachées notamment via un certain filtrage donné. Nous décrivons aussi une nouvelle manière d'exploiter des classes polynomiales à travers la fusion de valeurs. Sur le plan pratique, nous évaluons, par exemple, l'appartenance à des classes polynomiales de certaines instances couramment utilisées pour évaluer les solveurs et exploitons ces résultats pour fournir des explications à l'efficacité pratique des solveurs basés sur des algorithmes comme MAC ou RFL. Dans le cadre des classes polynomiales structurelles, les liens entre théorie et pratique sont encore plus évidents dans la mesure où la majeure partie des travaux porte sur la définition et l'étude d'algorithmes de résolution généraux exploitant des classes polynomiales structurelles (principalement la classe BTW_k). D'un point de vue théorique, nous présentons une nouvelle évaluation de la complexité des algorithmes comme nFC$_i$ ou RFL, relative cette fois à un paramètre structurel, ce qui nous permet ensuite de proposer une mise à jour de la hiérarchie des méthodes de décompositions structurelles Gottlob et al. (2000). Ensuite, nous avons étudié les méthodes de calculs de décompositions arborescentes et en avons proposé de nouvelles. Puis, les différents algorithmes de résolution ainsi que les différents méthodes de calculs de décompositions sont évalués expérimentalement afin de mettre en lumière leur intérêt pratique. Enfin, une partie de ces travaux sont ensuite étendus à des problèmes voisins comme le problème d'optimisation sous contraintes ou SAT.

Chapitre 2

Classes polynomiales hybrides pour le problème CSP

Sommaire

Dans ce chapitre, nous présentons les travaux réalisés concernant des classes polynomiales hybrides du problème CSP. Dans un premier temps, nous nous intéressons, dans la section 2.1, à des travaux portant directement sur la microstructure. D'une part, nous proposons une nouvelle évaluation de la complexité d'algorithmes énumératifs classiques comme Backtrack, Forward-Checking et RFL ainsi que de nouvelles classes polynomiales dans le cadre des instances binaires. D'autre part, nous définissons plusieurs généralisations de la microstructure pour les instances n-aires et nous étudions quelques possibilités offertes par ces généralisations concernant la définition de nouvelles classes polynomiales. Dans un second temps, dans la section 2.2, nous décrivons différents travaux autour de la classe BTP. En particulier, nous exploitons la classe BTP dans le cadre de la fusion de valeurs avant de présenter différentes extensions ou généralisations de cette classe. Enfin, dans la section 2.3, nous étudions les classes polynomiales d'un point de vue pratique. Nous définissons d'abord le concept de *classes polynomiales cachées*, puis nous dressons un bilan des observations faites sur des jeux de données utilisées habituellement par la communauté CSP.

Les travaux présentés dans ce chapitre sont principalement le fruit de collaborations avec Martin Cooper, Achref El Mouelhi, Philippe Jégou ou Bruno Zanuttini. Ces collaborations ont été initiées durant le projet ANR TUPLES (Tractability for Understanding and Pushing forward the Limits of Efficient Solvers) et se sont poursuivies à sa fin au printemps 2015.

2.1 Autour de la microstructure

Comme nous avons pu le constater dans le chapitre 1, de nombreuses classes polynomiales s'appuient sur la notion de microstructure. Dans cette section, nous nous proposons d'abord d'exploiter à nouveau cette notion,

mais dans un contexte a priori différent, en l'occurrence l'évaluation de la complexité d'algorithmes énumératifs classiques comme Backtrack, Forward-Checking et RFL. Fort de cette nouvelle évaluation, nous proposons ensuite de nouvelles classes polynomiales dont les instances peuvent être résolues en temps polynomial par ces mêmes algorithmes. Ensuite, compte tenu de la limitation de la microstructure aux instances binaires, nous présentons plusieurs extensions aux instances non binaires et étudions les possibilités qu'elles offrent en termes de définitions de nouvelles classes polynomiales.

2.1.1 Exploitation des cliques maximales

Nous proposons d'abord une nouvelle évaluation de la complexité en temps des algorithmes Backtrack, Forward-Checking et RFL. Cette évaluation s'appuie sur la correspondance existant entre les affectations cohérentes de k variables d'une instance CSP binaire et les cliques de taille k de sa microstructure (le théorème 1.1 étant une illustration de cette correspondance pour $k = n$). Plus précisément, nous avons établi que toute affectation cohérente construite par BT, FC ou RFL lors de la résolution d'une instance P appartient à une clique maximale de $\mu(P)$ et que chaque clique maximale de $\mu(P)$ est visitée au plus une fois durant la recherche. Il en découle que la taille de l'arbre de recherche exploré par BT, FC ou RFL est bornée par le nombre de cliques maximales dans la microstructure. À partir de ce résultat, il est possible d'exprimer d'une nouvelle manière la complexité de ces algorithmes :

Proposition 2.1 *Si $\omega_{\#}(\mu(P))$ représente le nombre de cliques maximales dans la microstructure $\mu(P)$, la complexité en temps pour résoudre une instance CSP binaire P est en :*

- $O(n^2 d \cdot \omega_{\#}(\mu(P)))$ *pour BT et FC,*
- $O(ned^2 \cdot \omega_{\#}(\mu(P)))$ *pour RFL.*

La complexité en temps de BT, FC et RFL est ainsi polynomiale dans le nombre $\omega_{\#}(\mu(P))$ de cliques maximales dans la microstructure de l'instance. Néanmoins, cela ne garantit en rien une résolution efficace pour toute instance CSP binaire car, généralement, la microstucture possède un nombre exponentiel de cliques maximales. Ce résultat peut être étendu à tout algorithme maintenant une cohérence de domaine (moyennant l'intégration du coût du filtrage associé) et exploitant une stratégie de branchement non binaire. Naturellement, la question d'une extension de ce résultat à un algorithme exploitant une stratégie de branchement binaire se pose. Malheureusement, le raisonnement exploité pour une stratégie de branchement non binaire n'est pas directement transposable à une stratégie binaire. Aussi, à l'heure actuelle, cette question reste ouverte, mais nous conjecturons que la réponse est positive :

Conjecture 2.1 *La complexité en temps de MAC est polynomiale en fonction du nombre de cliques maximales de la microstructure de l'instance binaire considérée.*

Bien que le nombre de cliques maximales dans un graphe soit généralement exponentiel, il existe des classes de graphes pour lesquelles ce nombre est polynomial comme par exemple :

- les graphes dépourvus de cycle de longueur trois,
- les graphes bipartis,
- les graphes planaires (Wood, 2007), toroïdaux (Dujmović et al., 2011) et plus généralement plongeables dans une surface (Dujmović et al., 2011),
- les graphes CSG^k (Chmeiss et Jégou, 1997).

S'il est possible de définir de nouvelles classes polynomiales hybrides pour le problème CSP en utilisant ces différentes classes de graphes, seule la classe des graphes CSG^k semble pertinente. En effet, les autres classes de graphes conduisent à des classes polynomiales pour CSP triviales et a priori dépourvues de tout intérêt pratique ou théorique. La classe des graphes CSG^k (pour Chordal Sub-Graph (Chmeiss et Jégou, 1997)) est définie inductivement comme suit :

- CSG^0 est la classe des graphes complets.

- Étant donné $k > 0$, CSG^k est la classe des graphes $G = (V, E)$ tels qu'il existe un ordre $\sigma = (v_1, ..., v_{|V|})$ sur V vérifiant que pour $i = 1, \ldots, |V|$, le graphe $G(N^+(v_i))$ est un graphe CSG^{k-1}, où $N^+(v_i)$ représente le *voisinage ultérieur* de v_i, c'est-à-dire, $N^+(v_i) = \{v_j \in V | \{v_i, v_j\} \in E, i < j\}$ et, pour tout ensemble V' de V, $G(V')$ est le sous-graphe *induit* par E sur V', soit, $G(V') = (V', E')$ où $E' = \{\{x, y\} \mid x, y \in V' \text{ et } \{x, y\} \in E\}$.

À partir de cette classe de graphe, nous pouvons définir une nouvelle famille de classes polynomiales d'instances CSP binaires :

Définition 2.1 (Classes des instances ayant une microstructure CSG^k)
Étant donné un entier k, la classe CSG^k est définie par l'ensemble des instances CSP binaires ayant une microstructure CSG^k.

La classe de graphes CSG généralise la classe des graphes complets (graphes CSG^0) et surtout la classe des graphes chordaux (graphes CSG^1). Il s'ensuit donc que $CSG^1 = CM$. Comme les graphes chordaux, les graphes CSG^k possèdent des propriétés intéressantes. En particulier, pour un entier k donné, les graphes CSG^k possèdent au plus $|V|^k$ cliques maximales. De plus, ils peuvent être reconnus en temps polynomial. Cependant, cette propriété ici n'est pas fondamentale. En effet, BT, FC et RFL sont capables de résoudre en temps polynomial les instances des classes CSG^k sans pour autant avoir à mettre en œuvre un quelconque traitement spécifique, ni avoir besoin de reconnaître la nature de l'instance. Cette exploitation implicite des classes CSG^k par des algorithmes classiques constitue ainsi un des intérêts majeurs de ce travail (El Mouelhi et al., 2012a, 2013d).

2.1.2 Extensions de la microstructure au cas des instances CSP n-aires

La notion de microstructure joue un rôle important dans la définition de nombreuses classes polynomiales. Malheureusement, sa définition concerne principalement les instances binaires. Une possibilité pour généraliser cette notion passe naturellement par l'emploi d'hypergraphes. Cette approche a été retenue par Cohen (Cohen, 2003). Toutefois, Cohen ne définit pas directement l'hypergraphe représentant la microstructure mais celui correspondant à son complémentaire. De plus, dans les faits, il ne l'utilise que dans le cas des instances binaires pour définir la classe CCM. Ceci n'est pas surprenant car généraliser les notions sur les graphes aux hypergraphes n'est pas toujours une tâche aisée. En effet, beaucoup de notions simples à exprimer au niveau des graphes connaissent plusieurs généralisations qui sont souvent plus complexes à manipuler. L'exemple typique est celui de la notion d'acyclicité qui est unique pour les graphes alors que plusieurs généralisations différentes existent pour les hypergraphes (comme, par exemple l'α-acyclicité et la β-acyclicité). De plus, la théorie des graphes est relativement plus riche que celle des hypergraphes. Aussi, nous avons préféré considérer des graphes pour les différentes modélisations de la microstructure des instances non binaires que nous proposons (El Mouelhi et al., 2013a, 2014a). Pour pouvoir continuer à travailler avec des graphes, nous nous sommes inspirés des différentes transformations polynomiales existantes (appelées aussi *codages* dans la communauté CSP) permettant de représenter une instance CSP quelconque sous la forme d'une instance binaire. Ces transformations garantissent bien entendu l'existence d'une bijection entre les ensembles de solutions de l'instance CSP de départ et de celle obtenue après transformation.

Le *codage dual*, dans le domaine de la Programmation par Contraintes, a été employé pour la première fois dans (Dechter et Pearl, 1987). En théorie des (hyper)graphes, il est appelé *line graph* et repose sur la transformation d'un hypergraphe en graphe. Dans la communauté CSP, il est également appelé *graphe dual* ou *intergraphe* dans (Jégou, 1991). Cette représentation avait aussi précédemment été employée dans la théorie des bases de données relationnelles sous le vocable de *qual graphs* (Bernstein et Goodman, 1981). Dans ce codage, les variables, dites *variables duales*, correspondent aux contraintes de l'instance originelle tandis qu'il existe une contrainte binaire reliant deux variables duales si elles partagent au moins une variable originelle (c'est-à-dire si l'intersection des portées des contraintes originelles correspondantes n'est pas vide). La définition de la microstructure associée, appelée *DR-microstructure* correspond ainsi à la microstructure de l'instance duale :

Définition 2.2 (DR-microstructure) *Étant donnée une instance CSP $P = (X, D, C)$, la* DR-microstructure *est le graphe non orienté $\mu_{DR}(P) = (V, E)$ avec :*

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

Il est bien connu que, dans le graphe dual, certaines arêtes redondantes peuvent être éliminées sans remettre en cause l'équivalence avec l'instance initiale (Janssen et al., 1989; Jégou, 1991). Sur cette base, nous pouvons ainsi définir un ensemble de microstructures différentes, voisines de la DR-microstructure. Dans (Jégou, 1991), il est montré que pour un CSP d'arité quelconque, il existe un ensemble de CSP binaires équivalents construits sur la base de l'ensemble des intergraphes, le maximal d'entre eux correspondant au codage dual. En considérant cet ensemble de graphes partiels, nous pouvons étendre la définition précédente de DR-microstructure :

Définition 2.3 (DSR-microstructure) *Étant donnés une instance CSP $P = (X, D, C)$ et un de ses intergraphes (C, F), la* DSR-microstructure *est le 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*
  - $E_1 = \{\ \{(c_i, t_i), (c_j, t_j)\}\ \mid\ \{c_i, c_j\} \in F, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]\}$
  - $E_2 = \{\ \{(c_i, t_i), (c_j, t_j)\}\ \mid\ \{c_i, c_j\} \notin F\}$.

Le second codage, dit *codage par variables cachées* (*Hidden Transformation en anglais*), 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 duales issues de C. Les nouvelles contraintes binaires vont relier une variable duale à une variable originelle si la variable originelle appartient à la portée de la contrainte originelle relative à la variable duale. La HT-microstructure sera donc basée sur cette représentation binaire.

Définition 2.4 (HT-microstructure) *Étant donnée une instance CSP $P = (X, D, C)$, la* HT-microstructure *de P est le graphe non orienté $\mu_{HT}(P) = (V, E)$ avec :*

- $V = S_1 \cup S_2$ *tel 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 $x_j \notin S(c_i)$*$\}$.

Enfin, le codage mixte (Stergiou et Walsh, 1999) d'un CSP non binaire, combine à la fois le codage dual et le codage par variables cachées. Cette approche consiste à connecter les valeurs des variables duales aux valeurs des variables originelles, les valeurs des variables originelles étant connectées entre elles si elles n'appartiennent pas à un même domaine et les tuples entre eux s'ils sont compatibles :

Définition 2.5 (ME-microstructure) *Étant donnée une instance CSP $P = (X, D, C)$, la* ME-microstructure *de P est le graphe non orienté $\mu_{ME}(P) = (V, E)$ avec :*

- $V = S_1 \cup S_2$ *tel que*
  - $S_1 = \{(c_i, t_i) : c_i \in C, t_i \in R(c_i)\}$,
  - $S_2 = \{(x_j, v_j) : x_j \in X, v_j \in d_{x_j}\}$,
- $E = E_1 \cup E_2 \cup E_3$ *tel 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 $x_j \notin S(c_i)$*$\}$
  - $E_3 = \{\ \{(x_i, v_i),\ (x_j, v_j)\} \mid\ x_i \neq x_j\}$.

Comme pour la microstructure dans le cas binaire, nous pouvons retrouver, pour chacune de ces microstructures, l'équivalence entre les solutions de l'instance et certaines cliques ou bicliques :

Théorème 2.1 *Étant donnée une instance CSP P, les conditions suivantes sont équivalentes :*

- *P possède une solution,*
- *$\mu_{DR}(P)$ a une clique de taille e,*
- *$\mu_{DSR}(P)$ a une clique de taille e,*

- $\mu_{HT}(P)$ *possède 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,*
- $\mu_{ME}(P)$ *possède une clique de taille* $n + e$.

Ces différentes microstructures peuvent être ensuite exploitées pour étendre certains résultats établis dans le cas binaire. Elles conduisent dans certains cas à la définition de nouvelles classes polynomiales. Par exemple, nous avons pu mettre en évidence que la complexité en temps de BT, nFC$_i$ (pour $i = 2, \ldots, 5$) et RFL est polynomiale dans le nombre de cliques maximales de la DR-microstructure sous réserve d'exploiter un ordre sur les variables spécifique (El Mouelhi et al., 2013d). Ainsi, il est possible de proposer l'équivalent de la classe CSG^k pour les instances n-aires en considérant les instances ayant une DR-microstructure CSG^k. Nous avons proposé d'autres exemples d'utilisation en considérant la classe ZOA (El Mouelhi et al., 2013a, 2014a) ou la classe BTP que nous évoquons dans la section suivante.

2.2 Autour de la classe BTP

La classe BTP (Cooper et al., 2008, 2010) a ouvert une nouvelle voie de recherche pour les classes polynomiales hybrides à différents titres. D'une part, il s'agit d'une classe qui peut être facilement (et implicitement) exploitée en pratique, dans la mesure où tout algorithme maintenant la cohérence d'arc en chaque nœud est capable de résoudre ses instances en temps polynomial sans avoir connaissance d'un ordre convenable sur les variables, Autrement dit, la grande majorité des solveurs existants sont capables d'exploiter implicitement cette classe. D'autre part, elle a été la source d'inspiration de nombreux travaux (comme par exemple (Naanaa, 2013, 2016)) permettant d'identifier de nouvelles classes polynomiales. Enfin, elle peut être exploitée au-delà de la simple résolution. Par exemple, dans (Cooper et al., 2010), Cooper et al. proposent un schéma d'élimination de variables exploitant une version locale de la propriété BTP.

Les travaux présentés ci-dessous s'inscrivent dans la continuité. Dans un premier temps, nous décrivons comment exploiter la propriété BTP pour fusionner des valeurs tout en préservant la satisfiabilité. Ensuite, nous proposons plusieurs classes polynomiales généralisant ou étendant la classe BTP.

2.2.1 Exploitation pour la fusion de valeurs

La *fusion* de deux valeurs v'_k, v''_k issues du domaine d_{x_k} d'une instance CSP binaire consiste à remplacer v'_k et v''_k dans d_{x_k} par une nouvelle valeur v_k qui est compatible avec toutes les valeurs des autres variables compatibles avec v'_k ou v''_k. Une *condition de fusion de valeurs* pour les valeurs $v'_k, v''_k \in d_{x_k}$ d'une instance P est une propriété $Q(x_k, v'_k, v''_k)$ calculable en temps polynomial telle que, si la propriété est vraie, alors l'instance P' obtenue à partir de P en fusionnant $v'_k, v''_k \in d_{x_k}$ est satisfiable si et seulement si P est satisfiable.

Une exploitation locale des triangles cassés de la propriété BTP, appelée *BTP-fusion*, permet de définir une condition de fusion de valeurs (Cooper et al., 2014, 2016a) :

Proposition 2.2 *L'absence de triangle cassé sur* v'_k *et* v''_k *est une condition de fusion de valeurs.*

Autrement dit, si deux valeurs v'_k et v''_k d'une variable x_k sont telles que quelles que soient les variables x_i et x_j (avec $x_i \neq x_k$ et $x_j \neq x_k$), pour toutes valeurs $v_i \in d_{x_i}$ et $v_j \in d_{x_j}$, il n'existe pas de triangle cassé (v_i, v_j, v'_k, v''_k), alors les valeurs v'_k et v''_k peuvent être remplacées dans d_{x_k} par une valeur v_k qui sera compatible avec toute valeur compatible avec v'_k ou v''_k. Par exemple, dans la microstructure représentée dans la figure 2.1(a), les valeurs 1 et 3 de x_4 sont fusionnables par BTP-fusion. Leur fusion en une nouvelle valeur 13 aboutit à la microstructure représentée dans la figure 2.1(b).

Cette opération de fusion ne remet pas en cause la satisfiabilité de l'instance. De plus, il est possible de reconstruire en temps polynomial toutes les solutions de P à partir des solutions de toute instance P^f obtenue en appliquant à P une suite de fusions. À noter que la reconstruction d'une solution de P à partir d'une solution de P^f peut s'effectuer en temps linéaire dans la taille de l'instance P.

Par ailleurs, cette règle de fusion généralise des propriétés bien connues de l'état de l'art :

Théorème 2.2 *La BTP-fusion généralise la substitution de voisinage (Freuder, 1991) et l'interchangeabilité virtuelle (Likitvivatanavong et Yap, 2013).*

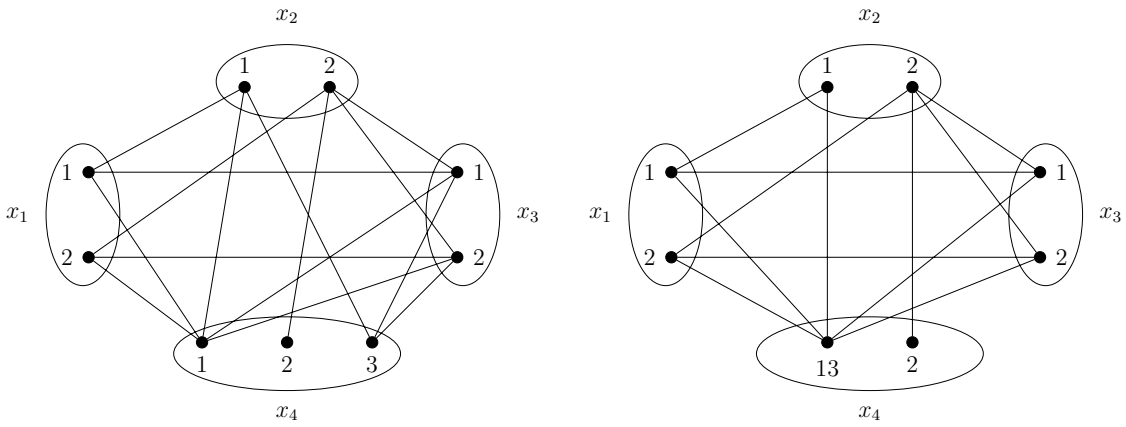


FIGURE 2.1 – Une microstructure dans laquelle les valeurs 1 et 3 de x_4 sont fusionnables par BTP-fusion (a) et la microstructure résultant de leur fusion en une nouvelle valeur 13 (b).

Cette règle de fusion a été généralisée aux instances n-aires (Cooper et al., 2014, 2016a). Toutefois, elle ne semble utilisable, d'un point de vue pratique, que pour les instances binaires.

La BTP-fusion repose sur l'absence de triangle cassé. Cependant, cette condition est plus forte que nécessaire et peut être partiellement relâchée en tolérant certains triangles cassés, à savoir les triangles légèrement cassés. Un triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in d_{x_i}$, $v_j \in d_{x_j}$ et $v'_k, v''_k \in d_{x_k}$ est dit *légèrement cassé* s'il existe au moins une variable $x_\ell \in X \setminus \{x_i, x_j, x_k\}$ telle que $\forall\ v_\ell \in d_{x_\ell}$, si $(v_i, v_\ell) \in R(c_{i\ell})$ et $(v_j, v_\ell) \in R(c_{j\ell})$ alors $(v'_k, v_\ell) \notin R(c_{k\ell})$ et $(v''_k, v_\ell) \notin R(c_{k\ell})$. La variable x_ℓ *soutient* alors le triangle légèrement cassé (v_i, v_j, v'_k, v''_k).

Cette notion a été inspirée par le récent travail de Naanaa au sujet de la classe WBTP (Naanaa, 2016). Nous l'exploitons maintenant pour introduire la propriété m-wBTP (Cooper et al., 2016b) :

Définition 2.6 *Un couple de valeurs $v'_k, v''_k \in d_{x_k}$ satisfait m-wBTP pour une constante $m \leq n-3$ si pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) avec $v_i \in d_{x_i}$ et $v_j \in d_{x_j}$, il existe un ensemble de $r \leq m$ variables de soutien $\{x_{\ell_1}, \ldots, x_{\ell_r}\} \subseteq X \setminus \{x_i, x_j, x_k\}$ tel que pour tout $(v_{\ell_1}, \ldots, v_{\ell_r}) \in d_{x_{\ell_1}} \times \ldots \times d_{x_{\ell_r}}$, si $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i, v_j)$ est une solution partielle, alors il existe $\alpha \in \{1, \ldots, r\}$ tel que $(v_{\ell_\alpha}, v'_k), (v_{\ell_\alpha}, v''_k) \notin R(c_{\ell_\alpha k})$.*
x_{ℓ_α} est alors dite variable bouclier *pour cette solution partielle.*

L'idée sous-jacente de cette propriété est que chaque triangle cassé sur un couple de valeur $v'_k, v''_k \in d_{x_k}$ soit soutenu par au moins une variable. Le paramètre m définit alors le nombre maximum de variables différentes pouvant soutenir des triangles légèrement cassés sur un couple de valeurs donné. L'exploitation de la propriété m-wBTP nous permet de définir une nouvelle opération de fusion appelée m-wBTP-fusion.

Proposition 2.3 *Étant donnés une instance CSP binaire et un entier m ($m \leq n-3$), fusionner deux valeurs $v'_k, v''_k \in d_{x_k}$ qui satisfont m-wBTP est une condition de fusion de valeurs.*

Comme pour la BTP-fusion, la m-wBTP-fusion préserve la satisfiabilité de l'instance et il est possible de calculer en temps polynomial les solutions d'une instance P à partir des solutions de toute instance P^f obtenue en appliquant à P une suite de m-wBTP-fusions.

La 0-wBTP-fusion correspond à la BTP-fusion puisqu'elle s'appuie sur zéro variable de soutien. La proposition établit le lien existant entre les différentes formes de wBTP-fusion :

Propriété 2.1 *Étant donnée une instance CSP binaire, si un couple de valeurs $v'_k, v''_k \in d_{x_k}$ satisfait m-wBTP alors il satisfait $(m+1)$-wBTP (pour $0 \leq m \leq n-4$).*

Il en découle le corollaire suivant :

Corollaire 2.1 *La m-wBTP-fusion généralise la BTP-fusion, et donc la substitution de voisinage et l'interchangeabilité virtuelle.*

Au-delà, nous avons pu montrer que la $(n-3)$-BTP-fusion est une condition de fusion maximale dans le sens où la fusion de n'importe quel couple de valeurs ne respectant pas la propriété entraîne nécessairement la modification de la satisfiabilité de l'instance.

Au final, la BTP-fusion et la m-BTP-fusion peuvent être exploitées en pré-traitement pour réduire la taille des domaines des variables avant de commencer la résolution. Contrairement aux cohérences de domaine qui n'entraînent que des suppressions de valeurs ne pouvant pas participer à une solution, les valeurs supprimées par fusion peuvent participer ou non à des solutions. Ainsi, rien ne s'oppose à exploiter conjointement cette règle de fusion et une cohérence de domaine comme la cohérence d'arc. Une autre différence notable par rapport aux cohérences de domaine concerne l'ordre dans lequel les fusions sont opérées. Si pour les cohérences de domaine, l'ordre dans lequel les valeurs sont supprimées n'a pas d'incidence sur le nombre ou l'identité des valeurs supprimées au final, ce n'est pas le cas pour la BTP-fusion ou la m-wBTP-fusion. En effet, la fusion de deux valeurs peut supprimer des triangles cassés mais aussi en engendrer de nouveaux. Malheureusement, Cooper et al. (Cooper et al., 2015a, 2016a) ont montré que maximiser le nombre de BTP-fusions (et donc de m-wBTP-fusions) constitue un problème NP-difficile. Cependant, des méthodes heuristiques restent envisageables pour mettre en œuvre ces opérations de fusion comme nous le montrons dans la sous-section 2.3.2.3 qui présente quelques résultats expérimentaux.

2.2.2 Extensions de la classe BTP

Nous présentons maintenant différentes généralisations ou extensions de la classe BTP (El Mouelhi et al., 2013b, 2015a; Jégou et Terrioux, 2015; Cooper et al., 2015c, 2016b). Certaines reposent sur l'observation que la propriété BTP impose des restrictions plus fortes que nécessaires. En effet, certains triangles cassés peuvent être tolérés tout en garantissant encore la polynomialité de la résolution. Une autre possibilité consiste en une exploitation de la propriété BTP sur la DR-microstructure, permettant ainsi de l'étendre aux instances non binaires.

Il existe différentes manières de tolérer la présence des triangles cassés. La première est basée sur la propriété *Extendable-Triple Property* (ETP (Jégou et Terrioux, 2015)).

Définition 2.7 (ETP) *Une instance CSP binaire $P = (X, D, C)$ satisfait la propriété* ETP *par rapport à un ordre sur les variables $<$ si et seulement si, pour tout quadruplet de variables (x_i, x_j, x_k, x_ℓ) tel que $x_i < x_j < x_k < x_\ell$, il existe au plus un triplet de variables cassé sur x_ℓ parmi (x_i, x_j, x_ℓ), (x_i, x_k, x_ℓ) et (x_j, x_k, x_ℓ).*
Un triplet de variables (x_i, x_j, x_k) (avec $x_i \neq x_j$, $x_i \neq x_k$ et $x_j \neq x_k$) est qualifié de cassé sur x_k par rapport à x_i et x_j *s'il existe au moins un triangle cassé sur x_k par rapport à x_i et x_j.*

Malheureusement, il s'avère que la classe constituée des instances satisfaisant la propriété ETP n'est pas une classe polynomiale. En effet, nous avons établi dans (Cooper et al., 2015c) que décider si une instance satisfaisant ETP possède une solution est un problème NP-complet. Pour pouvoir définir une nouvelle classe polynomiale tout en autorisant certains triangles cassés, nous devons considérer, en plus de la propriété ETP, la chemin-cohérence forte :

Théorème 2.3 *Une instance binaire P satisfaisant la chemin-cohérence forte et la propriété ETP par rapport à un ordre $<$ sur les variables possède une solution qui peut être trouvée en temps polynomial.*

Nous notons $ETP\text{-}SPC$ l'ensemble des instances binaires satisfaisant simultanément la chemin-cohérence forte et la propriété ETP. Pour aller toujours plus loin dans la recherche de classes polynomiales, une idée naturelle, mais erronée, est de considérer les instances satisfaisant ETP et les rendre chemin-cohérentes fortes par filtrage. En effet, cette alternative n'en est pas une car l'application du filtrage par chemin-cohérence forte est susceptible de créer de nouveaux triangles cassés et donc de détruire des propriétés comme ETP (ou BTP). Cependant, il est tout de même possible de généraliser encore cette classe en considérant la propriété k-BTP (Cooper et al., 2015c) définie comme suit :

Définition 2.8 (k-BTP) *Une instance CSP binaire P satisfait la propriété k-BTP pour un entier k donné $(2 \leq k < n)$ par rapport à un ordre $<$ sur les variables si et seulement si, pour tout sous-ensemble de variables $x_{i_1}, x_{i_2}, \ldots, x_{i_{k+1}}$ tel que $x_{i_1} < x_{i_2} < \ldots < x_{i_{k+1}}$, il existe au moins un couple de variables $(x_{i_j}, x_{i_{j'}})$ avec $1 \leq j < j' \leq k$ tel qu'il n'existe pas de triangle cassé sur x_{k+1} par rapport à x_{i_j} et $x_{i_{j'}}$.*

Cette propriété autorise davantage de triangles cassés et ainsi généralise les propriétés BTP et ETP. Plus précisément, nous pouvons constater que 2-BTP correspond exactement à la propriété BTP alors que 3-BTP inclut ETP. De plus, pour tout entier k compris entre 2 et $n-1$, les instances satisfaisant k-BTP satisfont aussi $(k+1)$-BTP. Il en résulte que pour k supérieur strictement à 2, décider si une instance satisfaisant k-BTP possède une solution est un problème NP-complet. Une nouvelle fois, pour pouvoir définir une classe polynomiale, il nous faut compenser la présence de triangles cassés par l'emploi d'une cohérence plus forte :

Théorème 2.4 *Une instance CSP binaire P satisfaisant k-BTP par rapport à un ordre sur les variables $<$ pour une constante k donnée (avec $2 \leq k < n$) et la k-cohérence forte possède une solution qui peut être trouvée en temps polynomial.*

Pour un entier k donné (avec $2 \leq k < n$), nous notons k-BTP-sC l'ensemble des instances binaires satisfaisant simultanément la k-cohérence forte et la propriété k-BTP. Plus récemment, nous avons exploré une autre voie avec toujours le même objectif d'autoriser des triangles cassés. Cette voie repose sur l'exploitation de la propriété m-wBTP (Cooper et al., 2016b) :

Définition 2.9 *Étant donné un entier $m \leq n-3$, une instance binaire P satisfait la propriété m-wBTP par rapport à un ordre $<$ sur les variables si pour toute variable x_k, chaque paire de valeurs de d_{x_k} satisfait m-wBTP dans la sous-instance de P restreinte aux variables x_i précédant x_k selon l'ordre $<$.*

Bien entendu, pour $m = 0$, la propriété correspond à BTP alors que pour $m = 1$ elle est plus générale que la propriété WBTP introduite par Naanaa (Naanaa, 2016). Par contre, elle s'avère être différente des propriétés ETP et k-BTP. Comme précédemment, la présence de triangles cassés nécessite de recourir à une cohérence généralement plus forte que la cohérence d'arc. Cependant, à la différence de k-BTP, le niveau de cohérence requis ne dépend pas directement de la valeur choisie pour la constante m, mais repose sur les interactions entre les variables intervenant dans les triangles légèrement cassés. Ces interactions sont mesurées à travers la notion de BT-largeur que nous introduisons maintenant :

Définition 2.10 (BT-largeur) *Soit une instance binaire P satisfaisant la propriété m-wBTP (pour un entier m donné) selon l'ordre $<$ sur les variables.*

- *L'*ensemble des BT-variables B_k *de x_k est l'ensemble des variables x_i telles que $x_i < x_k$ et qu'il existe un triangle cassé sur x_k relatif à x_i (et une autre variable x_j avec $x_j < x_k$).*
- *Un* ensemble bouclier S_k *de x_k est un ensemble de variables x_ℓ telles que $x_\ell < x_k$ et que, pour chaque triangle cassé (v_i, v_j, v'_k, v''_k) sur x_k relatif aux variables $x_i, x_j < x_k$, chaque solution partielle $(v_{\ell_1}, \ldots, v_{\ell_r}, v_i, v_j)$ des variables le soutenant possède une variable bouclier $x_{\ell_\alpha} \in S_k$.*
- *La* BT-largeur *de x_k est la plus petite valeur $|B_k \cap S_k|$ parmi tous les ensembles boucliers S_k de x_k.*
- *La* BT-largeur *de P est la BT-largeur maximale de ses variables.*

Les instances satisfaisant la propriété WBTP ont ainsi une BT-largeur de un et la cohérence d'arc suffit pour garantir la polynomialité de la résolution, comme établi dans (Naanaa, 2016). Plus généralement, nous avons :

Théorème 2.5 *Si une instance binaire P a une BT-largeur b et satisfait m-wBTP et la $\max(2, b+1)$-cohérence forte, alors P possède une solution.*

Nous pouvons alors définir la classe polynomiale $BBTW_b$ regroupant les instances binaires ayant une BT-largeur au plus égale à b et satisfaisant la $\max(2, b+1)$-cohérence forte et m-wBTP pour une certaine valeur de m.

Concernant la reconnaissance de ces classes polynomiales, il est possible de déterminer en temps polynomial s'il existe un ordre sur les variables satisfaisant la propriété pour les classes ETP et k-BTP (et bien sûr aussi le niveau de cohérence requis). Par contre, déterminer l'existence d'un ordre sur les variables de sorte que l'instance ait une BT-largeur b, pour un b fixé, est, à l'heure actuelle, un problème ouvert, même pour $b = 1$.

À présent, nous proposons d'étendre la propriété BTP aux instances non binaires (El Mouelhi et al., 2015a, 2013b), à travers la classe polynomiale suivante :

Définition 2.11 (Classe DBTP) *Une instance 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*

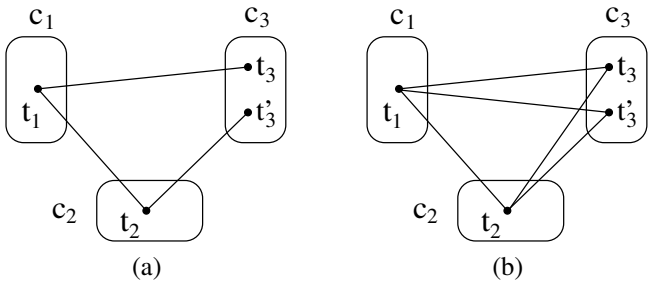


FIGURE 2.2 – Illustration de la propriété DBTP sur trois contraintes c_1, c_2 et c_3.

- $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$
- $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$
- $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$

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 instances.*

Cette définition est équivalente à satisfaire la propriété BTP sur l'instance duale (Dechter et Pearl, 1987) de P. En d'autres mots, cela revient à exploiter la propriété BTP sur la DR-microstructure de l'instance initiale. Par exemple, la figure 2.2 présente la DR-microstructure d'une instance P concernant trois contraintes. Dans la figure 2.2(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 2.2(b), si soit t_1 et t'_3, soit t_2 et t_3 sont compatibles, alors P vérifie DBTP relativement à l'ordre $\prec$.

Nous pouvons noter que la classe $DBTP$ est différente de la classe BTP. En particulier, une instance binaire peut vérifier DBTP tout en ne vérifiant pas BTP, et vice versa. Ceci n'a rien de surprenant car, même si l'instance originale et son instance duale représentent le même problème, leurs structures et microstructures sont très différentes. Cependant, la classe $DBTP$ possède des propriétés similaires à la classe BTP. Par exemple, elle est conservative pour tout filtrage qui se limite à supprimer des valeurs dans les domaines ou des tuples dans les relations. Une classe $\mathcal{C}$ d'instances CSP est dite *conservative* par rapport à un filtrage de cohérence ϕ si elle est fermée pour ϕ, c'est-à-dire, si l'instance obtenue après l'application de ϕ à toute instance de $\mathcal{C}$ appartient à la classe $\mathcal{C}$. De plus, toute instance de la classe $DBTP$ peut être résolue en temps polynomial par tout algorithme maintenant l'intercohérence en chaque nœud sans nécessité pour l'algorithme d'avoir connaissance de l'ordre. Sous certaines conditions, ce résultat peut être étendu aux algorithmes maintenant la cohérence d'arc comme MAC ou RFL. C'est notamment le cas pour toutes les instances binaires satisfaisant la propriété DBTP.

Enfin, au cours de nos différents travaux, nous avons établi les liens existant entre les classes que nous avons définies et certaines classes polynomiales de la littérature. Étant données deux classes polynomiales $\mathcal{C}_1$ et $\mathcal{C}_2$, deux relations sont possibles :

- $\mathcal{C}_1$ généralise $\mathcal{C}_2$ si $\mathcal{C}_1 \supsetneq \mathcal{C}_2$,
- $\mathcal{C}_1$ et $\mathcal{C}_2$ sont incomparables si $\mathcal{C}_1 \not\subseteq \mathcal{C}_2$ et $\mathcal{C}_1 \not\supseteq \mathcal{C}_2$.

Dans la figure 2.3 qui présente une partie de ces liens, $\mathcal{C}_1$ généralise $\mathcal{C}_2$ s'il existe un arc allant de $\mathcal{C}_1$ vers $\mathcal{C}_2$ tandis qu'une arête en pointillés représente l'incomparabilité de $\mathcal{C}_1$ et $\mathcal{C}_2$. La relation de généralisation étant transitive, seuls les principaux liens sont mis en avant.

2.3 Les classes polynomiales d'un point de vue pratique

Dans cette section, nous nous intéressons aux classes polynomiales d'un point de vue pratique. Dans un premier temps, nous introduisons la notion de classes polynomiales cachées dont l'objectif est de mettre en lumière

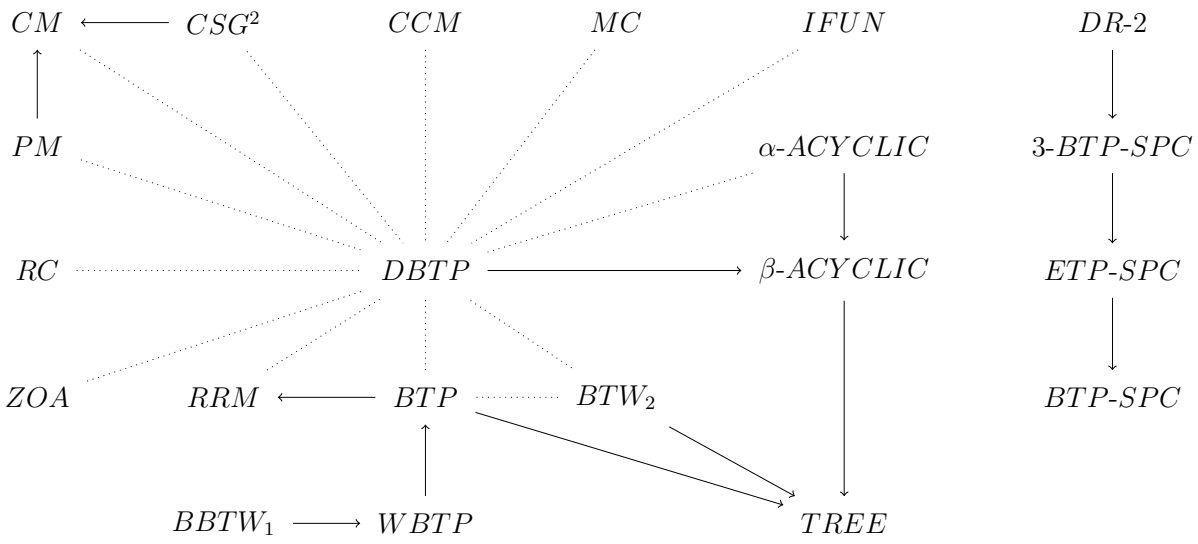


FIGURE 2.3 – Quelques relations entre certaines classes polynomiales.

des instances qui, initialement n'appartiennent pas à des classes polynomiales, mais qui via une transformation peuvent y appartenir. Ensuite, nous testons l'appartenance des instances habituellement employées pour évaluer et comparer les solveurs et nous mesurons l'intérêt pratique de la BTP-fusion et de la 1-wBTP-fusion.

2.3.1 Classes polynomiales cachées

L'idée sous-jacente dans le travail (El Mouelhi et al., 2014b,c) que nous présentons ici est que certaines instances n'appartenant pas à des classes polynomiales pourraient être transformées en instances figurant dans des classes polynomiales bien connues. Différentes transformations d'instances CSP ont été définies dans la littérature. Nous pouvons notamment citer la suppression de variables, de valeurs, l'ajout de contraintes ou la suppression de tuples dans les relations de compatibilité. Les filtrages associés aux cohérences locales sont des exemples typiques de tranformations possibles. Formellement, nous considérons la notion de transformation suivante :

Définition 2.12 *Étant donnée une instance de 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* $t_{dom}(d_x) \subseteq d_x\}$
- $\forall c \in C, t_{cons}(c)$ *vérifie :*
  - $t_{cons}(S(c)) = S(c) \backslash \{x \in X : x \notin t_{var}(X)\}$ *et*
  - $t_{cons}(R(c)) \subseteq R(c)[t_{cons}(S(c))]$.
- $\forall c' \in t_{cons}(C)$ *telle que* $c' \notin C$, $t_{cons}(R(c')) \subseteq \prod_{x' \in S(c')} t_{dom}(d_{x'})$.

Étant données une transformation t et une classe d'instances $\mathcal{C}$, nous pouvons définir l'image de la classe $\mathcal{C}$ par t, à savoir $t(\mathcal{C}) = \{t(P) : P \in \mathcal{C}\}$. La classe d'instances *mise en évidence par t pour $\mathcal{C}$* est $\mathcal{C}^t = \{P : t(P) \in \mathcal{C}\}$. Nous pouvons maintenant définir la notion de classe polynomiale cachée :

Définition 2.13 (Classe polynomiale cachée) *La classe $\mathcal{P}$ est dite* classe cachée *de la classe $\mathcal{C}$ pour la transformation t, si $\mathcal{P} \subseteq \mathcal{C}^t$, c'est-à-dire si $t(\mathcal{P}) \subseteq \mathcal{C}$.*
$\mathcal{P}$ est appelée classe polynomiale cachée de $\mathcal{C}$ pour t *si :*

- *$\mathcal{C}$ est une classe polynomiale,*

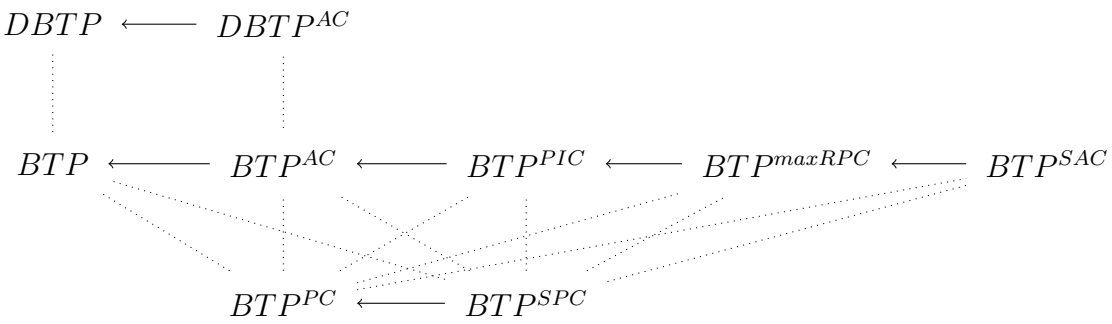


FIGURE 2.4 – Quelques relations entre des classes cachées de BTP et de $DBTP$.

- *t préserve la satisfiabilité et peut être calculée en temps polynomial.*

Ainsi, si $\mathcal{C}$ est une classe polynomiale et si $\mathcal{P}$ est une classe cachée de $\mathcal{C}$ pour une transformation t calculable en temps polynomial et préservant la satisfiabilité, toute instance $P \in \mathcal{P}$ sera traitable en temps polynomial. En effet, il suffit d'appliquer t sur P pour avoir une version modifiée de P possédant le même ensemble de solutions qui appartient à $\mathcal{C}$ et qui est donc traitable en temps polynomial. La classe $\mathcal{P}$ peut ainsi être considérée comme une classe polynomiale.

Nous avons pu mettre en évidence les propriétés que possède cette approche dans le cas de transformations générales. Certaines relations existantes entre les classes sont notamment conservées par les transformations. Par exemple, nous pouvons facilement prouver que pour toutes classes $\mathcal{C}_1$ et $\mathcal{C}_2$ telles que $\mathcal{C}_1 \subseteq \mathcal{C}_2$, et pour toute transformation t, on a $\mathcal{C}_1^t \subseteq \mathcal{C}_2^t$. De plus, d'autres propriétés peuvent être établies en considérant des transformations particulières comme les filtrages. Utiliser les techniques de filtrage comme transformations semble tout à fait naturel car ce sont les transformations les plus exploitées en pratique, que ce soit en prétraitement ou bien durant la résolution. Nous pouvons alors démontrer que, sous certaines conditions, les relations existantes entre les filtrages (Debruyne et Bessière, 2001; Bessière et al., 2008) induisent les relations entre classes cachées d'une même classe :

Propriété 2.2 *Soient t_1 et t_2 deux transformations qui sont des filtrages tels que t_2 est strictement plus fort que t_2 (au sens de (Debruyne et Bessière, 2001)). Pour toute classe $\mathcal{C}$ conservative par rapport à t_1 et t_2, nous avons $\mathcal{C}^{t_1} \subseteq \mathcal{C}^{t_2}$.*

La figure 2.4 présente quelques classes cachées des classes BTP et DBTP obtenues via différentes cohérences locales et les relations entre ces classes.

2.3.2 Bilan des observations effectuées

Nous nous intéressons, à présent, à l'intérêt pratique que peuvent receler les classes polynomiales. Dans ce but, nous avons considéré les instances habituellement utilisées pour les compétitions de solveurs (voir par exemple (CPA, 2008, 2009)). Dans un premier temps, nous avons testé l'appartenance de ces instances à quelques-unes des classes polynomiales citées ou définies dans ce manuscrit. Dans un second temps, nous avons établi des liens avec l'efficacité pratique des solveurs énumératifs classiques. Enfin, nous avons évalué l'intérêt des règles de fusion présentées précédemment.

2.3.2.1 Appartenance à des classes polynomiales

Pour pouvoir tester l'appartenance des instances à des classes polynomiales, la première étape consiste à proposer des algorithmes de reconnaissance, puis à les mettre en œuvre. Tous ces algorithmes ont une complexité en temps polynomial. Toutefois, pour différentes raisons, cela ne garantit pas pour autant que leur exécution sera

rapide. Dans certains cas, les raisons sont inhérentes à la taille de l'instance (avec, par exemple, un nombre de variables élevé ou des domaines de grande taille). Dans d'autres cas, elles découlent de l'hypothèse de travail faite par beaucoup de classes relationnelles ou hybrides qui considère que les relations sont exprimées en extension. Aussi, souvent, afin de réduire le temps de calcul, nous avons testé l'appartenance à des classes polynomiales cachées obtenues via un filtrage plutôt qu'aux classes polynomiales originelles. Enfin, hormis pour certaines classes structurelles, les instances ayant des contraintes globales n'ont pas pu être testées car notre bibliothèque CSP ne les prend pas en compte. Au final, nous avons considéré 3 795 instances binaires et 2 586 instances non binaires.

Nous présentons maintenant le fruit de nos observations concernant les classes polynomiales en commençant par la classe BTP et ses classes cachées. La table 2.1 présente le nombre d'instances ayant satisfait (ligne Oui) ou non (ligne Non) le test d'appartenance à BTP ou à une de ses classes cachées obtenues par application d'un certain filtrage ϕ. Parmi les instances appartenant à une de ces classes, certaines le sont trivialement car l'application du filtrage correspondant permet de détecter l'incohérence globale de l'instance. Aussi, nous précisons le nombre d'instances appartenant aux différentes classes et satisfiant le niveau de cohérence imposé par ϕ. Cette table permet de mettre en évidence l'intérêt des classes cachées dans la mesure où le nombre d'instances appartenant à une classe polynomiale augmente notablement simplement en passant de BTP à BTP^{AC}. Notons que parmi les instances appartenant à BTP, huit possèdent un graphe de contraintes acyclique.

Les tables 2.2 et 2.3 présentent les résultats correspondant pour un certain nombre de classes polynomiales (cachées) pour les instances binaires et non-binaires respectivement. Elles montrent que chaque classe contient plusieurs instances. Même si le nombre d'instances est plutôt faible, il s'agit en soi déjà d'un résultat intéressant dans la mesure où les classes polynomiales ont la réputation d'être généralement trop restrictives pour pouvoir exister dans la pratique. Ces tables mettent aussi en évidence que les classes proposées permettent d'identifier de nouvelles instances comme étant des instances appartenant à des classes polynomiales.

Enfin, la table 2.4 indique l'appartenance ou non de quelques instances à BTP ou à une de ses classes polynomiales cachées. Nous pouvons noter, dans cette table, la diversité de la nature des instances.

	BTP	BTP^{AC}	BTP^{PIC}	BTP^{maxRPC}	BTP^{SAC}	BTP^{SPC}
Oui	13	282	491	583	655	699
dont ϕ-cohérentes	-	46	47	47	47	71
Non	3 674	3 472	2 791	2 699	2 633	2 393

TABLE 2.1 – Nombre d'instances binaires qui appartiennent à BTP ou à une de ses classes polynomiales cachées via un filtrage ϕ (dont nombre d'instances ϕ-cohérentes) ou non.

	ZOA^{AC}	CC^{AC}	BTP^{AC}	BTP^{SPC}	ETP-SPC	3-BTP-SPC	DR-2	$DBTP^{AC}$	$IFUN^{AC}$	MC^{AC}
Oui	296	292	282	699	704	710	715	272	270	270
dont ϕ-c.	38	34	46	71	76	82	87	36	34	34
Non	3 551	3 555	3 472	2 393	1 606	614	1 668	2 037	3 522	3 475

TABLE 2.2 – Nombre d'instances binaires qui appartiennent à une des classes polynomiales cachées considérées via un filtrage ϕ (dont nombre d'instances ϕ-cohérentes) ou non.

	$DBTP^{AC}$	$IFUN^{AC}$	MC^{AC}
Oui	171	104	130
dont ϕ-cohérentes	67	0	26
Non	297	1 910	1 732

TABLE 2.3 – Nombre d'instances non binaires qui appartiennent à une des classes polynomiales cachées considérées via un filtrage ϕ (dont nombre d'instances ϕ-cohérentes) ou non.

Instances	BTP	BTP^{AC}	BTP^{PIC}	BTP^{maxRPC}	BTP^{SAC}	BTP^{SPC}
bqwh-15-106-43_ext	non	non	non	non	non	oui
domino-100-100	non	oui	oui	oui	oui	oui
ehi-90-315-96_ext	non	non	oui	oui	oui	oui
ehi-90-315-97_ext	non	non	non	oui	oui	oui
fapp17-0300-10	non	oui	oui	oui	oui	oui
graph12-w0	oui	oui	oui	oui	oui	oui
hanoi-3_ext	oui	oui	oui	oui	oui	oui
langford-4-8	non	non	non	non	non	oui
large-80-sat_ext	non	oui	oui	oui	oui	oui
os-taillard-4-95-0	non	non	non	non	oui	oui
pigeons-20-ord	oui	oui	oui	oui	oui	oui
queens-4	non	non	oui	oui	oui	oui
rand-23-23-253-131-48021_ext	non	non	non	non	non	non
rand-2-40-180-84-900-93_ext	non	non	non	non	oui	oui
will199GPIA-6	non	non	non	non	non	non

TABLE 2.4 – Quelques instances appartenant à BTP ou à une de ses classes polynomiales cachées.

2.3.2.2 Lien avec la résolution

Au-delà de l'appartenance de certaines instances à des classes polynomiales, il peut être pertinent d'établir des liens entre les classes polynomiales et la résolution des instances par des algorithmes énumératifs classiques. Ces derniers s'avèrent d'une efficacité remarquable sur bon nombre d'instances sans pour autant exploiter explicitement des classes polynomiales. Cependant, certains algorithmes, comme MAC ou RFL, sont capables d'exploiter implicitement certaines classes polynomiales. Par exemple, RFL est capable de résoudre en temps polynomial les instances appartenant à la classe BTP ou ayant une microstructure CSG^k, et cela sans avoir besoin de mettre en œuvre des traitements spécifiques, ni de détecter leur nature.

Ainsi, nous pouvons observer en pratique que RFL ou MAC résolvent certaines instances sans retour en arrière. Plusieurs cas de figure sont possibles. Lorsque les instances appartiennent à des classes polynomiales implicitement capturées par RFL ou MAC comme BTP ou MC, le simple fait d'appartenir à ces classes explique l'efficacité de la résolution. Quand les instances appartiennent à des classes polynomiales comme $DBTP$ qui ne sont pas complètement implicitement capturées par RFL ou MAC, nous avons pu constater que les conditions requises pour une capture implicite, bien que restrictives, sont, à plusieurs reprises, remplies. Dans les autres cas, nous avons pu établir qu'après affectation d'un petit nombre de variables, l'instance résultante appartient à une classe polynomiale implicitement capturée par MAC ou RFL. Nous retrouvons ainsi la notion de *backdoor* introduite dans (Williams et al., 2003). Enfin, il demeure quelques instances pour lesquelles le nombre de variables à instancier est trop important. Pour ces instances, la raison de la remarquable efficacité de RFL ou MAC reste inconnue.

2.3.2.3 Fusion

Pour mesurer l'intérêt pratique de la fusion de valeurs par BTP-fusion ou par 1-wBTP-fusion, nous avons commencé par définir des algorithmes mettant en œuvre ces fusions. Ensuite, compte tenu que maximiser le nombre de fusions s'avère être un problème NP-difficile, nous utilisons des heuristiques pour trouver un ordre convenable. D'après nos expérimentations, les heuristiques considérées ont plus d'influence sur le temps de calcul de la fusion que sur le nombre de valeurs fusionnées. Dans les résultats présentés ici, un temps maximum d'une heure est alloué pour effectuer la fusion de valeurs jusqu'à l'obtention d'un point fixe. Les expérimentations ont été effectuées sur 8 serveurs-lames Dell PowerEdge M820 dotés de deux processeurs Intel Xeon E5-2609 v2 cadencés à 2,5 GHz et de 32 Go de mémoire et fonctionnant sous Linux Ubuntu 14.04.

Au total, nous avons obtenu des résultats pour 2 535 benchmarks sur les 3 795 instances binaires considérées. Pour 1 001 instances, au moins un domaine a été réduit par fusion. Dans le tableau 2.5, nous présentons les résultats de la fusion pour quelques instances et nous les comparons à la substitution de voisinage (NS) et l'interchangeabilité virtuelle (VI). Les fusions basées sur BTP ou 1-wBTP permettent de fusionner plus de valeurs que la substitution de voisinage et l'interchangeabilité virtuelle, avec, pour certaines instances, un gain significatif.

La figure 2.5 compare les pourcentages de valeurs supprimées par BTP-fusion et 1-wBTP-fusion instance par instance. Si, pour une majorité d'instances, les résultats de la BTP-fusion et de la 1-wBTP-fusion sont comparables, nous pouvons remarquer que pour certaines d'entre elles, la 1-wBTP-fusion fusionne significativement plus de valeurs que la BTP-fusion. C'est notamment le cas pour les instances des familles `langford-*` pour lesquelles la 1-wBTP-fusion fusionne de 25 à 80% des valeurs là où la BTP-fusion n'en fusionne aucune.

L'apport de la BTP-fusion a également été évalué du point de vue de la résolution. Pour cela, nous avons considéré l'algorithme MAC avec et sans prétraitement par BTP-fusion. Il s'avère que la BTP-fusion permet souvent à MAC d'explorer un espace de recherche de taille plus réduite mais aboutit généralement à un temps d'exécution supérieur à cause du coût de la BTP-fusion. Ce résultat est tout de même prometteur dans la mesure où l'efficacité chronométrique de l'algorithme de fusion peut sans aucun doute être améliorée significativement.

Instances	n	d	#valeurs	NS	VI	BTP	1-wBTP
BlackHole-4-4-e-1_ext	64	16	674	207	324	329	351
BlackHole-4-7-h-5_ext	112	28	2 102	697	887	896	932
bqwh-15-106-22_ext	106	6	365	3	0	3	3
geom-40-2-ext	40	2	80	1	1	1	14
composed-25-1-2-8_ext	33	10	330	0	0	0	6
driverlogw-02c-sat_ext	301	8	1 161	8	2	8	74
ehi-85-297-22_ext	297	7	2 079	0	0	891	1 058
ehi-90-315-33_ext	315	7	2 205	0	0	945	1 112
lei450-15b-13	450	13	5 850	0	24	24	24
fpsol2-i-1-62	496	62	30 752	13 847	13 847	13 847	13 847
zeroin-i-3-27	206	27	5 562	1 274	1 274	1 274	1 274
haystacks-28	784	28	21 952	0	27	27	27
enddr1-10-by-5-8	50	122	5 720	422	18	422	426
langford-2-17	34	34	1 156	0	0	0	306
langford-4-17	68	68	4 624	0	0	0	1 258
os-taillard-4-105-1	16	244	3 173	2 364	0	2 378	2 401
os-taillard-4-95-8	16	238	3 034	2 696	0	2 704	2 764
os-taillard-5-95-9	25	297	6 311	3 863	0	3 870	3 880
queenAttacking-9	82	81	6 576	0	0	0	110
qwh-10-57-8_ext	100	10	613	0	0	0	18
graph5	200	44	7 416	0	96	134	2 345
scen2	200	44	8 004	0	298	341	1 211
scen06-sub4	44	44	1 856	0	0	78	639
super-os-taillard-4-6	32	189	4 712	0	0	0	727

TABLE 2.5 – Nombre de variables, taille du plus grand domaine, nombre total de valeurs, nombre de valeurs supprimées par la substitution de voisinage (NS), par l'interchangeabilité virtuelle (VI) par la BTP-fusion et par la 1-wBTP-fusion pour quelques instances représentatives des familles pour lequelles la 1-wBTP-fusion supprime au moins une valeur.

2.4 Conclusion

Dans ce chapitre, nous avons proposé plusieurs nouvelles classes polynomiales hybrides et étudié leur relations avec les classes existantes. Nous avons également défini la notion de classe polynomiale cachée afin d'étendre encore davantage l'ensemble des instances traitables en temps polynomial. D'un point de vue pratique, bien que les classes polynomiales aient généralement la réputation d'être trop restrictives pour pouvoir exister, nous avons pu établir l'existence d'instances appartenant à certaines classes polynomiales parmi les instances couramment employées pour évaluer l'efficacité des solveurs. Ainsi, la remarquable efficacité de ces solveurs peut alors s'expliquer, dans certains cas, par l'exploitation implicite de certaines classes polynomiales (cachées). Ensuite, nous avons exploité des propriétés locales autour de la notion de triangles cassés pour définir deux règles de fusion de

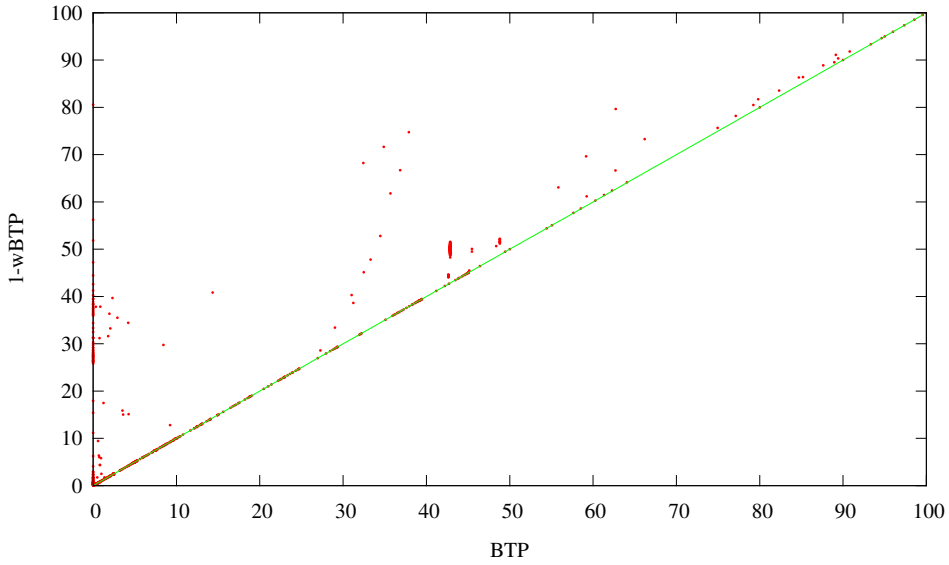


FIGURE 2.5 – Comparaisons des pourcentages de valeurs fusionnées par BTP-fusion et par 1-wBTP-fusion.

valeurs, qui réduisent parfois significativement la taille des domaines.

Au final, ces différents travaux combinent des aspects théoriques et pratiques. Il n'en demeure pas moins qu'il semble difficile d'exploiter les classes polynomiales hybrides explicitement du fait de l'existence de méthodes de reconnaissance et de résolution propres à chaque classe. Par contre, leurs exploitations implicites, comme le font MAC et RFL pour BTP par exemple, semblent constituer une voie des plus prometteuses par sa souplesse et sa simplicité.

Chapitre 3

Classes polynomiales structurelles et résolution

Sommaire

Les classes polynomiales structurelles semblent fournir les classes polynomiales les plus exploitables du point de vue de la résolution. D'abord, certaines d'entre elles comme la classe $TREE$ sont implicitement exploitables par des algorithmes classiques comme MAC ou RFL. Ensuite, elles font partie des rares classes polynomiales pouvant conduire à la définition d'algorithmes de résolution généraux. Dans ce cas-là, l'objectif est souvent de se ramener à une instance CSP acyclique en tirant profit de certaines propriétés structurelles. Ces propriétés peuvent notamment être relatives à une décomposition arborescente de l'hypergraphe de contraintes.

Dans ce chapitre, nous nous intéressons, dans un premier temps, aux méthodes de résolution. Plus précisément, nous décrivons d'abord une méthode énumérative (nommée BTD) reposant sur la notion de décomposition arborescente. Puis, nous proposons une nouvelle évaluation de la complexité d'algorithmes comme nFC$_2$ et présentons ses conséquences sur la hiérarchie des méthodes de décompositions structurelles de (Gottlob et al., 2000). Ensuite, nous définissons un cadre algorithmique générique permettant de capturer de nombreux algorithmes existants.

Dans un second temps, nous nous focalisons sur les méthodes calculant des décompositions d'hypergraphes. Après avoir rappelé quelques méthodes usuelles et considéré leur intérêt pratique du point de vue de la résolution, nous expliquons comment exploiter la notion de recouvrement acyclique avant de décrire un nouveau cadre algorithmique pour le calcul de décompositions arborescentes.

Enfin, nous présentons deux exploitations de la méthode BTD et deux de ses extensions. Les deux exploitations consistent à la mise en œuvre d'une méthode de résolution proposée dans (Jégou, 1990, 1991) et d'un algorithme de filtrage pour une nouvelle cohérence exploitant la structure que nous définissons. Concernant les extensions, il s'agit principalement d'adapter la méthode à des problèmes proches comme SAT ou l'optimisation sous contraintes.

Ces différents travaux ont été principalement accomplis en collaboration avec Philippe Jégou, Hanan Kanso, Samba Ndjoh Ndiaye ou Cédric Pinto. Pour une partie d'entre eux, ils se sont déroulés dans le cadre des projets ANR STAL-DEC-OPT (Stratégies et algorithmes pour la décomposition et la résolution de problèmes d'optimisation sous contraintes) et TUPLES.

3.1 Méthodes de résolution

Dans cette section, nous présentons d'abord la méthode BTD ainsi que quelques-unes de ses améliorations. Puis, nous donnons une nouvelle évaluation de la complexité d'algorithmes comme nFC_2. L'exploitation de ce résultat au niveau des méthodes de résolution par décomposition conduit à une mise à jour de la hiérarchie des méthodes de décompositions structurelles de (Gottlob et al., 2000). Enfin, nous définissons un cadre algorithmique générique permettant de capturer de nombreux algorithmes existants allant d'algorithmes purement enumératifs comme BT ou RFL à des méthodes énumératives structurelles comme BTD.

3.1.1 La méthode BTD

Lorsqu'on considère les algorithme de résolution associés aux classes polynomiales structurelles, il peut s'agir, dans certains cas, d'algorithmes de résolution énumératifs (comme, par exemple, MAC ou RFL pour la classe $TREE$). Cependant, bien souvent, ils reposent plutôt sur des approches de programmation dynamique (Bertelè et Brioschi, 1972). C'est, par exemple, le cas pour les instances ayant une décomposition arborescente de largeur bornée par une constante avec des méthodes comme Tree-Clustering ou la cohérence adaptative (Dechter et Pearl, 1989). Il en est de même pour les classes basées sur d'autres concepts de largeur (Gottlob et al., 2000; Grohe et Marx, 2014). Ces algorithmes procèdent généralement en trois étapes :

- décomposition de l'instance CSP en différents sous-problèmes,
- résolution de chaque sous-problème en réalisant la jointure des relations des contraintes présentes dans le sous-problème,
- construction et résolution d'une instance CSP acyclique équivalente à l'instance initiale à partir des solutions de chaque sous-problème.

Ils présentent alors plusieurs inconvénients. D'abord, la seconde étape nécessite de représenter les relations en extension, ce qui est parfois difficilement possible pour certaines contraintes en intention ou certaines contraintes globales du fait d'un nombre de tuples autorisés trop important. Ensuite, ces algorithmes ont besoin de stocker toutes les solutions de chaque sous-problème conduisant ainsi à une complexité spatiale exponentielle et à un coût en mémoire prohibitif d'un point de vue pratique. Enfin, les solutions des sous-problèmes sont calculées sans aucune garantie d'être compatibles avec au moins une solution des autres sous-problèmes. Au final, les exploitations pratiques de ce type d'approches sont plutôt rares (Amroun et al., 2016; Habbas et al., 2016). Aussi, à l'approche bottom-up de la programmation dynamique, nous préférons ici exploiter l'approche top-bottom des méthodes énumératives.

La méthode BTD (pour Backtracking on Tree-Decomposition (Jégou et Terrioux, 2003a)) est donc une méthode énumérative exploitant une décomposition (E, T) de l'hypergraphe de contraintes de l'instance P à résoudre. La principale différence existant entre cette méthode et des méthodes énumératives classiques comme BT ou RFL réside dans l'ordre dans lequel sont explorées les variables. Dans les méthodes classiques, aucune restriction n'est imposée à l'ordre sur les variables. Au contraire, BTD exploite un ordre sur les variables qui est induit par l'ordre selon lequel nous considérons les clusters. Tout ordre sur les clusters compatible avec un parcours en profondeur d'abord de la décomposition arborescente est exploitable. Ainsi, BTD débute son énumération avec les variables du cluster racine E_r de la décomposition arborescente considérée. Lorsqu'une solution de ce cluster racine est trouvée, BTD cherche à l'étendre sur un de ses clusters fils, et ainsi de suite. Notons que la résolution d'un cluster peut s'effectuer à l'aide de n'importe quel algorithme énumératif classique comme BT, FC ou RFL. De plus, l'ordre sur les variables induit par la décomposition utilisée est un ordre partiel. Ainsi, à l'intérieur d'un cluster, BTD reste libre d'ordonner les variables comme bon lui semble en employant l'heuristique de son choix (comme `dom/wdeg` par exemple).

Au cours de son exploration de l'espace de recherche, BTD mémorise des informations, sous la forme de goods ou de nogoods structurels, afin d'éviter certaines redondances. Un *good* (respectivement *nogood*) *structurel*

d'un cluster E_i par rapport à un de ses clusters fils E_j est une affectation cohérente sur $E_i \cap E_j$ qui peut (respectivement ne peut pas) être étendue de manière cohérente sur le sous-problème enraciné en E_j (c'est-à-dire le sous-problème constitué de tous les clusters figurant dans la descendance de E_j, E_j inclus). Ainsi, quand BTD termine la résolution d'un sous-problème donné, il va mémoriser le résultat sous la forme d'un good ou d'un nogood structurel. Ultérieurement, s'il rencontre à nouveau ce sous-problème, il continuera sa recherche sans avoir besoin de rechercher à nouveau une solution de ce sous-problème dans le cas d'un good ; il reviendra en arrière dans le cas d'un nogood. Notons que la validité des informations mémorisées découle du fait que l'intersection $E_i \cap E_j$ déconnecte E_j et sa descendance du reste du problème. Ces intersections constituent en fait des *séparateurs* de l'hypergraphe de contraintes. L'exploitation conjointe de la décomposition et de l'enregistrement de (no)goods structurels conduit à l'obtention des complexités suivantes :

Théorème 3.1 *La méthode BTD basée sur l'algorithme BT pour résoudre chaque cluster a une complexité en temps en* $O(n.a.s^2.e.\log(d^s).d^{w^+ + 1})$ *et une complexité en espace en* $O(n.s.d^s)$ *avec* w^+ *la largeur de la décomposition arborescente considérée et* s *la taille de la plus grande intersection entre deux clusters.*

Il est à noter que cette complexité en temps est potentiellement meilleure que celle des algorithmes énumératifs classiques puisque w^+ est toujours inférieur ou égal à $n-1$ et qu'il peut même être significativement plus petit. Il s'agit d'ailleurs d'un des atouts majeurs des méthodes structurelles par rapport aux méthodes énumératives classiques. De plus, BTD constitue une alternative à la cohérence adaptative ou au Tree-Clustering quand il s'agit de résoudre des instances appartenant à la classe polynomiale BTW_k.

Si, initialement, la méthode BTD a été décrite sur la base de l'algorithme BT pour résoudre chaque cluster, rien n'exclut d'utiliser un algorithme plus sophistiqué. Par exemple, il est possible de maintenir un certain niveau de cohérence en chaque nœud du moment où cette cohérence n'engendre pas de nouvelles contraintes remettant en cause la décomposition arborescente considérée (c'est-à-dire des contraintes dont la portée n'est pas incluse dans un cluster de la décomposition). De même, il est également possible d'exploiter une stratégie de branchement binaire (Jégou et Terrioux, 2014c, 2016) comme une stratégie de branchement non binaire (Jégou et Terrioux, 2003a).

L'efficacité pratique de BTD dépend de différents paramètres, parmi lesquels nous pouvons citer :

- la décomposition arborescente utilisée,
- le choix du cluster racine,
- l'ordre dans lequel les clusters fils d'un cluster donné sont considérés,
- le choix de l'heuristique d'ordonnancement des variables à l'intérieur de chaque cluster.

L'influence de la décomposition utilisée est multiple. Bien sûr, elle conditionne les complexités temporelle et spatiale. En particulier, exploiter des séparateurs de trop grande taille peut conduire à un coût en mémoire prohibitif et donc rendre la méthode inutilisable en pratique. De plus, comme les autres paramètres, elle influe directement sur la qualité de l'ordre partiel sur les variables qu'elle induit. Connaissant l'importance que peut revêtir l'utilisation d'un ordre dynamique pour l'efficacité d'une résolution, l'emploi d'un ordre partiel peut s'avérer problématique. Aussi, au cours de différents travaux (Jégou et Terrioux, 2003a; Jégou et al., 2006a,b, 2007a,b), nous avons proposé différentes solutions pour accorder plus de liberté à l'ordre sur les variables tout en garantissant des bornes de complexité en temps intéressantes. Dans les faits, les bornes de complexité obtenues dépendent directement du degré de liberté accordée et varient de la complexité du théorème 3.1 à la complexité d'algorithmes énumératifs classiques (en considérant une liberté totale). Toutefois, ces différentes solutions ne permettent pas de remettre en cause un mauvais choix de racine ou des choix malheureux effectués lors du calcul de la décomposition. Pour y remédier, nous avons proposé d'exploiter des redémarrages et de fusionner dynamiquement certains clusters.

L'emploi des redémarrages a pour objectif de permettre de choisir un nouveau cluster racine si l'algorithme le juge pertinent. Les redémarrages sont exploitées comme dans MAC+RST+NG (Lecoutre et al., 2007c). Autrement dit, nous exploitons une version de BTD basée sur l'algorithme MAC pour résoudre chaque cluster et une politique de redémarrage. Dans (Jégou et Terrioux, 2014c, 2016), nous avons considéré des politiques de redémarrage s'appuyant sur des suites géométriques ou des suites de Luby et basées sur le nombre de retours en arrière effectués. Toutefois, d'autres politiques sont possibles. De plus, le fait d'exploiter une décomposition arborescente permet de définir différents niveaux pour les politiques de redémarrage. Il peut s'agir d'une politique globale portant sur l'ensemble de la recherche comme dans (Jégou et Terrioux, 2014c, 2016), comme d'une politique locale à un cluster (par exemple basée sur le nombre de retour en arrière effectué au sein de chaque

cluster). Concernant les goods structurels, il est nécessaire de considérer leur orientation pour pouvoir continuer à les exploiter. Nous distinguons donc les goods d'un cluster E_i par rapport à un de ses fils E_j des goods de E_j par rapport à E_i. Pour les nogoods structurels, comme il s'agit de nogoods au sens classique du terme, ils peuvent être exploités sans tenir compte de l'orientation (c'est-à-dire de la racine courante). Cependant, la mise en évidence de (no)goods structurels pouvant nécessiter un temps plus ou moins long, nous n'avons pas de garantie qu'un (no)good structurel soit enregistré entre deux redémarrages. Aussi, à chaque redémarrage, nous mémorisons des nld-nogoods réduits (voir la définition 1.11) afin de garantir que l'espace de recherche restant à visiter diminue à chaque redémarrage. L'exploitation de la décomposition arborescente permet de limiter la taille de ces nogoods dans la mesure où l'enregistrement peut se faire cluster par cluster. À l'image de MAC+RST+NG, le stockage et l'exploitation des nld-nogoods s'effectuent par l'ajout d'une contrainte globale. La seule différence est que nous considérons ici une contrainte globale par cluster.

Dans (Jégou et Terrioux, 2014c, 2016), nous avons pu mettre en avant l'intérêt pratique des redémarrages dans le cadre des méthodes de résolution structurelles en considérant certaines instances de la compétition CSP 2008. Cet intérêt peut être dû simplement à l'emploi des redémarrages (par exemple pour les familles d'instances `jobshop` ou `geom`) ou à l'utilisation conjointe de la décomposition et des redémarrages (notamment pour les familles `renaultmod`, `superjobshop` or `scen11`).

Les redémarrages constituent un premier pas vers une décomposition arborescente dynamique dans la mesure où, à chaque redémarrage, BTD peut choisir une nouvelle racine. Un second pas peut être effectué en considérant la fusion dynamique de clusters. L'intérêt de cette fusion dynamique est double. D'une part, elle permet de prendre en compte des informations sémantiques explicitées durant la recherche comme le feraient des heuristiques de choix de variables adaptatives. Cet aspect sémantique peut s'avérer d'une importance non négligeable car souvent les décompositions arborescentes sont calculées sur la base de critères essentiellement structurels, et donc, sans tenir compte de la sémantique des contraintes. D'autre part, la fusion dynamique offre la possibilité d'accroître la liberté de l'ordre sur les variables lorsque cela semble nécessaire. Une possibilité, par exemple, consiste à fusionner un cluster E_i avec un de ses fils E_j, si durant la résolution, l'heuristique de choix de variables utilisée souhaite souvent privilégier une variable du cluster fils E_j avant une variable de E_i.

L'algorithme BTD-MAC+RST+Fusion (voir l'algorithme 3.1) intégre les techniques de redémarrage et de fusion dynamique de clusters. Bien entendu, l'ajout de ces deux techniques engendre des répercussions sur les complexités temporelle et spatiale :

Théorème 3.2 *BTD-MAC+RST+Fusion a une complexité en temps en* $O(R.((n.s^2.e.a.\log(d)+w'^+.N).d^{w'^+ +2}+ n.(w'^+)^2.d))$ *et une complexité en espace en* $O(n.s.d^s + w'^+.(d+N))$ *avec* w'^+ *la largeur de la décomposition arborescente finale,* s *la taille de la plus grande intersection* $E_i \cap E_j$ *de la décomposition initiale,* R *le nombre de redémarrages et* N *le nombre de nld-nogoods réduits mémorisés.*

En dépit de l'augmentation de la complexité temporelle, BTD-MAC+RST+Fusion bénéficie de l'apport de ces deux techniques et s'avère ainsi meilleur que les différentes versions existantes de BTD (Jégou et al., 2016).

La méthode BTD n'est pas la seule méthode structurelle reposant sur une énumération et l'enregistrement d'informations. D'autres méthodes ont été proposées (par exemple dans Pseudo tree-search (Freuder et Quinn, 1985), Learning Tree-Solve (Bayardo et Miranker, 1996), Recursive Conditioning (Darwiche, 2001), BCC (Baget et Tognetti, 2001), And/Or Search Graph (Dechter et Mateescu, 2004, 2007)). Bien qu'elles exploitent d'autres notions que la décomposition arborescentes, elles partagent certaines des problématiques de BTD (comme par exemple, le calcul d'une décomposition de qualité, le choix d'une bonne racine, un ordre partiel sur les variables ou encore la nécessité de contrôler la quantité de mémoire requise). Ainsi, les travaux présentés ci-dessus ne sont pas limités qu'à BTD et peuvent aussi avoir un intérêt pour les autres méthodes structurelles.

3.1.2 De nouvelles bornes de complexité

Habituellement, la complexité des algorithmes énumératifs comme FC ou RFL est calculée en dénombrant le nombre d'affectations pouvant être construites dans le pire des cas. Dans le chapitre précédent, nous avons présenté une évaluation de cette complexité basée sur le nombre de cliques maximales dans la microstructure pour les instances binaires ou la DR-microstructure pour les instances non binaires. À présent, nous envisageons une troisième évaluation dans le cas général des instances non binaires (Jégou et al., 2008a). Pour cela, nous considérons les algorithmes maintenant un niveau de cohérence au moins égal à celui de nFC$_2$. nFC$_2$ applique la

Algorithme 3.1 : BTD-MAC+Fusion ($P, \Sigma, E_i, V_{E_i}, G, N$)

Entrées : Σ : suite de décisions, E_i : cluster, V_{E_i} : ensemble de variables
Entrées-Sorties : $P = (X, D, C)$: CSP, G : ensemble de goods, N : ensemble de nogoods
Résultat : $vrai$ si Σ peut être étendue de façon cohérente aux variables de $V_{E_i} \cup \bigcup_{E_j \in Fils(E_i)} Desc(E_j)$, $inconnu$ si la recherche est interrompue, $faux$ sinon

si $V_{E_i} = \emptyset$ **alors**
  $résultat \leftarrow vrai$
  $S \leftarrow Fils(E_i)$
  tant que $résultat \notin \{faux, inconnu\}$ **et** $S \neq \emptyset$ **faire**
    Choisir un cluster $E_j \in S$
    $S \leftarrow S \backslash \{E_j\}$
    si $Pos(\Sigma)[E_i \cap E_j]$ *est un nogood dans* N **alors** $résultat \leftarrow faux$
    sinon
      si $Pos(\Sigma)[E_i \cap E_j]$ *n'est pas un good de* E_i *par rapport à* E_j *dans* G **alors**
        $résultat \leftarrow$ BTD-MAC+Fusion(P,Σ,E_j, $E_j \backslash (E_i \cap E_j)$,$G$,$N$)
        si $résultat = vrai$ **alors** Enregistrer $Pos(\Sigma)[E_i \cap E_j]$ comme good de E_i par rapport à E_j dans G
        sinon
          si $résultat = faux$ **alors**
            Enregistrer $Pos(\Sigma)[E_i \cap E_j]$ comme nogood de E_i par rapport à E_j dans N
          sinon
            si $fusion$ **alors** Fusionner E_j avec un de ses fils
            si *non* $redémarrage$ **alors**
              $S \leftarrow S \cup \{E_j\}$
              $résultat \leftarrow vrai$
  retourner $résultat$
sinon
  Choisir une variable $x \in V_{E_i}$
  Choisir une valeur $v \in d_x$
  $d_x \leftarrow d_x \backslash \{v\}$
  si *AC (*P,$\Sigma \cup \langle x = v \rangle$*)* **alors**
    $résultat \leftarrow$ BTD-MAC+Fusion(P,$\Sigma \cup \langle x = v \rangle$,$E_i$, $V_{E_i} \backslash \{x\}$,G,N)
  sinon $résultat \leftarrow faux$
  si $résultat = faux$ **alors**
    si $redémarrage$ *ou* $fusion$ **alors**
      Enregistrer nld-nogoods par rapport à la suite de décisions $(\Sigma \cup \langle x \neq v \rangle)[E_i]$
      retourner $inconnu$
    sinon
      si *AC (*P,$\Sigma \cup \langle x \neq v \rangle$*)* **alors**
        retourner *BTD-MAC+Fusion(*P,$\Sigma \cup \langle x \neq v \rangle$,$E_i$, V_{E_i},G,N*)*
      sinon retourner $faux$
  sinon retourner $résultat$

Algorithme 3.2 : BTD-MAC+RST+Fusion

Entrées : $P = (X, D, C)$: CSP
Résultat : $vrai$ si P possède une solution, $faux$ sinon

$G \leftarrow \emptyset$; $N \leftarrow \emptyset$
répéter
  Choisir un cluster racine E_r
  $résultat \leftarrow$ BTD-MAC+Fusion (P,$\emptyset$,E_r,E_r,G,N)
jusqu'à $résultat \neq inconnu$
retourner $résultat$

cohérence d'arc sur un sous-ensemble de contraintes en une seule passe. Il en résulte que l'ordre de traitement des contraintes peut avoir une incidence sur les valeurs supprimées et leur nombre. Aussi, nous commençons par définir l'algorithme nFC$_{2m}$ qui applique un filtrage équivalent au filtrage minimal accompli par nFC$_2$ pour l'ensemble des ordres potentiels de traitement des contraintes. À partir de cet algorithme, nous pouvons montrer que toute affectation partielle visitée par nFC$_{2m}$ est incluse dans la jointure d'une partie des relations associées aux contraintes. La complexité en temps de nFC$_{2m}$ s'exprime alors en $O(S.r^e)$ où S représente la taille de l'instance considérée. Ce résultat peut être raffiné en considérant la notion de recouvrement minimum. Etant donnés un ensemble fini X et une famille C de sous-ensembles de X, un *recouvrement* de X est un sous-ensemble $C' \subseteq C$ tel que $\bigcup_{c_i \in C'} c_i = X$. C' est un *recouvrement minimum* de X s'il n'existe pas de recouvrement C'' tel que $|C''| < |C'|$. La valeur $|C'|$ sera notée $k_{(X,C)}$. En considérant un recouvrement minimum de l'ensemble des variables par un minimum de contraintes, il est possible d'obtenir la complexité suivante :

Théorème 3.3 *La complexité en temps de nFC$_{2m}$ pour résoudre une instance CSP (X, D, C) est $O(S.r^{k_{(X,C)}})$.*

Bien entendu, ce résultat s'applique aussi à nFC$_i$ (pour i supérieur ou égal à 2) et à RFL. Trouver un recouvrement minimum constitue un problème NP-difficile. Cependant, ces algorithmes exploitent nativement un recouvrement minimum sans pour autant avoir besoin de le caculer.

Cette nouvelle évaluation a des conséquences sur l'évaluation des méthodes de décompositions structurelles. Dans (Gottlob et al., 2000), Gottlob et al. ont proposé une hiérarchie comparant certaines méthodes de décompositions structurelles. La figure 3.1(a) présente leur résultat. Un arc reliant une méthode M_1 à une méthode M_2 indique que M_2 généralise M_1 (c'est-à-dire que toute classe d'instances qui est résoluble en temps polynomial par M_1 l'est aussi par M_2). Nous pouvons noter que les méthodes basées sur une décomposition hyperarborescente occupent le sommet de cette hiérarchie et sont donc plus générales que celles exploitant une décomposition arborescente. Néanmoins, pour ces dernières, il est possible d'évaluer différemment leur complexité en considérant le résultat ci-dessus. Il suffit pour cela d'exploiter des algorithmes classiques comme nFC$_i$ (pour $i \geq 2$) ou RFL pour résoudre chaque cluster d'une décomposition arborescente construite à partir d'une décomposition hyperarborescente de largeur h. Ainsi, la résolution d'un cluster E_i, en considérant l'ensemble C_{E_i} des contraintes dont la portée est incluse dans E_i, peut s'effectuer en $O(S_{E_i}.r^{k_{(E_i,C_{E_i})}})$. Sachant que le maximum des $k_{(E_i,C_{E_i})}$ est majoré par h, les méthodes basées sur une décomposition arborescente ont alors une complexité en $O(S.r^h)$, c'est-à-dire la même complexité que celles basées sur l'hypertree-decomposition. Il est donc possible de mettre à jour à la hiérarchie comme le montre la figure 3.1(b).

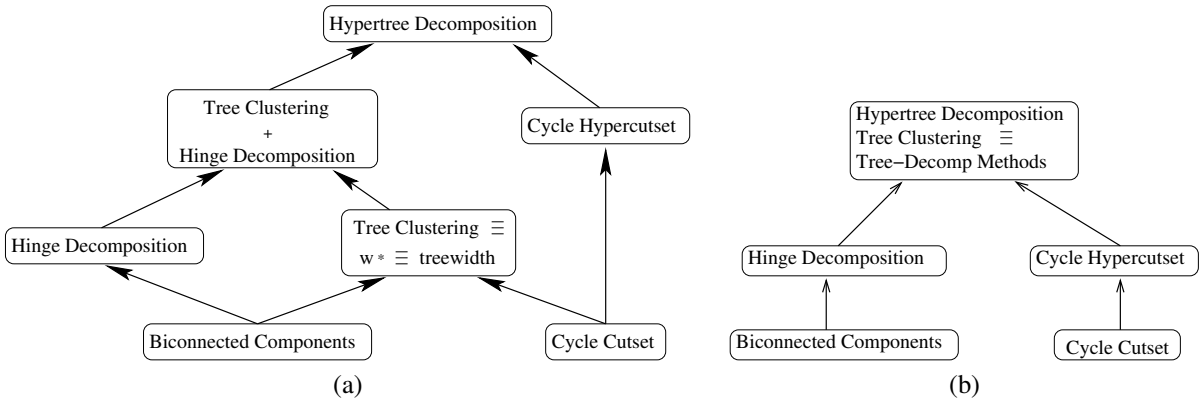


FIGURE 3.1 – (a) La hiérarchie des méthodes de décompositions structurelles (Gottlob et al., 2000), (b) la hiérarchie mise à jour grâce à la nouvelle évaluation de la complexité de nFC$_{2m}$.

La définition de cette hiéarchie considère la possibilité de déterminer, pour chaque instance, si elle possède une décomposition de largeur ℓ pour la notion de décomposition considérée. Bien que ce test soit réalisable en temps polynomial pour les décompositions considérées dans (Gottlob et al., 2000), il n'est pas pour autant très réaliste d'un point de vue pratique. En effet, en termes de méthodologie et de coût, il s'avère relativement éloigné des méthodes habituellement employées en pratique pour calculer des décompositions. Aussi, dans (Jégou et al.,

2009a), nous avons proposé une nouvelle hiérachie plus fine prenant en compte des couples formés d'une méthode de résolution et d'une méthode de calcul de décomposition.

3.1.3 Un cadre générique

Les différentes méthodes énumératives structurelles (comme (Freuder et Quinn, 1985; Bayardo et Miranker, 1996; Darwiche, 2001; Jégou et Terrioux, 2003a; Dechter et Mateescu, 2004, 2007)) possèdent de nombreux points communs. Par exemple, la plupart repose sur un ordre partiel sur les variables et mémorise des informations durant la résolution. De plus, toutes exploitent l'indépendance des sous-problèmes même si elles appréhendent la structure de l'instance grâce à des notions différentes. D'ailleurs, ces différentes notions peuvent s'exprimer d'une manière unique à travers les séparateurs qu'elles exploitent. Ainsi, nous proposons un cadre générique de méthode énumérative, appelé SBBT (pour Separator Based BackTracking (Ndiaye et Terrioux, 2007a)) qui repose sur certaines fonctions ou procédures paramétrables implémentant les étapes clés de la résolution comme :

- le choix de la prochaine variable (fonction $Heuristique_{var}$),
- le choix de la prochaine valeur (fonction $Heuristique_{val}$),
- l'identification des causes de l'échec (fonction $Failure$),
- l'enregistrement des (no)goods structurels (procédures $Good\_Recording$ et $Nogood\_Recording$),
- la gestion des (no)goods structurels (fonction $Check\_Good\_Nogood$ et procédure $Good\_Cancel$).

L'algorithme 3.3 décrit le cadre SBBT tandis que les algorithmes 3.4-3.8 donnent un exemple d'implémentation pour certaines fonctions et procédures évoquées ci-dessus.

Ce schéma générique nous permet alors de couvrir un large spectre d'algorithmes selon les choix effectués au niveau de l'ensemble de séparateurs et des procédures et fonctions. Ce spectre inclut des algorithmes allant des méthodes structurelles (par exemple BTD, BCC, Pseudo tree-search, Tree-Solve, Learning Tree-Solve, AND/OR Search Tree, AND/OR Search Graph) aux méthodes purement énumératives comme BT ou CBJ. L'emploi de techniques de filtrage est bien sûr possible sous réserve que le filtrage ne remette pas en cause l'existence des séparateurs considérés. Par exemple, nous pouvons définir une version de SBBT correspondant à RFL. Les complexités temporelle et spatiale sont directement liées aux choix effectués au niveau de l'ensemble de séparateurs et des procédures et fonctions. Ainsi, pour une instance ayant n variables et une largeur arborescente w, la complexité en temps varie entre $O(exp(w+1))$ et $O(exp(n))$ pour une complexité en espace en $O(n.s.exp(s))$ avec s la taille du plus grand séparateur.

Ce cadre générique possède de multiples avantages. D'abord, il permet de mieux mettre en avant les points communs et les différences existant entre les différentes méthodes structurelles. Ensuite, il ouvre de nouvelles perspectives en vue de l'exploitation conjointe de la structure et d'un ordre sur les variables le plus libre possible. En effet, dans ce cadre, quel que soit le degré de liberté de l'heuristique de choix de variables, il demeure possible, sous certaines conditions, de mémoriser et d'exploiter des (no)goods structurels. Enfin, même si la complexité en temps et en espace de SBBT dépend entre autres de l'ensemble de séparateurs utilisé et de l'heuristique de choix de variable, SBBT ne requiert aucune propriété particulière au niveau des séparateurs. Autrement dit, n'importe quel ensemble de séparateurs peut être exploité dans SBBT, offrant ainsi plus de possiblités pour calculer les séparateurs.

3.2 Calculs de décompositions

Le calcul d'une décomposition arborescente de qualité est une étape primordiale pour résoudre efficacement une instance avec la méthode BTD. Nous présentons d'abord nos travaux concernant l'exploitation des méthodes usuelles de calculs de décompositions arborescentes. Ensuite, nous nous intéressons aux recouvrements acycliques qui, d'une certaine manière, permettent de capturer simultanément plusieurs décompositions arborescentes. Enfin, nous proposons un nouveau cadre algorithmique pour le calcul de décomposition permettant d'intégrer certains critères à ce calcul.

Algorithme 3.3 : SBBT($\mathcal{A}, V, V_g$)

Entrées : $\mathcal{A}$: affectation, V : ensemble de variables
Entrées-Sorties : V_g : ensemble de variables
Résultat : $\emptyset$ si $\mathcal{A}$ peut être étendue de façon cohérente sur V, l'ensemble des variables expliquant l'échec sinon

```
si V − V_g = ∅ alors retourner ∅
sinon
    x ← Heuristique_var(V − V_g)
    d ← d_x
    J ← ∅
    Backjump ← faux
    tant que d ≠ ∅ et Backjump = faux faire
        v ← Heuristique_val(d)
        d ← d − {v}
        A' ← A ∪ ⟨x = v⟩
        si A' est cohérente alors
            si Check_Good_Nogood (A', x, V, V'_g, J) alors
                V_g ← V_g ∪ V'_g
                Good_Recording(A', x, V, V_g)
                J' ←SBBT(A', V − {x}, V_g)
                Good_Cancel(x, V, V_g)
                si x ∈ J' alors J ← J ∪ J'
                sinon
                    J ← J'
                    Backjump ← true
        sinon J ← J ∪ Failure(A', x)
    Nogood_Recording (A, x, V)
    retourner J
```

Algorithme 3.4 : Failure($\mathcal{A}', x$)

Entrées : $\mathcal{A}'$: affectation, x : variable
Résultat : l'ensemble des variables expliquant l'échec

```
retourner {x} ∪ {y ∉ V | ∃c ∈ C, S(c) = {x, y} et A' viole c}
```

Algorithme 3.5 : Check_Good_Nogood($\mathcal{A}', x, V, V'_g, J$)

Entrées : $\mathcal{A}'$: affectation, x : variable, V : ensemble de variables
Entrées-Sorties : V'_g, J : ensemble de variables
Résultat : $faux$ si $\mathcal{A}'$ contient un nogood, $vrai$ sinon

```
V'_g ← ∅
pour tous les S_j ∈ Sep t.q. S_j ∩ V = {x} faire
    pour tous les SP_{k,S_j} faire
        suivant A'[S_j] faire
            cas où good relatif à SP_{k,S_j} faire
                V'_g ← V'_g ∪ CC_{k,S_j}
            cas où nogood relatif à SP_{k,S_j} faire
                J ← J ∪ SP_{k,S_j}
                retourner faux
retourner vrai
```

Algorithme 3.6 : Nogood_Recording ($\mathcal{A}, x, V$)

Entrées : $\mathcal{A}$: affectation, x : variable, V : ensemble de variables
pour tous les $S_j \in Sep$ *t.q.* $S_j \cap V = \emptyset$ **faire**
  pour tous les CC_{k,S_j} *t.q.* $x \in CC_{k,S_j}$ **faire**
    si $J \cap CC_{k,S_j} \neq \emptyset$ **et** $CC_{k,S_j} \subseteq V$ **alors**
      Enregistrer $\mathcal{A}[S_j]$ comme nogood relatif à SP_{k,S_j}

Algorithme 3.7 : Good_Recording ($\mathcal{A}', x, V, V_g$)

Entrées : $\mathcal{A}'$: affectation, x : variable, V : ensemble de variables
Entrées-Sorties : V_g : ensemble de variables
pour tous les SP_{k,S_j} *t.q.* $SP_{k,S_j} \cap (V - V_g) = \{x\}$ **faire**
  Enregistrer $\mathcal{A}'[S_j]$ comme good relatif à SP_{k,S_j}
  $V_g \leftarrow V_g \cup CC_{k,S_j}$

3.2.1 Méthodes de calculs usuelles

La complexité en temps de BTD étant directement liée à la largeur de la décomposition employée, il semble naturel de vouloir employer une décomposition optimale (c'est-à-dire une décomposition de largeur w). Malheureusement, calculer une décomposition optimale s'avère être un problème NP-difficile et donc n'est pas envisageable comme étape préliminaire à une résolution d'une instance CSP. Aussi, dans la grande majorité des cas, le calcul d'une décomposition s'effectue par le biais d'une triangulation du graphe de contraintes dans le cas d'une instance binaire ou de la 2-section de l'hypergraphe de contraintes dans le cas d'une instance non binaire. Étant donné un graphe (X, C), la *triangulation* est une opération visant à ajouter un ensemble C' d'arêtes au graphe (X, C) de sorte à obtenir un graphe $(X, C \cup C')$ qui soit triangulé. Les cliques maximales du graphe $(X, C \cup C')$ sont alors calculables en temps polynomial et constituent les clusters d'une décomposition arborescente du graphe (X, C). Calculer une triangulation minimum (c'est-à-dire ajoutant un nombre minimum d'arêtes) est également un problème NP-difficile. Aussi, faute de pouvoir calculer une triangulation minimum, certains travaux (par exemple (Rose et al., 1976; Berry, 1999)) se sont intéressés au calcul de *triangulation minimale* (c'est-à-dire à des triangulations ajoutant un ensemble C' d'arêtes tel que pour tout sous-ensemble C'' de C', le graphe $(X, C \cup C'')$ ne soit pas triangulé). D'autres (Amir, 2001, 2010; Bouchitté et al., 2004) ont proposé des algorithmes garantissant une approximation de l'optimum par un facteur constant donné. Toutefois, d'un point de vue pratique, ces deux approches se révèlent généralement trop coûteuses en temps sans nécessairement fournir des décompositions intéressantes en termes de largeur. Enfin, l'approche heuristique avec des méthodes comme MCS (Tarjan et Yannakakis, 1984) ou Min-Fill (Rose, 1972) semble la plus intéressante à la fois en termes de temps d'exécution et de largeur. Par exemple, l'algorithme Min-Fill, qui est un des plus employés de nos jours, a une complexité temporelle en $O(n(n + e'))$ avec e' le nombre d'arêtes après triangulation et est connu pour fournir de bonnes approximations de décompositions optimales.

Nous avons comparé l'intérêt pratique des décompositions produites grâce aux triangulations minimales et aux triangulations heuristiques du point de vue de la résolution (Jégou et al., 2005a). Les décompositions produites par MCS et Min-Fill conduisent clairement aux meilleurs résultats en termes de nombre d'instances résolues et de temps d'exécution. Ces expérimentations ont également établi que la largeur de la décomposition arborescente

Algorithme 3.8 : Good_Cancel (x, V, V_g)

Entrées : x : variable, V : ensemble de variables
Entrées-Sorties : V_g : ensemble de variables
pour tous les $S_j \in Sep$ *t.q.* $S_j \cap V = \{x\}$ **faire**
  $V_g \leftarrow V_g - \bigcup_k CC_{k,S_j}$

utilisée n'est pas le seul paramètre à prendre en considération pour espérer avoir une résolution efficace. D'une part, si nous considérons deux décompositions ayant des largeurs différentes, celle ayant la plus petite largeur n'est pas nécessairement celle qui conduira à la résolution la plus efficace. D'autre part, la taille des intersections entre clusters joue aussi un rôle important. Bien entendu, elle permet de contrôler l'espace mémoire requis. Mais, son influence va bien au-delà. En effet, en limitant la taille des intersections (par exemple en fusionnant avec leur père les clusters dont le séparateur avec leur père est de taille supérieure à une valeur donnée), BTD est capable de résoudre plus d'instances et souvent avec des temps d'exécution réduits. Ce faisant, la quantité de mémoire requise est moindre et surtout, cela offre plus de liberté à l'ordre sur les variables.

Dans (Jégou et al., 2009a), nous avons détaillé comment produire et exploiter une décomposition arborescente à partir d'une décomposition hyperarborescente donnée. Toutefois, les décompositions hyperarborescentes visant à réduire le nombre de contraintes incluses dans chaque cluster, les décompositions arborescentes ainsi construites ne semblent pas des plus pertinentes pour la résolution d'instances CSP. Une nouvelle fois, celles produites par MCS et Min-Fill conduisent aux meilleurs résultats.

En dépit de certains bons résultats, les méthodes de calcul de décompositions procédant par triangulation peuvent parfois ajouter un grand nombre d'arêtes et ainsi conduire à une identification assez grossière de la structure de l'instance. Pour pallier ce défaut potentiel, une autre alternative (Pinto et Terrioux, 2008) est d'identifier un sous-ensemble de variables dont le retrait conduit à obtenir une 2-section triangulée. À partir de cette 2-section (qui n'est pas nécessairement connexe), il est alors possible de calculer une décomposition arborescente par composante connexe. Ces différentes décompositions sont ensuite réunies en une seule en ajoutant un cluster contenant les variables retirées tout en respectant les propriétés inhérentes aux décompositions arborescentes (ceci implique notamment que certaines variables retirées soient ajoutées à d'autres clusters). Pour certaines instances aléatoires structurées, ces décompositions ont permis d'obtenir de meilleurs résultats du point de vue de la résolution que celles calculées via une triangulation heuristique.

La principale conclusion commune à ces différents travaux est la nécessité de produire des décompositions adaptées à la résolution par une méthode comme BTD.

3.2.2 Recouvrements acycliques

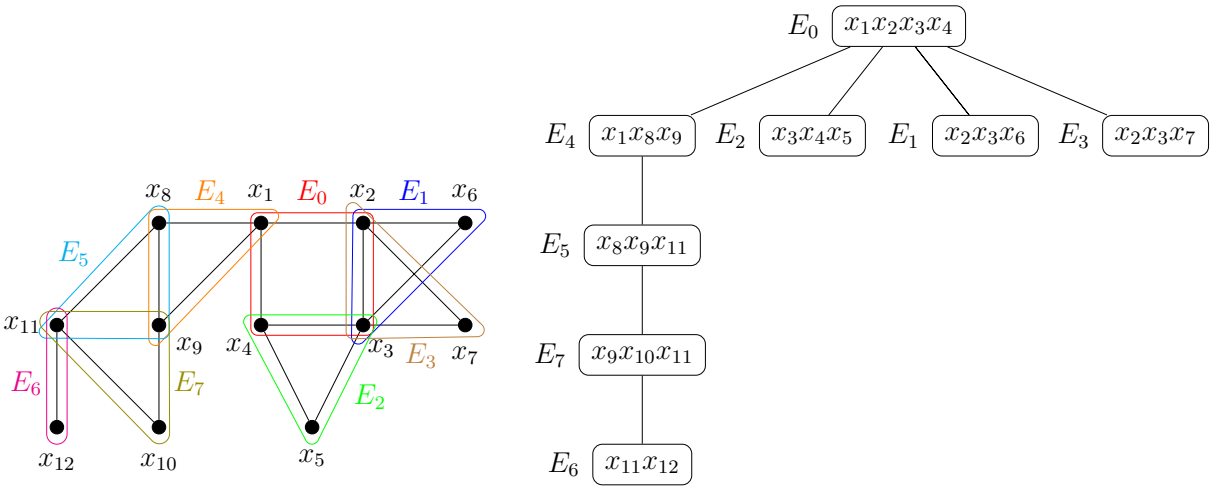
Nous considérons maintenant la notion d'hypergraphe acyclique recouvrant qui peut être vue comme une forme d'extension de la notion de décomposition arborescente (Jégou et al., 2007b).

Définition 3.1 (Hypergraphe acyclique recouvrant) *Un* hypergraphe acyclique recouvrant *(CAH pour Covering by an Acyclic Hypergraph) du graphe $G = (X, C)$ est un hypergraphe α-acyclique $H = (X, E)$ tel que pour chaque $\{x, y\} \in C$, il existe $E_i \in E$ tel que $\{x, y\} \subseteq E_i$.*
La largeur *d'un hypergraphe acyclique recouvrant (X, E) est égale à $max_{E_i \in E}|E_i| - 1$. La* CAH-largeur w^* *de G est la largeur minimale parmi tous les hypergraphes acycliques recouvrant de G.*

Exemple 3.1 *La figure 3.2 présente un recouvrement acyclique de largeur 3 du graphe de contraintes de la figure 1.3.*

Comme pour la décomposition arborescente, ce concept de recouvrement par un hypergraphe acyclique peut s'étendre aux hypergraphes en considérant leur 2-section. Notons que si la CAH-largeur d'un graphe est égale à sa largeur arborescente, ce concept s'avère moins restrictif que celui de décomposition arborescente. En effet, étant donné un graphe, il peut exister plusieurs décompositions arborescentes qui correspondent à un même ensemble de clusters structurés en arbre alors que pour ce même ensemble de clusters, il n'existera qu'un unique hypergraphe acyclique. Par exemple les figures 1.3(b) et 3.2(b) présentent deux décompositions arborescentes possibles pour l'ensemble de clusters exploité dans le recouvrement acyclique du graphe de la figure 1.3(a) présenté dans la figure 3.2(a). Ainsi, l'exploitation d'une décomposition arborescente par un algorithme de résolution comme BTD requiert de choisir une décomposition parmi celles-ci en effectuant un choix a priori arbitraire. Dans le cas de l'utilisation des recouvrements acycliques, l'unicité du recouvrement nous évite de faire un choix qui pourrait se révéler malheureux par la suite durant la résolution.

Maintenant, à partir d'un hypergraphe acyclique recouvrant H de l'hypergraphe de contraintes, nous pouvons définir différentes classes d'hypergraphes acycliques qui recouvrent H. Ces classes sont définies sur la base de critères relatifs à la nature des recouvrements et des relations existant avec les méthodes de résolution : bornes



(a) (b)

FIGURE 3.2 – Un recouvrement acyclique du graphe de contraintes de la figure 1.3 (a) et une autre décomposition arborescente de largeur 3 de ce graphe (b).

issues de paramètres comme la largeur arborescente, préservation des séparateurs de H, fusion d'hyperarêtes voisines, capacité à implémenter des heuristiques dynamiques efficaces, ... Un des objectifs ici est notamment de pouvoir proposer des compromis entre la vérification de bornes de complexité théorique de qualité et la nécessité péremptoire d'exploiter des heuristiques efficaces. Aussi, dans un premier temps, ces recouvrements ont été étudiés théoriquement de sorte à identifier leurs caractéristiques et leurs propriétés. Ensuite, nous avons montré qu'ils préservent les résultats de complexité connus, mais permettent en outre d'en améliorer certains. Pour cela, nous présentons une nouvelle variante de BTD, appelée BDH (pour Backtracking sur des recouvrements Dynamiques par Hypergraphes acycliques) pour laquelle il est assez facile d'étendre les heuristiques connues comme celles exploitées dans (Jégou et al., 2007a). Ainsi, pendant la résolution, nous pouvons prendre en compte, non seulement un recouvrement par hypergraphe acyclique, mais aussi un ensemble de recouvrements. Cette démarche facilite alors grandement une gestion dynamique des heuristiques d'un point de vue pratique, mais engendre également des répercussions du point de vue théorique. Par exemple, pour l'ordonnancement dynamique des variables, il devient alors possible d'agrandir dynamiquement à Δ variables supplémentaires, l'étendue du choix généralement offert au sein d'un cluster qui porte sur seulement au plus $w^+ + 1$ variables. Alors que dans (Jégou et al., 2007a), la complexité d'une telle démarche est en $O(exp(2(w^+ + \Delta + 1) - s^-))$ où s^- est la taille minimum des séparateurs, nous montrons qu'avec BDH, la complexité en temps est limitée à $O(exp(w^+ + \Delta + 2))$.

Sur un plan pratique, les expérimentations accomplies sur des instances aléatoires structurées ont montré l'intérêt que peut avoir une telle approche. Dans ces expérimentations, l'hypergraphe acyclique de référence est calculé à partir de la méthode MCS et des heuristiques basées sur la fusion d'hyperarêtes ont été définies pour calculer dynamiquement des hypergraphes recouvrants.

3.2.3 Un nouveau cadre algorithmique pour le calcul de décomposition

Les triangulations heuristiques comme MCS et Min-Fill ont permis de calculer des décompositions arborescentes rendant opérationnelle la méthode BTD. Cependant, ces décompositions ne sont pas pour autant toujours adaptées à une résolution efficace. D'abord, il faut garder à l'esprit que les méthodes heuristiques comme MCS et Min-Fill ont été proposées dans un cadre général et bien éloigné de celui de la résolution d'instances CSP. MCS est avant tout une méthode permettant de tester si un graphe est triangulé alors que Min-Fill vise à ajouter le moins d'arêtes possible pour trianguler le graphe. Aussi, si ces méthodes permettent souvent de produire des décompositions intéressantes et pertinentes en termes de largeur, ce n'est pas toujours le cas du point de vue de la résolution car la largeur de la décomposition n'est pas le seul paramètre qui influe sur l'efficacité de la résolution. Récemment, nous avons mis en lumière plusieurs défauts de ces algorithmes dans le cadre de la résolution d'ins-

tances CSP (Jégou et Terrioux, 2014a,d, 2016; Jégou et al., 2015b, 2016). D'abord, il s'avère que pour environ 32 % des 7 272 instances de la compétition CSP 2008 (dont la plupart des instances RLFAP ou FAPP), MCS ou Min-Fill produisent des décompositions arborescentes qui possèdent au moins un cluster non connexe. La présence de ces clusters non connexes peut, dans certains cas, être à l'origine de dégradations des performances pour un algorithme comme BTD. De plus, parfois, le pourcentage de clusters non connexes dans ces instances peut être considérable, atteignant jusqu'à 99 % et en moyenne aux environs de 35 %. Ensuite, la phase de triangulation peut engendrer l'ajout d'un grand nombre d'arêtes et ainsi s'avérer extrêmement coûteuse. Par ailleurs, un algorithme comme Min-Fill procède d'une certaine façon à l'aveugle en se basant sur des décomptes d'arêtes à ajouter sans se préoccuper des propriétés topologiques du graphe traité, du moins, explicitement. Enfin, souvent le nombre de sommets propres à chaque cluster (c'est-à-dire les sommets appartenant à un cluster, mais pas à son cluster père) est très réduit. Ainsi, il n'est pas rare d'avoir des décompositions ayant une largeur w^+ et une taille de séparateur maximale s proche ou même égale à w^+. Sachant l'importance que peut avoir la taille des séparateurs dans l'efficacité de la résolution, cela n'est pas sans conséquence sur la capacité de BTD à résoudre certaines instances. Aussi, nous présentons ici un nouveau cadre algorithmique pour le calcul de décomposition arborescente.

Ce nouveau cadre, nommé *H-TD-WT* (pour *Heuristic Tree-Decomposition Without Triangulation*), calcule une décomposition arborescente du graphe $G = (X, C)$ en temps polynomial sans procéder à une triangulation du graphe. Outre l'absence de triangulation, il se distingue des algorithmes existants par la possibilité de le paramétrer afin de produire des décompositions respectant un ou plusieurs critères. De nombreux critères sont envisageables comme la largeur w^+ de la décomposition et/ou la taille maximale de ses séparateurs s. Bien entendu, comme Min-Fill ou MCS, ce cadre n'offre aucune garantie quant à l'optimalité de la largeur obtenue.

Pour un graphe (X, C), la première étape de H-TD-WT (ligne 1 dans l'algorithme 3.9) calcule un premier cluster, noté E_0, à l'aide d'une heuristique. Puis, les composantes connexes du sous-graphe $G[X \backslash E_0]$ induit par la suppression dans G des sommets de E_0 [1] sont placées dans une file. Les éléments de la file sont ensuite traités les uns après les autres. Ainsi, pour une composante connexe donnée X_i, H-TD-WT identifie d'abord l'ensemble V_i de sommets déjà traités qui partagent une arête avec X_i (ligne 7), puis, calcule un nouveau cluster composé de V_i et d'un certain nombre de sommets de X_i (ligne 8). Le choix des sommets ajoutés est guidé par une heuristique dépendant des critères que nous souhaitons optimiser. Les sommets du cluster sont ensuite retirés du graphe et les nouvelles composantes connexes ainsi produites ajoutées à la file. Ce processus est répété jusqu'à ce que la file F soit vide.

Ce schéma est actuellement décliné en cinq heuristiques (Jégou et Terrioux, 2014a,d, 2016; Jégou et al., 2015b, 2016) :

- H_1 qui considère le sommet de V_i ayant le moins de voisins dans X_i,
- H_2 qui garantit que tous les clusters sont connexes et s'appuie en cela sur la notion de *Connected Tree-Width* récemment introduite en théorie des graphes dans (Müller, 2012; Diestel et Müller, 2014; Hamann et Weißauer, 2015),
- H_3 qui tente d'identifier des parties indépendantes dans le graphe et les sépare aussitôt que possible en effectuant un parcours en largeur à partir des sommets de V_i,
- H_4 et H_5 qui cherchent à maîtriser la taille des séparateurs de la décomposition (H_5 ayant une approche plus fine lui permettant ainsi de détecter plus de séparateurs que H_4).

La première heuristique vise à produire des décompositions ayant une largeur la plus petite possible alors que les quatre autres cherchent à optimiser la résolution de différentes manières. La complexité en temps de H-TD-WT avec l'une des heuristiques ci-dessus est en $O(n(n+e))$, c'est-à-dire une complexité meilleure que celle de Min-Fill. Notons que H-TD-WT avec H_2 est le premier algorithme permettant de calculer, d'un point de vue pratique, des décompositions garantissant des clusters connexes, les travaux existants autour de cette notion (Müller, 2012; Diestel et Müller, 2014; Hamann et Weißauer, 2015) étant purement théoriques. Par ailleurs, cela nous permet d'introduire un nouveau paramètre pouvant influer sur l'efficacité pratique des méthodes de résolution basées sur des décompositions comme BTD.

D'un point de vue pratique, nous avons considéré quatre versions de BTD à savoir BTD-MAC+RST+Fusion, BTD-MAC+Fusion, BTD-MAC+RST et BTD-MAC (c'est-à-dire BTD-MAC+RST+Fusion avec et sans fusion dynamique et/ou redémarrages). Pour chacune de ces versions et pour chacune des méthodes de décomposition

1. Le sous-graphe $G[Y]$ de $G = (X, C)$ induit par $Y \subseteq X$ est le graphe (Y, C_Y) où $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

Algorithme	Min-Fill		H_2		H_3		H_5	
	#rés.	temps	#rés.	temps	#rés.	temps	#rés.	temps
BTD-MAC	1 344	43 272	1 405	31 429	1 466	31 469	1 469	33 564
BTD-MAC+RST	1 495	43 557	1 518	35 042	1 529	30 187	1 543	33 049
BTD-MAC+Fusion	1 481	42 505	1 518	37 440	1 523	35 101	1 534	34 048
BTD-MAC+RST+Fusion	1 544	41 622	1 547	32 547	1 554	33 736	1 567	34 432

TABLE 3.1 – Nombre d'instances résolues et temps d'exécution en secondes pour BTD-MAC(+RST) et BTD-MAC(+RST)+Fusion selon la méthode de décomposition utilisée.

Algorithme 3.9 : H-TD-WT

Entrées : Un graphe $G = (X, C)$
Sorties : Un ensemble de clusters $E_0, \dots E_m$ d'une décomposition arborescente de G

Choix d'un premier cluster E_0 dans G
$X' \leftarrow E_0$
Soient $X_1, \dots X_k$ les composantes connexes de $G[X \backslash E_0]$
$F \leftarrow \{X_1, \dots X_k\}$
tant que $F \neq \emptyset$ **faire** `/* calcul d'un nouveau cluster` E_i `*/`
  Enlever X_i de F
  Soit $V_i \subseteq X'$ le voisinage de X_i dans G
  Déterminer un sous-ensemble $X_i'' \subseteq X_i$ tel qu'il existe au moins un sommet $v \in V_i$ tel que $N(v, X_i) \subseteq X_i''$
  $E_i \leftarrow X_i'' \cup V_i$
  $X' \leftarrow X' \cup X_i''$
  Soient $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ les composantes connexes de $G[X_i \backslash E_i]$
  $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

considérées, la table 3.1 présente le nombre d'instances résolues parmi les 1 859 instances issues de la compétition CSP 2008 considérées et le temps total d'exécution en secondes. Nous pouvons constater que quelle que soit la version de BTD considérée, l'exploitation des décompositions produites par H-TD-WT avec H_2, H_3 ou H_5 conduit à de meilleurs résultats que l'utilisation de décompositions calculées via Min-Fill aussi bien en nombre d'instances résolues qu'en temps de résolution. L'heuristique H_5 est, au final, celle qui conduit aux meilleurs résultats. Nous pouvons aussi constater que l'utilisation d'une version plus sophistiquée de BTD comme BTD-MAC+RST+Fusion permet de compenser en partie les faiblesses inhérentes à la décomposition exploitée. L'exploitation conjointe de BTD-MAC+RST+Fusion et de H-TD-WT avec H_5 permet de résoudre plus d'instances que toutes les autres combinaisons, ce qui montre bien la complémentarité des deux approches. Enfin, nous pouvons souligner l'efficacité chronométrique du calcul de décomposition avec H-TD-WT. En effet, le calcul des décompositions avec H_i ($i = 2, 3, 5$) s'avère nettement plus rapide qu'avec Min-Fill. À titre d'exemple, H_5 requiert seulement 7 s pour calculer les décompositions des 1 234 instances résolues par toutes les versions de BTD contre 7 582 s pour Min-Fill.

3.3 Exploitations et extensions

La méthode BTD a été exploitée au-delà du problème de décision CSP. D'une part, elle a rendu possible la mise en œuvre opérationnelle de la méthode Cyclic-Clustering (Jégou, 1990, 1991). D'autre part, elle a été exploitée pour définir un algorithme de filtrage pour une nouvelle forme de cohérence que nous proposons. Enfin, des extensions ont été proposées pour pouvoir aborder des problèmes voisins du problème CSP comme les problèmes d'optimisation sous contraintes ou le problème SAT.

3.3.1 Mise en œuvre opérationnelle de Cyclic-Clustering

La méthode structurelle Cyclic-Clustering (Jégou, 1990, 1991) réalise un compromis entre le temps et l'espace lui permettant d'obtenir des complexités en temps et en espace comprises entre celles du Tree-Clustering (Dechter

et Pearl, 1989) et celles de la méthode Coupe-Cycle (Dechter, 1990). Elle procède en quatre étapes. La première étape consiste à rechercher toutes les cliques maximales de la 2-section de l'hypergraphe de contraintes initial. La deuxième étape considère chaque clique maximale pour résoudre le sous-problème associé. Le résultat consiste en un hypergraphe de contraintes dont les contraintes correspondent aux différents sous-problèmes résolus. Les deux dernières étapes déterminent d'abord un ensemble coupe-cycle (c'est-à-dire un ensemble de sommets dont le retrait rend la structure acyclique), puis pour terminer, appliquent la méthode Coupe-Cycle sur ce nouveau CSP. La présentation dans (Jégou, 1990) se limitait essentiellement à présenter les idées de l'approche sans fournir d'éclaircissement pour une mise en œuvre effective de la méthode, contrairement à (Jégou, 1991) où de tels détails sont donnés. Dans (Jégou et Terrioux, 2004c), nous avons réalisé une telle mise en œuvre en exploitant la méthode BTD. Plus précisément, une des principales difficultés rencontrées pour rendre opérationnelle la méthode Cyclic-Clustering réside dans l'emploi de la méthode Tree-Clustering. Ainsi, le remplacement de cette dernière par la méthode BTD rend possible l'exploitation de cette méthode d'un point de vue pratique. Dans les faits, deux versions différentes (nommées respectivement CC-BTD$_1$ et CC-BTD$_2$) ont été proposées. CC-BTD$_2$ se distingue de CC-BTD$_1$ par une exploitation plus poussée des (no)goods structurels lui permettant d'éviter plus de redondances dans les calculs. Au final, CC-BTD$_1$ et CC-BTD$_2$ possèdent la même borne de complexité en temps que Cyclic-Clustering mais une meilleure borne de complexité en espace. CC-BTD$_1$ et surtout CC-BTD$_2$ présentent des résultats expérimentaux prometteurs, notammant en se révélant meilleurs que RFL ou BTD sur certaines instances ayant une structure adéquate (Jégou et Terrioux, 2004c).

L'application de CC-BTD$_1$ ou de CC-BTD$_2$ nécessite d'appeler plusieurs fois BTD sur une instance CSP qui évolue en permanence en fonction de l'affectation des variables de l'ensemble coupe-cycle mais dont la décomposition est toujours la même. Il en résulte que pour des raisons de validité, des (no)goods structurels sont mémorisés lors d'un appel et généralement oubliés lors de l'appel suivant. Aussi, nous avons proposé des conditions permettant la réutilisation de (no)goods structurels produits lors d'un appel précédent à BTD . De plus, nous avons pu constater qu'il n'est pas toujours nécessaire, ni pertinent, d'attendre que toutes les variables de l'ensemble coupe-cycle soient instanciées pour faire appel à BTD. L'exploitation de BTD avant d'avoir instancié tout l'ensemble coupe-cycle permet en effet d'une part, une détection précoce de certains échecs et, d'autre part, une meilleure exploitation des (no)goods. Ces différentes modifications nous ont conduit à définir un nouvel algorithme appelé CC-BTD-gen. Les complexités en temps et en espace de CC-BTD-gen sont du même ordre que celles de CC-BTD$_1$ et de CC-BTD$_2$ pour une efficacité pratique accrue (Pinto et Terrioux, 2009a).

3.3.2 Une forme de cohérence basée sur la structure

Nous introduisons ici une nouvelle forme de cohérence pour les instances CSP que nous appelons k-SC (pour k-Structural Consistency (Jégou et Terrioux, 2010a,b, 2014b)). Comme son nom l'indique, elle s'appuie sur la structure de l'instance et un paramètre k. Cette cohérence est basée sur une approche significativement différente de celles habituellement en usage. En effet, tandis que les cohérences classiques reposent généralement sur des propriétés locales étendues à l'ensemble du réseau, cette cohérence partielle considère à l'opposé la cohérence globale de sous-problèmes. Ces sous-problèmes sont définis par des hypergraphes de contraintes partiels dont la largeur arborescente est bornée par une constante k, qui correspond au paramètre associé à la cohérence k-SC. Plus précisément, nous exploitons la notion de *décomposition arborescente partielle* de largeur k (notée k-PST pour partial spanning tree-decomposition). Une décomposition arborescente partielle recouvrante de largeur k d'un graphe $G = (X, C)$ est un graphe partiel de G dont la largeur arborescente est k. Comme précédemment, cette notion peut s'étendre aux hypergraphes en considérant leur 2-section.

Définition 3.2 *Soient une instance CSP $P = (X, D, C)$ et un k-PST $G = (X, W)$ associé à l'hypergraphe de contraintes de P.*

- *La valeur $v_i \in d_{x_i}$ est k-SC-cohérente par rapport à G si l'instance P restreinte aux contraintes dont la portée est présente dans W et dont le domaine de x_i est restreint à la valeur v_i possède une solution.*
- *Le domaine d_{x_i} est k-SC-cohérent par rapport à G si $\forall v_i \in d_{x_i}$, la valeur v_i est k-SC-cohérente par rapport à G.*
- *L'instance P est k-*SC-cohérente *par rapport à G si $\forall d_{x_i} \in D$, d_{x_i} est k-SC-cohérent par rapport à G.*

Nous introduisons un algorithme de filtrage qui permet d'obtenir la k-SC cohérence. Il procède en considérant les valeurs des domaines les unes après les autres et en testant pour chacune si elle est k-SC-cohérente. Ce dernier

test est accompli à l'aide de la méthode BTD. Comme c'est le cas habituellement dans les algorithmes de filtrage, seules les valeurs satisfaisant la propriété sont conservées, les autres étant donc supprimées. L'algorithme itère ainsi jusqu'à l'obtention du point fixe ou la survenue d'un domaine vide. Sa complexité en temps est directement liée au paramètre k :

Théorème 3.4 *La complexité en temps de l'algorithme de filtrage est en $O(n^2.k.d^{k+2})$ alors que sa complexité en espace est $O(n.k.d^s)$ avec s la taille du plus grand séparateur dans le k-PST considéré.*

Sur un plan théorique, nous avons comparé cette nouvelle cohérence aux cohérences existantes. Nous avons ainsi pu établir les relations suivantes au sens de (Debruyne et Bessière, 2001) :

Théorème 3.5

- *1-SC est moins puissant que AC ,*
- *k-SC est moins puissant que la $(k+1)$-cohérence forte,*
- *k-SC (avec $k > 1$) est incomparable avec AC, SAC et la k-cohérence forte.*

L'efficacité de cette cohérence dépend évidemment de la valeur de k et du k-PST considérés. Il existe différentes possibilités pour calculer un k-PST. Dans tous les cas, il semble pertinent de choisir prioritairement les contraintes les plus dures pour pouvoir espérer filtrer le plus de valeurs possibles. Quant à la valeur de k, nos expérimentations ont montré que la valeur 6 semble offrir un bon compromis entre la puissance du filtrage et son coût.

3.3.3 Extension aux CSP valués et à SAT

Compte tenu de la qualité des résultats obtenus en pratique par la méthode BTD au niveau du problème de décision, il semble naturel d'étendre cette approche à d'autres problèmes voisins, notamment aux problèmes d'optimisation sous contraintes et à la logique propositionnelle.

Dans le premier cas, le cadre de travail utilisé pour représenter des problèmes d'optimisation est celui des CSP valués (VCSP (Schiex et al., 1995; Bistarelli et al., 1996; Meseguer et al., 2006)). Une des principales difficultés rencontrées lors de la résolution de VCSP consiste à ne pas prendre en compte plusieurs fois la même contrainte. Aussi, l'extension de l'algorithme BTD au cadre des CSP valués nécessite de définir un nouveau cadre formel dans lequel l'ensemble des contraintes est partitionné en accord avec la décomposition arborescente exploitée. Cette partition est en particulier exploitée pour définir la notion de sous-problèmes. La résolution de chaque sous-problème donne alors lieu à la mémorisation de goods structurels valués, qui peuvent ensuite être exploités pour éviter certaines redondances dans la recherche. Au final, cette adaptation de BTD parvient à atteindre les mêmes bornes de complexités en temps et en espace que dans le cadre du problème de décision (Terrioux et Jégou, 2003; Jégou et Terrioux, 2004a,b). Sur le plan pratique, elle obtient des résultats intéresssants dans le cas d'instances structurées (Jégou et Terrioux, 2004a). Toutefois, elle présente un inconvénient non négligeable, à savoir l'impossibilité de tirer le meilleur profit possible des différentes formes existantes de cohérences locales valuées. Ce défaut a été corrigé dans une nouvelle version de BTD proposée par de Givry et al. (de Givry et al., 2006). Par ailleurs, certains des travaux décrits précédemment dans le cadre du problème de décision (notamment ceux portant sur l'exploitation d'ordres plus dynamiques) ont également été étendus dans le cadre de l'optimisation (Jégou et al., 2006c, 2007d, 2008b; Ndiaye et al., 2008). Enfin, certains de ces résultats ont été intégrés au sein du solveur Toulbar2/BTD (Sánchez et al., 2008) qui a remporté la compétition internationale CPAI 2008 dans la catégorie « instances Max-CSP exprimées par des contraintes n-aires en extension ».

Concernant la logique propositionnelle, le problème SAT (Biere et al., 2009) est un problème central en théorie de la complexité et en intelligence artificielle. Bien que sa proximité avec le formalisme CSP soit immédiate, adapter la méthode BTD à ce problème nécessite, une nouvelle fois, de définir un nouveau cadre formel. En effet, si dans le cadre du problème CSP, il est nécessaire d'affecter chaque variable pour obtenir une solution, dans le cas du problème SAT, une solution (ou modèle) peut ne contenir qu'une partie des variables. Si cette différence peut sembler minime, elle se révèle très importante au niveau de l'exploitation de la décomposition arborescente et des sous-problèmes qu'elle engendre. Plus précisément, dans le cadre de SAT, il est possible que seule une partie des variables d'un sous-problème donné soient instanciées avant de passer au sous-problème suivant. Or, si ces deux sous-problèmes ne sont pas indépendants, cela peut remettre en cause la validité de l'algorithme. Pour pallier ce problème, le cadre formel a dû être enrichi en introduisant notamment la notion de « variables d'indépendance ».

D'un point de vue pratique, si l'intérêt de cette approche a pu être mis en évidence pour certaines instances, de nombreuses questions se posent concernant la possibilité de décomposer en temps raisonnable certaines instances SAT industrielles ou encore d'intégrer une telle approche dans les solveurs CDCL actuels (Marques Silva et al., 2009). Ce travail a été réalisé en collaboration avec Djamal Habet et Lionel Paris (Habet et al., 2009a).

3.4 Conclusion

Dans ce chapitre, nous nous sommes intéressés à différentes exploitations des classes polynomiales structurelles.

Dans un premier temps, nous avons d'abord proposé une méthode énumérative qui, en s'appuyant sur une décomposition arborescente, est capable de résoudre des instances CSP quelconques avec une complexité exponentielle dans la largeur de la dite décomposition. Il s'agit de la première mise en œuvre opérationnelle d'une telle méthode, les autres travaux concernant l'exploitation des décompositions arborescentes étant essentiellement théoriques. Par ailleurs, en proposant une nouvelle évaluation de la complexité d'algorithmes comme nFC$_2$ ou RFL, nous avons pu positionner les méthodes de résolution basées sur les décompositions arborescentes au même niveau (en l'occurrence le niveau le plus haut) que celles exploitant des décompositions hyperarborescentes dans la hiérarchie de (Gottlob et al., 2000). Ensuite, nous avons présenté un cadre générique permettant de capturer de nombreuses méthodes énumératives structurelles, les présentant ainsi sous un jour nouveau.

Dans un second temps, nous avons considéré le calcul de décomposition arborescente avec en ligne de mire la définition de décompositions adaptées à la résolution. Nous avons ainsi exploité la notion de recouvrements acycliques. Puis, nous avons défini un nouveau cadre algorithmique pour calculer des décompositions sans triangulation qui peut être paramétré afin de prendre en compte différents critères que doit posséder la décomposition arborescente calculée.

Dans un troisième temps, la méthode BTD a rendu possible la mise en œuvre opérationnelle de la méthode Cyclic-Clustering (Jégou, 1990) et d'une nouvelle forme de cohérence basée sur la structure. Enfin, des extensions ont été proposées pour pouvoir aborder des problèmes voisins du problème CSP comme les problèmes d'optimisation sous contraintes ou le problème SAT.

Conclusion et perspectives

Cette première partie du manuscrit a présenté les différents travaux que nous avons mené autour des classes polynomiales. Ces travaux se divisent en deux axes.

Le premier axe concerne les classes polynomiales hybrides. Nous avons ainsi proposé plusieurs nouvelles classes polynomiales hybrides et étudié leurs relations avec les classes existantes. Nous avons également défini la notion de classe polynomiale cachée qui nous permet d'étendre encore davantage l'ensemble des instances traitables en temps polynomial. D'un point de vue pratique, bien que les classes polynomiales aient généralement la réputation d'être trop restrictives pour pouvoir exister dans la réalité, nous avons pu mettre en évidence l'appartenance à des classes polynomiales de certaines instances habituellement utilisées par la communauté CSP pour évaluer l'efficacité des solveurs. Ainsi, la remarquable efficacité de ces solveurs peut alors s'expliquer, dans certains cas, par l'exploitation implicite de certaines classes polynomiales. Ensuite, nous avons exploité des propriétés locales autour de la notion de triangles cassés pour définir deux règles de fusion de valeurs, qui réduisent parfois significativement la taille des domaines.

Le second axe porte sur différentes exploitations des classes polynomiales structurelles. Nous avons, dans un premier temps, défini une méthode énumérative qui repose sur la notion de décomposition arborescente de graphe et qui est capable de résoudre des instances CSP quelconques. Au niveau théorique, cette méthode offre des bornes de complexité en temps parmi les meilleures existantes tandis qu'elle s'avère particulièrement efficace en pratique sur des instances possédant une structure adéquate. Sa mise en œuvre opérationnelle a notamment nécessité de proposer des méthodes de calcul de décomposition pertinentes du point de vue de la résolution. Nous avons défini, dans ce but, un nouveau cadre algorithmique pour calculer des décompositions sans triangulation qui peut être paramétré afin de prendre en compte différents critères que doit posséder la décomposition arborescente calculée. De plus, en proposant une nouvelle évaluation de la complexité d'algorithmes comme nFC$_2$ ou RFL, nous avons pu positionner les méthodes de résolution basées sur les décompositions arborescentes au niveau le plus haut dans la hiérarchie de (Gottlob et al., 2000), à égalité avec celles exploitant des décompositions hyperarborescentes. Par ailleurs, nous avons présenté un cadre générique permettant de capturer de nombreuses méthodes énumératives structurelles et de les présenter sous un jour nouveau. Enfin, des extensions de certains de ces travaux ont été proposées pour pouvoir aborder des problèmes voisins du problème CSP comme les problèmes d'optimisation sous contraintes ou le problème SAT.

Ces différents travaux offrent de nombreuses perspectives. Nous présentons, à présent, quelques-unes des pistes envisagées pour étendre ces travaux dans le futur.

Concernant les classes polynomiales hybrides, les travaux présentés dans ce manuscrit combinent des aspects théoriques et pratiques. Il n'en demeure pas moins qu'il semble difficile d'exploiter les classes polynomiales hybrides explicitement du fait de l'existence d'algorithmes de reconnaissance et de résolution propres à chaque classe. Aussi, la voie la plus prometteuse consiste à exploiter les classes polynomiales de façon implicite comme le font MAC et RFL pour BTP par exemple. Bien entendu, une première possibilité est évidemment d'identifier de nouvelles classes polynomiales. Une seconde est de proposer de nouveaux algorithmes généraux capables d'exploiter implicitement un plus grand nombre de classes polynomiales hybrides tout en garantissant une certaine efficacité pratique. Ensuite, nous avons vu que l'exploitation implicite d'une classe polynomiale par un solveur pouvait nous permettre d'expliquer, dans certains cas, sa capacité à résoudre efficacement certaines instances, soit directement, soit par l'intermédiaire de la notion de backdoor (Williams et al., 2003). L'étape suivante consiste naturellement à considérer plusieurs classes polynomiales simultanément. Par exemple, une partie de l'espace de recherche peut relever d'une certaine classe polynomiale alors qu'une autre partie est relative à une autre classe.

La fusion de valeurs basées sur les propriétés BTP et m-wBTP offrent aussi des perspectives intéressantes.

L'objectif ici est clairement de réduire les temps de calcul pour pouvoir exploiter cette approche en amont de la résolution. Cela passe d'abord par la définition d'algorithmes plus efficaces pour la mise en œuvre de ces fusions. Ensuite, comme trouver le meilleur ordre possible dans lequel effectuer les fusions constitue un problème NP-difficile, la mise au point d'heuristiques pertinentes semble une nécessité. Par ailleurs, si, actuellement, nos algorithmes effectuent des fusions jusqu'à l'obtention d'un point fixe ou d'un domaine vide, il pourrait se révéler pertinent de rechercher un compromis satisfaisant entre le nombre d'itérations accomplies et le nombre de valeurs supprimées.

En ce qui concerne les classes polynomiales structurelles, l'efficacité pratique des méthodes structurelles comme BTD demeure très dépendante de la qualité de la décomposition employée. Aussi, une première piste est, bien sûr, de mieux appréhender les paramètres influençant cette efficacité et de proposer de nouveaux algorithmes de calcul de décomposition. Au-delà, il peut être intéressant d'accroître encore la liberté de ces méthodes concernant l'ordre sur les variables. Une alternative alors envisageable est d'aller encore plus loin dans la dynamicité de la décomposition. Cela peut passer par d'autres opérations que la fusion dynamique de clusters, mais aussi par le recalcul d'une partie ou de toute la décomposition sur la base d'informations sémantiques explicitées durant la résolution. De telles solutions peuvent alors nécessiter de proposer de nouveaux algorithmes de résolution. Ensuite, de nombreuses autres notions de décompositions ont été introduites, notamment en théorie des graphes, comme, par exemple, (Seymour et Thomas, 1994; Courcelle et Olariu, 2000; Oum, 2005; Bui-Xuan et al., 2011; Nešetřil et de Mendez, 2012; Gajarský et al., 2013; Jeong et al., 2015; Wollan, 2015). Il pourrait être intéressant d'en exploiter certaines pour proposer de nouvelles classes polynomiales et de nouvelles méthodes de résolution. Enfin, compte tenu de l'intérêt pratique démontré par ces méthodes dans le cas du problème de décision CSP et du problème d'optimisation associé, il semble pertinent de vouloir étendre ces travaux à des problèmes encore plus difficiles comme le comptage ou la compilation.

Deuxième partie

Curriculum vitae

Notice individuelle

Cyril TERRIOUX
Nationalité française

Maître de Conférences (Classe normale - échelon 6, recruté le 1er octobre 2003, titularisé le 1er octobre 2004)
Aix-Marseille Université - Faculté des Sciences
Chercheur au Laboratoire des Sciences de l'Information et des Systèmes (LSIS - UMR CNRS 7296)
Titulaire de la PEDR de 2005 à 2009 et de la PES de 2009 à 2013
Section CNU 27

Adresse professionnelle :
Domaine universitaire de Saint Jérôme
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20

☎ 04 91 28 89 91 FAX 04 91 28 83 34
✉ cyril.terrioux@univ-amu.fr / cyril.terrioux@lsis.org
http://www.lsis.org/terriouxc

Diplômes obtenus

2002 **Doctorat en Informatique** (Mention Très Honorable) Université d'Aix-Marseille I
≪ *Approches structurelles et coopératives pour la résolution des problèmes de satisfaction de contraintes* ≫
Thèse soutenue le 10 décembre 2002 devant le jury composé de Messieurs Boi FALTINGS (rapporteur), Philippe JÉGOU (directeur de thèse), Thomas SCHIEX (rapporteur), Pierre SIEGEL (président) et Gérard VERFAILLIE (examinateur)

1999 **DEA Informatique** (Mention Bien) Université d'Aix-Marseille I

1998 **Maîtrise d'Informatique** (Mention Très Bien, major) Université d'Aix-Marseille I

1997 **Licence d'Informatique** (Mention Très Bien, major) Université d'Aix-Marseille I

1996 **DEUG MIAS** (Mention Bien) Université d'Aix-Marseille I

1994 **Baccalauréat série C** (Mention Bien).

Fonctions occupées

- Maître de conférences à l'université d'Aix-Marseille III d'octobre 2003 à décembre 2011, puis à Aix-Marseille Université depuis janvier 2012.
- Attaché temporaire d'enseignement et de recherche (à mi-temps) à l'université d'Aix-Marseille III d'octobre 2002 à août 2003.
- Allocataire de recherche à l'université d'Aix-Marseille I d'octobre 1999 à septembre 2002.
- Moniteur à l'université d'Aix-Marseille III d'octobre 1999 à septembre 2002.

Activités liées à la recherche

Sommaire

Mots-clés

Intelligence artificielle, Programmation par contraintes, Problèmes de satisfaction de contraintes, Optimisation combinatoire sous contraintes, Théorie des graphes et des hypergraphes, Classes polynomiales, Décomposition arborescente, Résolution.

Faits marquants

- En 2008, le solveur ToulBar2/BTD (Sánchez et al., 2008) a remporté la compétition internationale CPAI 2008 dans la catégorie « instances Max-CSP exprimées par des contraintes n-aires en extension » (voir le site de la compétition à l'adresse `http://www.cril.univ-artois.fr/CPAI08/results/results.php?idev=16` pour plus de détails). Ce solveur repose notamment sur des travaux développés autour de la méthode BTD dans le cadre du projet ANR STAL-DEC-OPT ainsi que sur des méthodes de propagation valuée.
- Best Technical Paper (Cooper et al., 2014) à la conférence internationale CP 2014 (International Conference on Principles and Practice of Constraint Programming).

Participation à des projets

J'ai participé à deux projets non thématiques soutenus par l'ANR :

- Le projet STAL-DEC-OPT (pour *Stratégies et algorithmes pour la décomposition et la résolution de problèmes d'optimisation sous contraintes*) regroupant des chercheurs de l'INRA (Toulouse), du LIFO (Orléans) et du LSIS s'est déroulé de décembre 2005 à décembre 2008. Il concernait la résolution de problèmes d'optimisation et de dénombrement sous contraintes via la définition de nouvelles méthodes hybridant décomposition et propagation de contraintes (filtrage par cohérence locale).
- Le projet TUPLES (pour *Tractability for Understanding and Pushing forward the Limits of Efficient Solvers*) regroupant des chercheurs du CRIL (Lens), du GREYC (Caen), de l'IRIT (Toulouse) et du LSIS s'est déroulé d'octobre 2010 à avril 2015. Son principal objectif était de repousser significativement les limites des solveurs CSP et SAT les plus efficaces à l'aide des classes polynomiales des problèmes CSP et SAT. Dans un premier temps, il s'agissait notamment de mieux appréhender le comportement des solveurs et d'expliquer leur efficacité pratique à l'aide des classes polynomiales existantes ou en définissant de nouvelles classes polynomiales. Dans un second temps, il était question de réduire le fossé entre théorie et pratique en exploitant les classes polynomiales pour définir de nouveaux solveurs.
  J'avais la responsabilité de coordonner une des six tâches du projet. La finalité de cette tâche était de fournir des arguments reposant sur les classes polynomiales pour expliquer la remarquable efficacité pratique des solveurs CSP et SAT.

Encadrements

Les différents encadrements listés ci-dessous concernent essentiellement l'étude des classes polynomiales et l'étude des méthodes structurelles et leurs applications pour la résolution de problèmes de décision et d'optimisation à base de contraintes.

Encadrements de thèse

- D'octobre 2004 à décembre 2007, j'ai coencadré à 50 % avec Philippe Jégou la thèse de Samba Ndojh Ndiaye (allocataire-moniteur). Cette thèse est intitulée « *Calcul et exploitation de recouvrements acycliques pour la résolution de (V)CSP* ». Samba Ndojh Ndiaye est, depuis la rentrée 2008, Maître de Conférences à l'université Claude Bernard Lyon 1 et effectue sa recherche au Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS).
  Publications associées : (Jégou et al., 2007d, 2005a, 2006a, 2007a,b, 2008a, 2006b,c; Ndiaye et Terrioux, 2007b; Jégou et al., 2005b, 2006d)
- À partir de janvier 2005, j'ai coencadré à 50 % avec Philippe Jégou la thèse de Karim Boutaleb. Le sujet de cette thèse portait sur l'exploitation et l'extension des ROBDD (Bryant, 1986, 1992) pour la gestion de (no)goods valués. Après des débuts encourageants ayant donné lieu à trois publications internationales (une conférence internationale (Boutaleb et al., 2006a) et deux workshops (Boutaleb et al., 2006b,c) dont un workshop AAAI), le doctorant a dû mettre un terme à sa thèse en cours de deuxième année pour des raisons d'ordre matériel (le financement marocain dont il devait initialement bénéficier n'ayant pas été obtenu).
  Publications associées : (Boutaleb et al., 2006a,b,c)
- D'octobre 2011 à décembre 2014, j'ai coencadré à 50 % avec Philippe Jégou la thèse d'Achref El Mouelhi. Cette thèse s'est déroulée dans le cadre de l'ANR TUPLES et s'intitule « *Classes polynomiales pour CSP : de la théorie à la pratique* ». Achref El Mouelhi est actuellement qualifié aux fonctions de maître de conférences et occupe un poste d'ATER.
  Publications associées : (El Mouelhi et al., 2014a, 2012a, 2013a,b,d; Cooper et al., 2014; El Mouelhi et al., 2014b,d, 2012b, 2013c, 2014c)
- Depuis octobre 2014, je coencadre à 50 % avec Philippe Jégou la thèse d'Hanan Kanso (allocataire). Cette thèse porte sur l'exploitation des décompositions de graphes et des techniques de redémarrage pour la résolution des problèmes de satisfaction de contraintes.
  Publications associées : (Jégou et al., 2015b,a, 2016; Jégou et al., 2016)

Encadrements de mémoire de master

- 2005 : Sébastien Martin ≪ *Calcul d'un ensemble coupe-cycle pour la méthode Coupe-Cycle* ≫
- 2006 : Nabil Babouri ≪ *Recherche énumérative bornée pour la coloration de graphes* ≫
- 2007 : Jean-Michel Jennequin ≪ *Recherche énumérative bornée pour la coloration de graphes* ≫
- 2008 : Cédric Pinto ≪ *Recherche de compromis de qualité entre coupe-cycle et décomposition arborescente* ≫
  Publications associées : (Pinto et Terrioux, 2008, 2009a,b)
- 2011 : Achref El Mouelhi ≪ *Classes Polynomiales Hybrides et Résolution de CSP* ≫ - Co-encadrement à 50% avec Philippe Jégou
- 2011 : Hamza Hadfi ≪ *Résolution de problèmes de planification par décomposition d'un CSP associé* ≫ - Co-encadrement à 50% avec Cyril Pain-Barre
- 2012 : Julien Lacroix ≪ *Redémarrages et (Re)décompositions pour la résolution de CSP* ≫ - Co-encadrement à 50% avec Philippe Jégou
- 2012 : Jean Elisée Ngimut Kigwe ≪ *Résolution des problèmes de planification par décomposition d'un CSP associé* ≫ - Co-encadrement à 50% avec Cyril Pain-Barre

Logiciels développés

- Outil *peo* : cet outil écrit en C est actuellement intégré dans le solver Toulbar2. Il permet de calculer des ordres d'élimination parfaits selon différentes méthodes. Ces ordres d'élimination parfaits peuvent être ensuite exploités dans Toulbar2/BTD pour produire des décompositions arborescentes.
- Bibliothèque Graph : cette bibliothèque a pour vocation de manipuler des graphes et des hypergraphes. Elle permet notamment de calculer des décompositions arborescentes selon différentes méthodes.
- Bibliothèque CSP : cette bibliothèque contient plusieurs outils développés en C++. Ces outils concernent :
  - la résolution de CSP : plusieurs algorithmes classiques de résolution ont été implémentés (dont notamment BT, FC, RFL et MAC) auxquels s'ajoutent les différents algorithmes proposés comme BTD. Différentes techniques permettant généralement de rendre la résolution de CSP plus efficaces ont également été mises en œuvre. En particulier, on peut citer une dizaine d'algorithmes de filtrage, différentes heuristiques de choix de variables ou de valeurs et plusieurs politiques de redémarrage.
  - la détection de classes polynomiales : actuellement, les algorithmes de reconnaissance de 15 classes polynomiales de CSP binaires ou d'arité quelconque ont été implémentés. Ils ont permis de mettre en évidence l'existence d'instances appartenant à ces classes polynomiales parmi les instances classiquement employées pour l'évaluation et la comparaison de solveurs.
  - la fusion de valeurs : cet outil permet de réaliser la fusion des valeurs selon quatre propriétés différentes, et ainsi d'évaluer l'intérêt pratique de ce type de fusion.
  - l'analyse de la résolution : afin de fournir des explications à l'efficacité pratique des solveurs, différents modules leur ont été adjoints permettant notamment d'analyser l'état courant du problème durant sa résolution. L'un de ces modules consiste, par exemple, à détecter certaines classes polynomiales (en relation avec la notion de backdoor (Williams et al., 2003)).

Ces outils ont pour certains été developpés dans le cadre des projets ANR STAL-DEC-OPT et TUPLES. Par ailleurs, les bibliothèques Graph et CSP sont en constante évolution (optimisation du code, implémentation de nouvelles méthodes en fonction de l'avancée des recherches, ...).

Évaluation scientifique

Participation à des comités de programme

- Conférences internationales :

  - AAAI Conference on Artificial Intelligence (AAAI) – 2008
  - International Joint Conference on Artificial Intelligence (IJCAI) – 2009, 2015, 2016
- Conférences nationales :
  - Journées Francophones de Programmation par Contraintes (JFPC) – 2009, 2010, 2015, 2016
  - Journées nationales de la résolution pratique des problèmes NP-Complets (JNPC) – 2003, 2004
  - Reconnaissance des Formes et Intelligence Artificielle (RFIA) – 2008

Relectures

- Revues internationales :
  - AI Communications – 2013
  - Artificial Intelligence Journal – 2005
  - Studia Informatica Universalis – 2005
  - RAIRO Operations Research – 2014
- Conférences internationales :
  - AAAI Conference on Artificial Intelligence (AAAI) – 2007, 2016
  - ACM Symposium on Applied Computing (SAC), Special Track on Artificial Intelligence Computational Logic and Image Analysis – 2004
  - European Conference on Artificial Intelligence (ECAI) – 2008
  - IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Special Track on SAT and CSP Technologies – 2013
  - International Joint Conference on Artificial Intelligence (IJCAI) – 2003
  - International Conference on Principles and Practice of Constraint Programming (CP) – 2006, 2014, 2015, 2016.

Organisation de conférences ou de workshop

- Co-organisateur du workshop SOFT 2006 (8th International Workshop on Preferences and Soft Constraints) organisé au sein de la conférence internationale CP 2006.
- Membre du Comité d'organisation de la conférence internationale TABLEAUX 2007 (Automated Reasoning with Analytic Tableaux and Related Methods).
- Membre du Comité d'organisation des conférences nationales JFPC 2013 (Journées Francophones de Programmation par Contraintes) et JIAF 2013 (Journées de l'Intelligence Artificielle Fondamentale).

Liste des publications

Pour les revues et conférences internationales, le nombre de publications de rang A*, A et B selon le classement de référence CORE est précisé.

Revues internationales avec comité de lecture (4 dont 2 A* et 2 A)

Philippe Jégou et Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. *Constraints*, 20(4) :383–413, 2015.

Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, et Bruno Zanuttini. Broken Triangles : From Value Merging to a Tractable Class of General-Arity Constraint Satisfaction Problems. *Artificial Intelligence*, 234 :196–218, 2016.

Philippe Jégou et Cyril Terrioux. Combining Restarts, Nogoods and Bag-Connected Decompositions for Solving CSPs. *Constraints*, 2016. À paraître.

Revue nationale avec comité de lecture (1)

Philippe Jégou et Cyril Terrioux. Recherche arborescente bornée pour la résolution de CSP valués. *Journal électronique d'intelligence artificielle (JEDAI)*, 3(28), 2004.

Chapitres d'ouvrage (3)

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic Heuristics for Branch and Bound on Tree-Decomposition of Weighted CSPs. In *Trends in Constraint Programming*, chapter 20, pages 317–332. ISTE, 2007.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Different Classes of Graphs to Represent Microstructures for CSPs. In *Graph Structures for Knowledge Representation and Reasoning, LNAI 8323*, pages 21–38. Springer, 2014.

Philippe Jégou et Cyril Terrioux. Structural Consistency : A New Filtering Approach for Constraint Networks. In *Graph Structures for Knowledge Representation and Reasoning, LNAI 8323*, pages 74–91. Springer, 2014.

Conférences internationales avec comité de lecture (32 dont 4 A*, 12 A et 16 B)

Cyril Terrioux. Cooperative Search and Nogood Recording. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 260–265, 2001.

Cyril Terrioux et Philippe Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 709–723, 2003.

Philippe Jégou et Cyril Terrioux. Decomposition and Good Recording. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 196–200, 2004.

Philippe Jégou et Cyril Terrioux. A time-space trade-off for constraint networks decomposition. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 234–239, 2004.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 777–781, 2005.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. An extension of complexity bounds and dynamic heuristics for tree-decompositions of CSP. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 741–745, 2006.

Karim Boutaleb, Philippe Jégou, et Cyril Terrioux. (No)good Recording and ROBDDs for Solving Structured (V)CSPs. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 297–304, 2006.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 112–117, 2007.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic Management of Heuristics for Solving Structured CSPs. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 364–378, 2007.

Samba Ndojh Ndiaye, Philippe Jégou, et Cyril Terrioux. Extending to Soft and Preference Constraints a Framework for Solving Efficiently Structured Problems. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 299–306, 2008.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. A New Evaluation of Forward Checking and its Consequences on Efficiency of Tools for Decomposition of CSPs. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 486–490, 2008.

Cédric Pinto et Cyril Terrioux. A New Method for Computing Suitable Tree-decompositions with Respect to Structured CSP Solving. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 491–495, 2008.

Djamal Habet, Lionel Paris, et Cyril Terrioux. A Tree Decomposition Based Approach to Solve Structured SAT Instances. In *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 115–122, 2009.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Combined Strategies for Decomposition-based Methods for solving CSPs. In *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 184–192, 2009.

Cédric Pinto et Cyril Terrioux. A generalized Cyclic-Clustering Approach for Solving Structured CSPs. In *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 724–728, 2009.

Philippe Jégou et Cyril Terrioux. A New Filtering Based on Decomposition of Constraint Sub-Networks. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 263–270, 2010.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. On the Efficiency of Backtracking Algorithms for Binary Constraint Satisfaction Problems. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2012.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Microstructures for CSPs with Constraints of Arbitrary Arity. In *10th Symposium on Abstraction, Reformulation, and Approximation (SARA)*, 2013.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 947–954, 2013.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. Some New Tractable Classes of CSPs and their Relations with Backtracking Algorithms. In *Proceedings of the 10th International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (CPAIOR)*, pages 61–76, 2013.

Philippe Jégou et Cyril Terrioux. Bag-Connected Tree-Width : A New Parameter for Graph Decomposition. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2014.

Philippe Jégou et Cyril Terrioux. Combining Restarts, Nogoods and Decompositions for Solving CSPs. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, pages 465–470, 2014.

Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, et Bruno Zanuttini. On Broken Triangles. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 9–24, 2014. Best technical paper.

Philippe Jégou et Cyril Terrioux. Tree-Decompositions with Connected Clusters for Solving Constraint Networks. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 407–423, 2014.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Hidden Tractable Classes : from Theory to Practice. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2014.

Martin C. Cooper, Philippe Jégou, et Cyril Terrioux. A Microstructure-based Family of Tractable Classes for CSPs. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, pages 74–88, 2015.

Philippe Jégou et Cyril Terrioux. The Extendable-Triple Property : A New CSP Tractable Class beyond BTP. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pages 3746–3754, 2015.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. An Algorithmic Framework for Decomposing Constraint Networks. In *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1–8, 2015.

Martin C. Cooper, Achref El Mouelhi, et Cyril Terrioux. Extending Broken Triangles and Enhanced Value-merging. In *Proceedings of the 22nd International Conference on Principle and Practice of Constraint Programming (CP)*, pages 173–188, 2016.

Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, et Bruno Zanuttini. On Broken Triangles. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4135–4139, 2016.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. Towards a Dynamic Decomposition of CSPs with Separators of Bounded Size. In *Proceedings of the 22nd International Conference on Principle and Practice of Constraint Programming (CP)*, pages 298–315, 2016.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. Improving Exact Solution Counting for Decomposition Methods. In *Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 327–334, 2016.

Workshops internationaux (13)

Karim Boutaleb, Philippe Jégou, et Cyril Terrioux. Optimizing the space to extend the tractability of (valued) structured CSP. In *Proceedings of the Annual Workshop of ERCIM on Constraint Solving and Constraint Logic Programming (CSCLP'06)*, pages 85–99, 2006.

Karim Boutaleb, Philippe Jégou, et Cyril Terrioux. Storing learnt (no)goods in ROBDDs for solving structured CSPs. In *Proceedings of the AAAI Workshop on Learning for Search*, pages 65–71, 2006.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Strategies and Heuristics for Exploiting Tree-decompositions of Constraint Networks. In *Proceedings of the Inference methods based on graphical structures of knowledge (WIGSK'06), ECAI workshop*, pages 13–18, 2006.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic heuristics for branch and bound search on tree-decomposition of weighted csps. In *Proceedings of the Eighth International Workshop on Preferences and Soft Constraints (Soft-2006)*, pages 63–77, 2006.

Samba Ndojh Ndiaye et Cyril Terrioux. A generic bounded backtracking framework for solving CSPs. In *Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming (CSCLP'07)*, pages 107–121, 2007.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Extending to Soft and Preference Constraints a Framework for Solving Efficiently Structured Problems. In *Proceedings of the 4th Multidisciplinary Workshop on Advances in Preference Handling (M-PREF 2008), AAAI workshop*, pages 61–66, 2008.

Martí Sánchez, Sylvain Bouveret, Simon de Givry, Federico Heras, Philippe Jégou, Javier Larrosa, Samba Ndojh Ndiaye, Emma Rollon, Thomas Schiex, Cyril Terrioux, Gérard Verfaillie, et Matthias Zytnicki. Max-CSP competition 2008 : toulbar2 solver description. In *Proceedings of the 3rd CSP Solver Competition, CP workshop*, pages 63–70, 2008.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Hypertree decomposition vs tree decomposition for solving constraint networks. In *International Workshop on Graph Decomposition : Theoretical, Algorithmic and Logical Aspect*, 2008.

Philippe Jégou et Cyril Terrioux. Structural Consistency : A New Filtering Approach for Constraint Networks. In *Proceedings of the 1st Workshop on Constraint Reasoning and Graphical Structures*, 2010.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Different Classes of Graphs to Represent Microstructures for CSPs. In *3rd International Workshop on Graph Structures for Knowledge Representation and Reasoning*, 2013.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. Some New Tractable Classes of CSPs and their Relations with Backtracking Algorithms. In *Fourth International Workshop on the Cross-Fertilization Between CSP and SAT*, 2014.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Hidden Tractable Classes. In *First Workshop on Bridging the Gap Between Theory and Practice in Constraint Solvers*, 2014.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. Exact Solution Counting for Artificial Intelligence based on Decomposition of Constraint Networks. In *Workshop on Constraint Programming and Artificial Intelligence*, 2016.

Conférences nationales avec comité de lecture (25)

Cyril Terrioux. Recherche Coopérative et Nogood Recording. In *Actes des Journées Francophones de Programmation en Logique et par Contraintes (JFPLC)*, pages 173–187, 2001.

Philippe Jégou et Cyril Terrioux. Recherche arborescente bornée. In *Actes des* $8^{\text{ème}}$ *Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC)*, pages 127–141, 2002.

Philippe Jégou et Cyril Terrioux. Recherche arborescente bornée pour la résolution de CSP valués. In *Actes des* $9^{\text{ème}}$ *Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC)*, pages 161–175, 2003.

Philippe Jégou et Cyril Terrioux. Décomposition et Good Recording pour le problème Max-CSP. In *Actes des* $10^{\text{ème}}$ *Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC)*, pages 219–234, 2004.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Sur la génération et l'exploitation de décompositions pour la résolution de réseaux de contraintes. In *Actes des* $1^{\text{ère}}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 149–158, 2005.

Philippe Jégou et Cyril Terrioux. Un compromis temps-espace pour la résolution de réseaux de contraintes par décomposition. In *Actes des* $1^{\text{ère}}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 159–168, 2005.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre. In *Actes des* $2^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 219–228, 2006.

Samba Ndojh Ndiaye et Cyril Terrioux. Un schéma générique d'algorithmes énumératifs avec (no)good recording pour la résolution bornée de CSP. In *Actes des* $3^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 209–218, 2007.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Recouvrement de problèmes par des hypergraphes acycliques : analyses théorique et expérimentale. In *Actes des* $3^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 271–280, 2007.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Complexité de Forward Checking et Hiérarchie des Décompositions de CSP Revisitées. In *Actes des* $4^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 153–163, 2008.

Cédric Pinto et Cyril Terrioux. Une généralisation de l'approche Cyclic-Clustering pour la résolution. In *Actes des* $5^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 5–14, 2009.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Stratégies hybrides pour des décompositions optimales et efficaces. In *Actes des* $5^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 35–44, 2009.

Djamal Habet, Lionel Paris, et Cyril Terrioux. Une approche basée sur la décomposition arborescente pour la résolution d'instances SAT structurées. In *Actes des* $5^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 85–94, 2009.

Philippe Jégou et Cyril Terrioux. Une nouvelle technique de filtrage basée sur la décomposition de sous-réseaux de contraintes. In *Actes des* $6^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 157–166, 2010.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. Sur la complexité des algorithmes de backtracking et quelques nouvelles classes polynomiales pour CSP. In *Actes des* $8^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 160–169, 2012.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Sur une classe polynomiale hybride pour les CSP d'arité quelconque. In *Actes des* $9^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 237–247, 2013.

Philippe Jégou et Cyril Terrioux. Combiner les Redémarrages, Nogoods et Décompositions pour la Résolution de CSP. In *Actes des* $10^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 19–28, 2014.

Philippe Jégou et Cyril Terrioux. Un nouveau paramètre de graphes pour la résolution de CSP par décomposition. In *Actes des* $10^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 77–88, 2014.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Microstructures pour CSP d'arité quelconque. In *Actes des* $10^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 193–202, 2014.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Classes Polynomiales Cachées : de la théorie à la pratique. In *Actes des* $11^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 16–17, 2015.

Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, et Bruno Zanuttini. Autour des Triangles Cassés. In *Actes des* $11^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 57–58, 2015.

Martin C. Cooper, Philippe Jégou, et Cyril Terrioux. Une famille de classes polynomiales de CSP basée sur la microstructure. In *Actes des* $11^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 59–68, 2015.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. De nouvelles approches pour la décomposition de réseaux de contraintes. In *Actes des* $11^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 140–149, 2015.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. Vers une décomposition dynamique des réseaux de contraintes. In *Actes des* $12^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 123–132, 2016.

Martin C. Cooper, Achref El Mouelhi, et Cyril Terrioux. Les triangles cassés, encore et encore. In *Actes des* $12^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 133–141, 2016.

Thèse

Cyril Terrioux. *Approches structurelles et coopératives pour la résolution des problèmes de satisfaction de contraintes*. PhD thesis, Université de Provence, Décembre 2002.

Rapports de recherche (12)

Philippe Jégou et Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. Rapport de recherche LSIS.2002.005, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2002. 27 pages.

Cyril Terrioux. Cooperative search vs classical algorithms. Rapport de recherche LSIS.2002.006, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2002. 12 pages.

Cyril Terrioux. Exchanging nogoods : an efficient cooperative search for solving constraint satisfaction problems. Rapport de recherche LSIS.2003.002, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2003. 26 pages.

Philippe Jégou et Cyril Terrioux. Recherche arborescente bornée pour la résolution de CSP valués. Rapport de recherche LSIS.2003.003, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2003. 13 pages.

Philippe Jégou et Cyril Terrioux. A time-space trade-off for constraint networks decomposition. Rapport de recherche LSIS.2004.005, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2004. 12 pages.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Computing and exploiting tree-decompositions for (Max-)CSP. Rapport de recherche LSIS.RR.2005.005, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2005. 11 pages.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre. Rapport de recherche LSIS.RR.2006.004, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2006. 12 pages.

Cédric Pinto et Cyril Terrioux. A New Method for Computing Suitable Tree-decompositions with respect to Structured CSP Solving. Rapport de recherche LSIS.RR.2008.002, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2008.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. A new Evaluation of Forward Checking and its Consequences on Efficiency of Tools for Decomposition of CSPs. Rapport de recherche LSIS.RR.2008.003, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2008.

Djamal Habet, Lionel Paris, et Cyril Terrioux. A Tree Decomposition Based Approach to Solve Structured Satisfiability Instances. Rapport de recherche LSIS.RR.2009.001, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2009.

Cédric Pinto et Cyril Terrioux. A generalized Cyclic-Clustering Approach for Solving Structured CSPs. Rapport de recherche LSIS.RR.2009.004, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2009.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. A New Filtering Based on Decomposition of Constraint Sub-Networks. Rapport de recherche, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), 2010.

Activités liées à l'enseignement

J'enseigne depuis 1999 d'abord en qualité de moniteur, puis d'attaché temporaire d'enseignement et de recherche et enfin de maître de conférences. Tous mes enseignements se sont déroulés au sein de la Faculté des Sciences et Techniques de l'université d'Aix-Marseille III puis au sein de la Faculté des Sciences d'Aix-Marseille université suite à la fusion des trois universités d'Aix-Marseille. Ils visent un public scientifique varié (étudiants en informatique ou en sciences de la matière par exemple) du BAC+1 au BAC+5 avec une majorité d'intervention dans les filières informatiques (Licence et Master).

Liste des enseignements effectués

- *Algorithmique et programmation* – TD, TP — 1999 - 2001, 2003 - 2012
  Deug MIAS $1^{ère}$ année, L1 Mathématiques et Informatique
- *Introduction à l'informatique et à la programmation* – CM, TD, TP — 2012 - ...
  L1 Mathématiques et Informatique
- *Algorithmique avancée* – TD, TP — 1999 - 2001, 2002 - 2003, 2004 - 2012
  Deug MIAS $1^{ère}$ année, L1 Mathématiques et Informatique
- *Architecture des ordinateurs et des systèmes* – TD, TP — 2001 - 2003, 2004 - 2012
  Deug MIAS $2^{ème}$ année, L1 Mathématiques et Informatique
- *Introduction au langage Pascal* – CM, TP — 2003 - 2004
  DEUG SM $1^{ère}$ année
- *Programmation avancée en Pascal* – CM, TP — 2003 - 2004
  DEUG SM $2^{ème}$ année
- *Programmation en C* – TP — 2003 - 2004
  Licence professionnelle Systèmes Informatiques et Logiciels
- *Algorithmique des arbres et des graphes* – TD — 2004 - 2012
  L3 Informatique
- *Langage et automates* – TP — 2005 - 2008
  L2 Informatique
- *Intelligence artificielle* – CM, TD, TP — 2004 - ...
  L3 Informatique
- *Représentation des connaissances* – TD — 2003 - 2004
  Maîtrise Informatique
- *Complexité* – TD, TP — 2004 - ...
  M1 Informatique
- *Système d'exploitation* – CM, TP — 2003 - 2004
  DESS Informatique Appliquée à la chimie
- *Problèmes de satisfaction de contraintes valués* – CM, TD — 2002 - ...
  DEA MCAO, M2R Informatique Spécialité Sciences de l'Information et des Systèmes

En plus de ces enseignements présentiels, j'ai encadré plusieurs TER en M1 Informatique ainsi que plusieurs projets industriels dans le cadre du M2P Informatique spécialité Génie Logiciel.

Responsabilités pédagogiques

- Co-responsable du Master Sciences de l'Information et des Systèmes (Master SIS, http://www.lsis.org/master/ancien_site) de septembre 2008 à septembre 2012 (habilitation 2008-2011). Ce master était composé de quatre spécialités à finalité professionnelle et d'une spécialité à finalité recherche et porté par les trois universités d'Aix-Marseille, l'université du Sud Toulon Var et l'ENSAM d'Aix-en-Provence. En tant que co-responsable de la mention, j'ai notamment participé au montage du dossier d'habilitation.
- Directeur des études, de septembre 2008 à septembre 2012 du Master SIS.
- Responsable de la spécialité Informatique du Master SIS de septembre 2008 à septembre 2012.
- Responsable de la première année des spécialités Génie Informatique et Systèmes d'Information du Master SIS de septembre 2004 à août 2008.
- Enseignant référent en L1 Informatique de septembre 2013 à juin 2014.

Charges collectives et responsabilités

Responsabilités au sein de l'établissement

- Membre élu du Conseil Scientifique de l'université Paul Cézanne (Aix-Marseille 3) de février 2008 à décembre 2011.
- Membre de droit du conseil du département MIS (Mathématiques, Informatique et Systèmes) de juillet 2008 à décembre 2011 (au titre de ma responsabilité pédagogique du Master SIS). Ce département de la Faculté des Sciences et Techniques de l'université Paul Cézanne concernait environ 150 permanents (enseignants, enseignants-chercheurs ou chercheurs).
- Membre élu du conseil du département MIS de décembre 2004 à juillet 2008.
- Membre suppléant de la commission de spécialistes des sections 27 et 61 de l'université Paul Cézanne (Aix-Marseille 3) d'octobre 2004 à août 2008.

Gestion d'équipement scientifique

- Responsable de la gestion d'un cluster informatique dédié à la résolution de problèmes NP-complets et NP-difficiles depuis 2006 :
  Ce cluster est constitué d'un serveur, de 32 PC et de 32 serveurs-lames. Il a pour finalité de permettre aux chercheurs de l'équipe INCA du LSIS de réaliser les expérimentations nécessaires à la validation de leurs travaux. Il a notamment été utilisé pour mener à bien les parties expérimentales de 3 projets ANR (ANR STAL-DEC-OPT, ANR UNLOC et ANR TUPLES).

Gestion d'équipement pédagogique

- Responsable de la gestion de salles de TP informatique de 2002 à 2006 :
  Cet équipement pédagogique était composé d'un serveur et de 20 PC fonctionnant au choix sous Window ou Linux et était principalement utilisé par les étudiants du DESS ISII (Ingénierie des Systèmes Informatisés Industriels) puis du master SIS.

Troisième partie

Sélection de publications

Liste des publications sélectionnées

Sommaire

Classes polynomiales

Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, et Bruno Zanuttini. On Broken Triangles. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 9–24, 2014. Best technical paper.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. *Constraints*, 20(4) :383–413, 2015.

Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, et Bruno Zanuttini. Broken Triangles : From Value Merging to a Tractable Class of General-Arity Constraint Satisfaction Problems. *Artificial Intelligence*, 234 :196–218, 2016.

Résolution de CSP par des approches structurelles

Philippe Jégou et Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 112–117, 2007.

Philippe Jégou et Cyril Terrioux. Combining Restarts, Nogoods and Decompositions for Solving CSPs. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, pages 465–470, 2014.

Résolution de CSP valués par des approches structurelles

Cyril Terrioux et Philippe Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 709–723, 2003.

Philippe Jégou et Cyril Terrioux. Decomposition and Good Recording. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 196–200, 2004.

On Broken Triangles*

Martin C. Cooper[1], Achref El Mouelhi[2], Cyril Terrioux[2], and Bruno Zanuttini[3]

[1] IRIT, University of Toulouse III, 31062 Toulouse, France
cooper@irit.fr
[2] LSIS, Aix-Marseille University, 13397 Marseille, France
{achref.elmouelhi, cyril.terrioux}@lsis.org
[3] GREYC, University of Caen Basse-Normandie, 14032 Caen, France
bruno.zanuttini@unicaen.fr

Abstract. A binary CSP instance satisfying the broken-triangle property (BTP) can be solved in polynomial time. Unfortunately, in practice, few instances satisfy the BTP. We show that a local version of the BTP allows the merging of domain values in arbitrary instances of binary CSP, thus providing a novel polynomial-time reduction operation. Extensive experimental trials on benchmark instances demonstrate a significant decrease in instance size for certain classes of problems. We show that BTP-merging can be generalised to instances with constraints of arbitrary arity and we investigate the theoretical relationship with resolution in SAT. A directional version of the general-arity BTP then allows us to extend the BTP tractable class previously defined only for binary CSP.

1 Introduction

At first sight one could assume that the discipline of constraint programming has come of age. On the one hand, efficient solvers are regularly used to solve real-world problems in diverse application domains while, on the other hand, a rich theory has been developed concerning, among other things, global constraints, tractable classes, reduction operations and symmetry. However, there often remains a large gap between theory and practice which is perhaps most evident when we look at the large number of deep results concerning tractable classes which have yet to find any practical application. The research reported in this paper is part of a long-term project to bridge the gap between theory and practice. Our aim is not only to develop new tools but also to explain why present tools work so well.

Most research on tractable classes has been based on classes defined by placing restrictions either on the types of constraints or on the constraint hypergraph whose vertices are the variables and whose hyper-edges are the constraint scopes. Another way of defining classes of binary CSP instances consists in imposing conditions on the microstructure, a graph whose vertices are the possible variable-value assignments with an edge linking each pair of compatible assignments [9,12]. If each vertex of the microstructure, corresponding to a

* supported by ANR Project ANR-10-BLAN-0210.

variable-value assignment $\langle x, a\rangle$, is labelled by the variable x, then this so-called coloured microstructure retains all information from the original instance. The broken-triangle property (BTP) is a simple local condition on the coloured microstructure which defines a tractable class of binary CSP [5]. Inspired by the BTP, investigation of other forbidden patterns in the coloured microstructure has led to the discovery of new tractable classes [1,4,6,8] as well as new reduction operations based on variable elimination [2].

For simplicity of presentation we use two different representations of constraint satisfaction problems. In the binary case, our notation is fairly standard, whereas in the general-arity case we use a notation close to the representation of SAT instances. This is for presentation only, though, and our algorithms do *not* need instances to be represented in this manner.

Definition 1. *A* binary CSP instance *I consists of*

- *a set X of n variables,*
- *a domain $\mathcal{D}(x)$ of possible values for each variable $x \in X$,*
- *a relation $R_{xy} \subseteq \mathcal{D}(x) \times \mathcal{D}(y)$, for each pair of distinct variables $x, y \in X$, which consists of the set of compatible pairs of values (a, b) for variables (x, y).*

A partial solution *to I on $Y = \{y_1, \ldots, y_r\} \subseteq X$ is a set $\{\langle y_1, a_1\rangle, \ldots, \langle y_r, a_r\rangle\}$ such that $\forall i, j \in [1, r]$, $(a_i, a_j) \in R_{y_i y_j}$. A* solution *to I is a partial solution on X.*

For simplicity of presentation, Definition 1 assumes that there is exactly one constraint relation for each pair of variables. The number of constraints e is the number of pairs of variables x, y such that $R_{xy} \neq \mathcal{D}(x) \times \mathcal{D}(y)$. An instance I is *arc consistent* if for each pair of distinct variables $x, y \in X$, for each value $a \in \mathcal{D}(x)$, there is a value $b \in \mathcal{D}(y)$ such that $(a, b) \in R_{xy}$.

In our representation of general-arity CSP instances, we require the notion of *tuple* which is simply a set of variable-value assignments. For example, in the binary case, the tuple $\{\langle x, a\rangle, \langle y, b\rangle\}$ is *compatible* if $(a, b) \in R_{xy}$ and *incompatible* otherwise.

Definition 2. *A* (general-arity) CSP instance *I consists of*

- *a set X of n variables,*
- *a domain $\mathcal{D}(x)$ of possible values for each variable $x \in X$,*
- *a set NoGoods(I) consisting of incompatible tuples.*

A partial solution *to I on $Y = \{y_1, \ldots, y_r\} \subseteq X$ is a tuple $t = \{\langle y_1, a_1\rangle, \ldots, \langle y_r, a_r\rangle\}$ such that no subset of t belongs to NoGoods(I). A* solution *is a partial solution on X.*

2 Value merging in binary CSP based on the BTP

In this section we consider a method, based on the BTP, for reducing domain size while preserving satisfiability. Instead of eliminating a value, as in classic reduction operations such as arc consistency or neighbourhood substitution,

we merge two values. We show that the absence of broken-triangles [5] on two values for a variable x in a binary CSP instance allows us to merge these two values in the domain of x while preserving satisfiability. This rule generalises the notion of virtual interchangeability [11] as well as neighbourhood substitution [10].

It is known that if for a given variable x in an arc-consistent binary CSP instance I, the set of (in)compatibilities (known as a broken-triangle) shown in Figure 1 occurs for no two values $a, b \in \mathcal{D}(x)$ and no two assignments to two other variables, then the variable x can be eliminated from I without changing the satisfiability of I [5,2]. In figures, each bullet represents a variable-value assignment, assignments to the same variable are grouped together within the same oval and compatible (incompatible) pairs of assignments are linked by solid (broken) lines. Even when this variable-elimination rule cannot be applied, it may be the case that for a given pair of values $a, b \in \mathcal{D}(x)$, no broken triangle occurs. We will show that if this is the case, then we can perform a domain-reduction operation which consists in merging the values a and b.

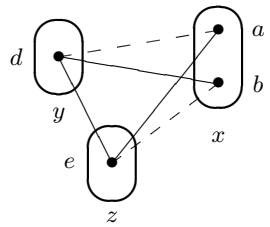


Fig. 1. A broken triangle on two values a, b for a given variable x.

Definition 3. Merging *values $a, b \in \mathcal{D}(x)$ in a binary CSP consists in replacing a, b in $\mathcal{D}(x)$ by a new value c which is compatible with all variable-value assignments compatible with at least one of the assignments $\langle x, a\rangle$ or $\langle x, b\rangle$. A* value-merging condition *is a polytime-computable property $P(x, a, b)$ of assignments $\langle x, a\rangle$, $\langle x, b\rangle$ in a binary CSP instance I such that when $P(x, a, b)$ holds, the instance I' obtained from I by merging $a, b \in \mathcal{D}(x)$ is satisfiable if and only if I is satisfiable.*

We now formally define the value-merging condition based on the BTP.

Definition 4. *A* broken triangle *on the pair of variable-value assignments $a, b \in \mathcal{D}(x)$ consists of a pair of assignments $d \in \mathcal{D}(y)$, $e \in \mathcal{D}(z)$ to distinct variables $y, z \in X \setminus \{x\}$ such that $(a, d) \notin R_{xy}$, $(b, d) \in R_{xy}$, $(a, e) \in R_{xz}$, $(b, e) \notin R_{xz}$ and $(d, e) \in R_{yz}$. The pair of values $a, b \in \mathcal{D}(x)$ is* BT-free *if there is no broken triangle on a, b.*

Proposition 1. *In a binary CSP instance, being BT-free is a value-merging condition. Furthermore, given a solution to the instance resulting from the merging of two values, we can find a solution to the original instance in linear time.*

Proof. Let I be the original instance and I' the new instance in which a,b have been merged into a new value c. Clearly, if I is satisfiable then so is I'. It suffices to show that if I' has a solution s which assigns c to x, then I has a solution. Let s_a, s_b be identical to s except that s_a assigns a to x and s_b assigns b to x. Suppose that neither s_a nor s_b are solutions to I. Then, there are variables $y, z \in X \setminus \{x\}$ such that $\langle a, s(y)\rangle \notin R_{xy}$ and $\langle b, s(z)\rangle \notin R_{xz}$. By definition of the merging of a, b to produce c, and since s is a solution to I' containing $\langle x, c\rangle$, we must have $(b, s(y)) \in R_{xy}$ and $(a, s(z)) \in R_{xz}$. Finally, $(s(y), s(z)) \in R_{yz}$ since s is a solution to I'. Hence, $\langle y, s(y)\rangle$, $\langle z, s(z)\rangle$, $\langle x, a\rangle$, $\langle x, b\rangle$ forms a broken-triangle, which contradicts our assumption. Hence, the absence of broken triangles on assignments $\langle x, a\rangle$, $\langle x, b\rangle$ allows us to merge these assignments while preserving satisfiability.

Reconstructing a solution to I from a solution s to I' simply requires checking which of s_a or s_b is a solution to I. □

We can see that the BTP-merging rule, given by Proposition 1, generalises neighbourhood substitution [10]: if b is neighbourhood substitutable by a, then no broken triangle occurs on a, b and merging a and b produces a CSP instance which is identical (except for the renaming of the value a as c) to the instance obtained by simply eliminating b from $\mathcal{D}(x)$. BTP-merging also generalises the merging rule proposed by Likitvivatanavong and Yap [11]. The basic idea behind their rule is that if the two assignments $\langle x, a\rangle$, $\langle x, b\rangle$ have identical compatibilities with all assignments to all other variables except concerning at most one other variable, then we can merge a and b. This is clearly subsumed by BTP-merging.

The BTP-merging operation is not only satisfiability-preserving but, from Proposition 1, we know that we can also reconstruct a solution in polynomial time to the original instance I from a solution to an instance I^m to which we have applied a sequence of merging operations until convergence. It is known that for the weaker operation of neighbourhood substitutability, all solutions to the original instance can be generated in $O(N(de + n^2))$ time, where N is the number of solutions to the original instance, n is the number of variables, d the maximum domain size and e the number of constraints [3]. We now show that a similar result also holds for the more general rule of BTP-merging.

Proposition 2. *Let I be a binary CSP instance and suppose that we are given the set of all solutions to the instance I^m obtained after applying a sequence of BTP-merging operations. All N solutions to I can then be determined in $O(Nn^2d)$ time.*

Proof. Let I' be the CSP instance which results after performing a single BTP-merging operation of values $a, b \in \mathcal{D}(x)$ in I. As we saw in the proof of Proposition 1, given the set of solutions $sol(I')$ to I' we can generate the set of solutions to I by testing for each $s \in Sol(I')$ whether s_a or s_b (or both) are solutions to I. This requires $O(n)$ time per solution to I, since there are at most $n - 1$ constraints to be tested involving the variable x, and at least one of s_a or s_b is a solution to I.

The total number of BTP-merging operations performed to transform I into I^m is at most $n(d-1)$. Therefore, the total time to generate all N solutions to I from the set of solutions to I^m is $O(Nn^2d)$. □

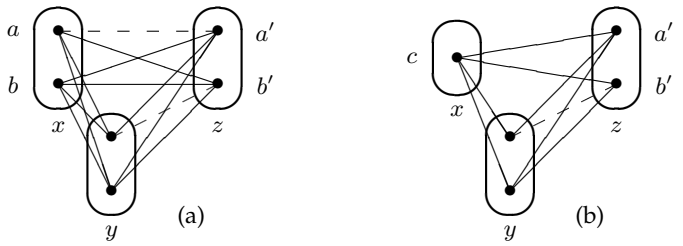


Fig. 2. (a) A broken triangle exists on values a', b' at variable z. (b) After BTP-merging of values a and b in $\mathcal{D}(x)$, this broken triangle has disappeared.

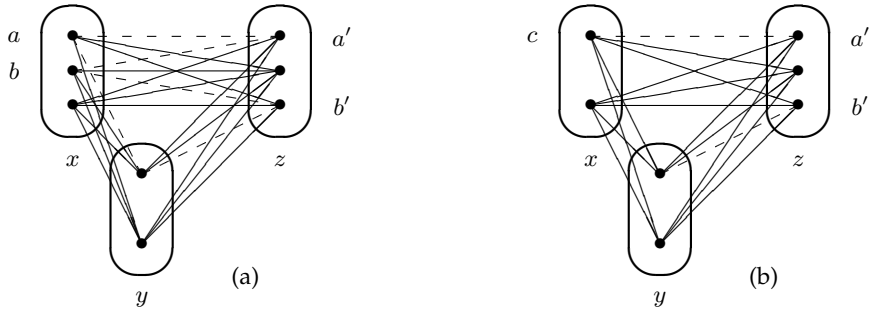


Fig. 3. (a) This instance contains no broken triangle. (b) After BTP-merging of values a and b in $\mathcal{D}(x)$, a broken triangle has appeared on values $a', b' \in \mathcal{D}(z)$.

The weaker operation of neighbourhood substitution has the property that two different convergent sequences of eliminations by neighbourhood substitution necessarily produce isomorphic instances I_1^m, I_2^m [3] . This is not the case for BTP-merging. Firstly, and perhaps rather surprisingly, BTP-merging can have as a side-effect to eliminate broken triangles. This is illustrated in the 3-variable instance shown in Figure 2. The instance in Figure 2(a) contains a broken triangle on values a', b' for variable z, but after BTP-merging of values $a, b \in \mathcal{D}(x)$ into a new value c, as shown in Figure 2(b), there are no broken triangles in the instance. Secondly, BTP-merging of two values in $\mathcal{D}(x)$ can introduce a broken triangle on a variable $z \neq x$, as illustrated in Figure 3. The instance in Figure 3(a) contains no broken triangle, but after the BTP-merging of $a, b \in \mathcal{D}(x)$ into a new value c, a broken triangle has been created on values $a', b' \in \mathcal{D}(z)$.

3 Experimental trials

To test the utility of BTP-merging we performed extensive experimental trials on benchmark instances from the International CP Competition[4]. For each instance not including global constraints, we performed BTP-mergings until convergence with a time-out of one hour. In total, we obtained results for 2,547 instances out of 3,811 benchmark instances. In the other instances the search for all BTP-mergings did not terminate within a time-out of one hour.

domain	no. instances	no. values	no. values deleted	%age deleted
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	6400	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%
others	1919	1,396	28	0.02%

Table 1. Results of experiments on CSP benchmark problems.

All instances from the benchmark-domain `hanoi` satisfy the broken-triangle property and BTP-merging reduced all variable domains to singletons. After establishing arc consistency, 38 instances from diverse benchmark-domains satisfy the BTP, including all instances from the benchmark-domain `domino`. We did not count those instances for which arc consistency detects inconsistency by producing a trivial instance with empty variable domains (and which trivially satisfies the BTP). In all instances from the `pigeons` benchmark-domain with a suffix `-ord`, BTP-merging again reduced all domains to singletons. This is because BTP-merging can eliminate broken triangles, as pointed out in Section 2, and hence can render an instance BTP even though initially it was not BTP. The same phenomenon occurred in a 680-variable instance from the benchmark-domain `rlfapGraphsMod` as well as the 3-variable instance `ogdPuzzle`.

Table 1 gives a summary of the results of the experimental trials. We do not include those instances mentioned above which are entirely solved by BTP-merging. We give details about those benchmark-domains where BTP-merging

[4] http://www.cril.univ-artois.fr/CPAI08

was most effective. All other benchmark-domains are grouped together in the last line of the table. The table shows the number of instances in the benchmark-domain, the average number of values (i.e. variable-value assignments) in the instances from this benchmark-domain, the average number of values deleted (i.e. the number of BTP-merging operations performed) and finally this average represented as a percentage of the average number of values.

We can see that for certain types of problem, BTP-merging is very effective, whereas for others (grouped together in the last line of the table) hardly any merging of values occurred.

4 Generalising BTP-merging to constraints of arbitrary arity

In the remainder of the paper, we assume that the constraints of a general-arity CSP instance I are given in the form described in Definition 2, i.e. as a set of incompatible tuples NoGoods(I), where a tuple is a set of variable-value assignments. For simplicity of presentation, we use the predicate Good(I, t) which is true iff the tuple t is a partial solution, i.e. t does not contain any pair of distinct assignments to the same variable and $\nexists t' \subseteq t$ such that $t' \in$ NoGoods(I). We first generalise the notion of broken triangle and merging to the general-arity case, before showing that absence of broken triangles allows merging.

Definition 5. *A* general-arity broken triangle *(GABT) on values $a, b \in \mathcal{D}(x)$ consists of a pair of tuples t, u (containing no assignments to variable x) satisfying the following conditions:*

1. *Good($I, t \cup u$) $\wedge$ Good($I, t \cup \{\langle x, a\rangle\}$) $\wedge$ Good($I, u \cup \{\langle x, b\rangle\}$)*
2. *$t \cup \{\langle x, b\rangle\} \in$ NoGoods(I) $\wedge$ $u \cup \{\langle x, a\rangle\} \in$ NoGoods(I)*

The pair of values $a, b \in \mathcal{D}(x)$ is GABT-free *if there is no broken triangle on a, b.*

Observe that Good($I, t \cup \{\langle x, a\rangle\}$) entails $t \cup \{\langle x, a\rangle\} \notin$ NoGoods(I). Hence to decide whether there is a GABT on a, b in a CSP instance, one can either explore all pairs $t \cup \{\langle x, b\rangle\}, u \cup \{\langle x, a\rangle\} \in$ NoGoods(I), as suggested by Definition 5, or, equivalently, explore all pairs $t \cup \{\langle x, a\rangle\}, u \cup \{\langle x, b\rangle\}$ of tuples explicitly allowed by the constraints in I. Whatever the representation, a pair t, u can be checked to be a GABT on a, b by evaluating the properties of Definition 5, all of which involve only constraint checks. Hence deciding whether a pair a, b is GABT-free is polytime for constraints given in extension (as the set of satisfying assignments) as well as for those given by nogoods (the set of assignments violating the constraint).

Definition 6. Merging *values $a, b \in \mathcal{D}(x)$ in a general-arity CSP instance I consists in replacing a, b in $\mathcal{D}(x)$ by a new value c which is compatible with all variable-value*

assignments compatible with at least one of the assignments $\langle x, a\rangle$ or $\langle x, b\rangle$, thus producing an instance I' with the new set of nogoods defined as follows:

$$\begin{aligned}\text{NoGoods}(I') = &\{t \in \text{NoGoods}(I) \mid \langle x, a\rangle, \langle x, b\rangle \notin t\} \\ &\cup \{t \cup \{\langle x, c\rangle\} \mid t \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I) \ \wedge \\ &\qquad \exists t' \in \text{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle x, b\rangle\}\} \\ &\cup \{t \cup \{\langle x, c\rangle\} \mid t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I) \ \wedge \\ &\qquad \exists t' \in \text{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle x, a\rangle\}\}\end{aligned}$$

A value-merging condition *is a polytime-computable property $P(x, a, b)$ of assignments $\langle x, a\rangle$, $\langle x, b\rangle$ in a CSP instance I such that when $P(x, a, b)$ holds, the instance I' is satisfiable if and only if I is satisfiable.*

Clearly, this merging operation can be performed in polynomial time whether constraints are represented positively in extension or negatively as nogoods.

Proposition 3. *In a general-arity CSP instance, being GABT-free is a value-merging condition. Furthermore, given a solution to the instance resulting from the merging of two values, we can find a solution to the original instance in linear time.*

Proof. In order to prove that satisfiability is preserved by this merging operation, it suffices to show that if s is a solution to I' containing $\langle x, c\rangle$, then either $s_a = (s \setminus \{\langle x, c\rangle\}) \cup \{\langle x, a\rangle\}$ or $s_b = (s \setminus \{\langle x, c\rangle\}) \cup \{\langle x, b\rangle\}$ is a solution to I. Suppose, for a contradiction that this is not the case. Then there are tuples $t, u \subseteq s \setminus \{\langle x, c\rangle\}$ such that $t \cup \{\langle x, b\rangle\} \in$ NoGoods(I) and $u \cup \{\langle x, a\rangle\} \in$ NoGoods(I). Since t, u are subsets of the solution s to I' and t, u contain no assignments to x, we have Good($I, t \cup u$). Since $t \cup \{\langle x, c\rangle\}$ is a subset of the solution s to I', we have $t \cup \{\langle x, c\rangle\} \notin$ NoGoods(I'). By the definition of NoGoods(I') given in Definition 6, and since $t \cup \{\langle x, b\rangle\} \in$ NoGoods(I), we know that $\nexists t' \in$ NoGoods(I) such that $t' \subseteq t \cup \{\langle x, a\rangle\}$. But then Good($I, t \cup \{\langle x, a\rangle\}$). By a symmetric argument, we can deduce Good($I, u \cup \{\langle x, b\rangle\}$). This provides the contradiction we were looking for, since we have shown that a general-arity broken triangle occurs on t, u, $\langle x, a\rangle$, $\langle x, b\rangle$.

Reconstructing a solution to the original instance can be achieved in linear time, since it suffices to verify which (or both) of s_a or s_b is a solution to I. □

Relationship with Resolution in SAT

We now show that in the case of Boolean domains, there is a close relationship between merging two values a, b on which no GABT occurs and a common preprocessing operation used by SAT solvers. Given a propositional CNF formula φ in the form of a set of clauses (each clause C_i being represented as a set of literals) and a variable x occurring in φ, recall that *resolution* is the process of inferring the clause $(C_0 \cup C_1)$ from the two clauses $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1)$. Define the formula $Res(x, \varphi)$ to be the result of performing all such resolutions on φ, removing all clauses containing x or $\bar{x}$, and removing subsumed clauses:

$$Res(x, \varphi) = \min_{\subset}(\{C \mid C \in \varphi; x, \bar{x} \notin C\} \cup \{(C_0 \cup C_1) \mid (\{\bar{x}\} \cup C_0), (\{x\} \cup C_1) \in \varphi\})$$

It is a well-known fact that $Res(x, \varphi)$ is satisfiable if and only if φ is.

Eliminating variables in this manner from SAT instances, to get an equisatisfiable formula with less variables, is a common preprocessing step in SAT solving, and is typically performed provided it does not increase the size of the formula [7]. A particular case is when it amounts to simply removing all occurrences of x, which is the case, for instance, if x or $\bar{x}$ is unit or pure in φ, or if all resolutions on x yield a tautological clause.

Definition 7. *A variable x is said to be* erasable *from a CNF φ if*

$$Res(x, \varphi) \subseteq \{C \mid C \in \varphi; x, \bar{x} \notin C\} \cup \{C_0 \mid (\{\bar{x}\} \cup C_0) \in \varphi\} \cup \{C_1 \mid (\{x\} \cup C_1) \in \varphi\}$$

A CNF φ can be seen as the CSP instance I_φ on the set X of variables occurring in φ, with $\mathcal{D}(x) = \{\top, \bot\}$ for all $x \in X$, and NoGoods$(I_\varphi) = \{\overline{C} \mid C \in \varphi\}$, where $\overline{(\{x_1, \cdots x_p, \bar{x}_{p+1}, \cdots, \bar{x}_q\})} = \{\langle x_1, \bot\rangle, \ldots, \langle x_p, \bot\rangle, \langle x_{p+1}, \top\rangle, \ldots, \langle x_q, \top\rangle\}$.

Proposition 4. *Assume that no GABT occurs on values $\bot, \top$ for x in I_φ. Assume moreover that no clause in φ is subsumed by another one*[5]*. Then x is erasable from φ.*

Proof. Rephrasing Definition 5 in terms of clauses, for any two clauses $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1) \in \varphi$ we have one of (i) $\exists C \in \varphi, C \subseteq (C_0 \cup C_1)$, (ii) $\exists C' \in \varphi, C' \subseteq (C_0 \cup \{x\})$, or (iii) $\exists C'' \in \varphi, C'' \subseteq (C_1 \cup \{\bar{x}\})$. Moreover, in Case (ii) C' must contain x, for otherwise the clause $(\{\bar{x}\} \cup C_0)$ would be subsumed in φ, contradicting our assumption. Similarly, in Case (iii) C'' must contain $\bar{x}$.

In Case (i) the resolvent $(C_0 \cup C_1)$ of $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1)$ is subsumed by C in $Res(x, \varphi)$, and hence does not occur in it. Similarly, in the second case $(C_0 \cup C_1)$ is subsumed by the resolvent of $(\{\bar{x}\} \cup C_0)$ and C', which is precisely C_0. The third case is dual. We finally have that the only resolvents added are of the form C_0 (resp. C_1) for some clause $(\{\bar{x}\} \cup C_0)$ (resp. $(\{x\} \cup C_1)$) of φ, as required. □

We can show the converse is also true provided that a very reasonable property holds.

Proposition 5. *Assume that φ satisfies: $\forall (\{x\} \cup C) \in \varphi, \nexists C' \subseteq C, (\{\bar{x}\} \cup C') \in \varphi$ and dually $\forall (\{\bar{x}\} \cup C) \in \varphi, \nexists C' \subseteq C, (\{x\} \cup C') \in \varphi$. If x is erasable from φ, then no GABT occurs on values $\bot, \top$ for x in I_φ.*

Proof. Assume for a contradiction that there is a GABT on values $\bot, \top$ for x in I_φ, let t, u be witnesses to this, and write $t \cup \{\langle x, \top\rangle\} = \overline{(\{\bar{x}\} \cup C_0)}$, $u \cup \{\langle x, \bot\rangle\} = \overline{(\{x\} \cup C_1)}$. Then the clause $(C_0 \cup C_1)$ is produced by resolution on x. Since x is erasable, $(C_0 \cup C_1)$ is equal to or subsumed by a clause $C \in Res(x, \varphi)$, where (applying Definition 7 in reverse) either C, or $(\{x\} \cup C)$, or $(\{\bar{x}\} \cup C)$ is in φ. The first case contradicts Good$(I_\varphi, t \cup u)$, so assume by symmetry $(\{x\} \cup C) \in \varphi$. From $C \notin \varphi$ and $C \in Res(x, \varphi)$ we get $\exists C' \subseteq C, (\{\bar{x}\} \cup C') \in \varphi$. But then the pair of clauses $(\{x\} \cup C), (\{\bar{x}\} \cup C') \in \varphi$ violates the assumption of the claim. □

[5] This is without loss of generality since such clauses can be removed in polytime and such removal preserves logical equivalence.

5 A tractable class of general-arity CSP

In binary CSP, the broken-triangle property defines an interesting tractable class when broken-triangles are forbidden according to a given variable ordering. Unfortunately, the original definition of BTP was limited to binary CSPs [5]. Section 4 described a general-arity version of the broken-triangle property whose absence on two values allows these values to be merged while preserving satisfiability. An obvious question is whether GABT-freeness can be adapted to define a tractable class. In this section we show that this is possible for a fixed variable ordering, but not if the ordering is unknown.

Definition 5 defined a general-arity broken triangle (GABT). What happens if we forbid GABTs according to a given variable ordering? Absence of GABTs on two values a, b for the last variable x in the variable ordering allows us to merge a and b while preserving satisfiability. It is possible to show that if GABTs are absent on all pairs of values for x, then we can merge all values in the domain $D(x)$ of x to produce a singleton domain. This is because (as we will show later) merging a and b, to produce a merged value c, cannot introduce a GABT on c, d for any other value $d \in \mathcal{D}(x)$. Once the domain $D(x)$ becomes a singleton $\{a\}$, we can clearly eliminate x from the instance, by deleting $\langle x, a\rangle$ from all nogoods, without changing its satisfiability. It is at this moment that GABTs may be introduced on other variables, meaning that forbidding GABTs according to a variable ordering does not define a tractable class.

Nevertheless, we will show that strengthening the general-arity BTP allows us to avoid this problem. The resulting directional general-arity version of BTP (for a known variable ordering) then defines a tractable class which includes the binary BTP tractable class as a special case.

Note that the set of general-arity CSP instances whose dual instance satisfies the BTP also defines a tractable class which can be recognised in polynomial time even if the ordering of the variables in the dual instance is unknown [8]. This DBTP class is incomparable with the class we present in the present paper (which is equivalent to BTP in binary CSP) since DBTP is known to be incomparable with the BTP class already in the special case of binary CSP [8].

5.1 Directional general-arity BTP

We suppose given a total ordering $<$ of the variables of a CSP instance I. We write $t^{<x}$ to represent the subset of the tuple t consisting of assignments to variables occurring before x in the order $<$, and $Vars(t)$ to denote the set of all variables assigned by t.

Definition 8. *A* directional general-arity (DGA) broken triangle *on assignments a, b to variable x in a CSP instance I is a pair of tuples t, u (containing no assignments to variable x) satisfying the following conditions:*

1. *$t^{<x}$ and $u^{<x}$ are non-empty*
2. *Good($I, t^{<x} \cup u^{<x}$) $\wedge$ Good($I, t^{<x} \cup \{\langle x, a\rangle\}$) $\wedge$ Good($I, u^{<x} \cup \{\langle x, b\rangle\}$)*

3. $\exists t'$ *s.t.* $Vars(t') = Vars(t) \ \wedge \ (t')^{<x} = t^{<x} \ \wedge \ t' \cup \{\langle x, a\rangle\} \notin$ *NoGoods(I)*
4. $\exists u'$ *s.t.* $Vars(u') = Vars(u) \ \wedge \ (u')^{<x} = u^{<x} \ \wedge \ u' \cup \{\langle x, b\rangle\} \notin$ *NoGoods(I)*
5. $t \cup \{\langle x, b\rangle\} \in$ *NoGoods(I)* $\wedge \ u \cup \{\langle x, a\rangle\} \in$ *NoGoods(I)*

I satisfies the directional general-arity broken-triangle property (DGABTP) *according to the variable ordering $<$ if no directional general-arity broken triangle occurs on any pair of values a, b for any variable x.*

We will show that any instance I satisfying the DGABTP can be solved in polynomial time by repeatedly alternating the following two operations: (i) merge all values in the last remaining variable (according to the order $<$); (ii) eliminate this variable when its domain becomes a singleton. We will give the two operations (merging and variable-elimination) and show that both operations preserve satisfiability and that neither of them can introduce DGA broken triangles. Moreover, as for GABT-freeness, the DGABTP can be tested in polynomial time for a given order whether constraints are given as tables of satisfying assignments or as nogoods. Indeed, in the former case, using items (3) and (4) in Definition 8 we can restrict the search for a DGA broken triangle to pairs of tuples satisfying some constraint (there must be a constraint with scope $Vars(t' \cup \{x\})$ since there is a nogood on these variables by item (5), and similarly for u'). This is sufficient to define a tractable class.

5.2 Merging

Let x be the last variable according to the variable order $<$. When values a, b in the domain of variable x do not belong to any DGA broken triangle, we can replace a, b by a new value c to produce an instance I' with the new set of nogoods given by Definition 6. Since x is the last variable in the ordering $<$, DGA broken triangles on $a, b \in \mathcal{D}(x)$ are GA broken triangles (and vice versa). Thus, from Proposition 3 we can deduce that satisfiability is preserved by this merging operation. What remains to be shown is that merging two values in the domain of the last variable cannot introduce the forbidden pattern.

Lemma 1. *Merging two values a, b into a value c in the domain of the last variable x (according to the variable order $<$) in an instance I cannot introduce a directional general-arity broken triangle (DGABT) in the resulting instance I'.*

Proof. We first claim that this operation cannot introduce a DGABT on a variable $y < x$. Indeed, assume there is a DGABT on $d, e \in \mathcal{D}(y)$ in I', that is, that there are tuples v, w such that

1. $v^{<y}$ and $w^{<y}$ are non-empty
2. Good$(I', v^{<y} \cup w^{<y}) \ \wedge \ $ Good$(I', v^{<y} \cup \{\langle y, d\rangle\}) \ \wedge \ $ Good$(I', w^{<y} \cup \{\langle y, e\rangle\})$
3. $\exists v' \ Vars(v') = Vars(v) \ \wedge \ (v')^{<y} = v^{<y} \ \wedge \ v' \cup \{\langle y, d\rangle\} \notin$ NoGoods(I')
4. $\exists w' \ Vars(w') = Vars(w) \ \wedge \ (w')^{<y} = w^{<y} \ \wedge \ w' \cup \{\langle y, e\rangle\} \notin$ NoGoods(I')
5. $v \cup \{\langle y, e\rangle\} \in$ NoGoods$(I') \ \wedge \ w \cup \{\langle y, d\rangle\} \in$ NoGoods(I')

If v' contains the assignment $\langle x, c\rangle$ then, by construction of NoGoods(I') (Definition 6), $\exists v'' \in \{(v' \setminus \langle x, c\rangle) \cup \{\langle x, a\rangle\}, (v' \setminus \langle x, c\rangle) \cup \{\langle x, b\rangle\}\}$ such that $v'' \cup \{\langle y, d\rangle\} \notin$ NoGoods(I). If v' does not contain $\langle x, c\rangle$ then let $v'' = v'$. Define w'' in a similar way. Now considering the last item, if v contains $\langle x, c\rangle$ then by construction of NoGoods(I') there is v''' assigning a or b to x and otherwise equal to v, such that $v''' \cup \{\langle y, e\rangle\}$ was in NoGoods(I), and if $v \not\ni \langle x, c\rangle$ we let $v''' = v$. We define w''' similarly. Then:

1. $(v''')^{<y} = v^{<y}$ and $(w''')^{<y} = w^{<y}$ are non-empty
2. Good($I, (v''')^{<y} \cup (w''')^{<y}$) $\wedge$ Good($I, (v''')^{<y} \cup \{\langle y, d\rangle\}$) $\wedge$ Good($I, (w''')^{<y} \cup \{\langle y, e\rangle\}$) (since x is the last variable, $(v''')^{<y} = v^{<y}$ and $(w''')^{<y} = w^{<y}$)
3. $Vars(v'') = Vars(v''')$ $\wedge$ $(v'')^{<y} = (v''')^{<y}$ $\wedge$ $v'' \cup \{\langle y, d\rangle\} \notin$ NoGoods(I)
4. $Vars(w'') = Vars(w''')$ $\wedge$ $(w'')^{<y} = (w''')^{<y}$ $\wedge$ $w'' \cup \{\langle y, e\rangle\} \notin$ NoGoods(I))
5. $v''' \cup \{\langle y, e\rangle\} \in$ NoGoods(I) $\wedge$ $w''' \cup \{\langle y, d\rangle\} \in$ NoGoods(I)

that is, there was a DGABT on d, e in I, contradicting our assumption.

We now show that a broken triangle cannot be introduced on x. Observe that since x is the last variable, for all tuples t not containing an assignment to x, $t^{<x} = t$ holds. We use this tacitly in the rest of the proof. Suppose for a contradiction that I contained no DGABT, but that after merging $a, b \in D(x)$ in I to produce the instance I', in which a, b have been replaced by a new value c, we have a DGABT on c, d. Then there is a pair of non-empty tuples t, u (containing no assignments to variable x) satisfying in particular the following conditions:

(1) Good($I', t \cup u$)
(2) Good($I', t \cup \{\langle x, c\rangle\}$)
(3) Good($I', u \cup \{\langle x, d\rangle\}$)
(4) $t \cup \{\langle x, d\rangle\} \in$ NoGoods(I')
(5) $u \cup \{\langle x, c\rangle\} \in$ NoGoods(I')

We show that there was a DGABT in I either on a, d, on b, d or on a, b.

Since merging only affects tuples containing $\langle x, a\rangle$ or $\langle x, b\rangle$, (1) implies that Good($I, t \cup u$) and hence Good($I, t \cup u'$) for all $u' \subseteq u$. Similarly, (3) implies that Good($I, u \cup \{\langle x, d\rangle\}$) and hence Good($I, u' \cup \{\langle x, d\rangle\}$) for all $u' \subseteq u$. Similarly, (4) implies that $t \cup \{\langle x, d\rangle\} \in$NoGoods($I$).
There are three possible cases to consider:

(a) Good($I, t \cup \{\langle x, a\rangle\}$),
(b) Good($I, t \cup \{\langle x, b\rangle\}$),
(c) $\exists t_1, t_2 \subseteq t$ such that $t_1 \cup \{\langle x, a\rangle\}$, $t_2 \cup \{\langle x, b\rangle\} \in$ NoGoods(I).

case (a): By Definition 6 of the creation of nogoods during merging, (5) implies that $\exists u' \subseteq u$ such that $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I). We know that u' is non-empty since $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I) but Good($I, t \cup \{\langle x, a\rangle\}$) (and hence Good($I, \{\langle x, a\rangle\}$)). We have Good($I, t \cup u'$), Good($I, t \cup \{\langle x, a\rangle\}$) (and hence $t \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)), Good($I, u' \cup \{\langle x, d\rangle\}$) (and hence $u' \cup \{\langle x, d\rangle\} \notin$ NoGoods(I)), $t \cup \{\langle x, d\rangle\} \in$ NoGoods(I), $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I) and hence there was a DGABT on a, d in I.
case (b): Symmetrically to case (a), there was a DGABT on b, d in I.
case (c): We claim that Good($I, t_1 \cup \{\langle x, b\rangle\}$). If not, then we would have $\exists t_3 \subseteq t_1$ such that $t_3 \cup \{\langle x, b\rangle\} \in$ NoGoods(I) which would imply $t_1 \cup \{\langle x, c\rangle\} \in$

NoGoods(I') which is impossible since, by (2) above, we have Good($I', t \cup \{\langle x, c\rangle\}$). By a symmetrical argument, we can deduce Good($I, t_2 \cup \{\langle x, a\rangle\}$). Since Good($I, t \cup u$) and $t_1, t_2 \subseteq t$, we have Good($I, t_1 \cup t_2$). Since $t_1 \cup \{\langle x, a\rangle\} \in$ NoGoods(I) and Good($I, t_2 \cup \{\langle x, a\rangle\}$) (and hence Good($I, \{\langle x, a\rangle\}$)), we must have $t_1 \neq \emptyset$. By a symmetric argument, $t_2 \neq \emptyset$. We therefore have non-empty tuples t_1, t_2 such that Good($I, t_1 \cup t_2$), Good($I, t_1 \cup \{\langle x, b\rangle\}$ (and hence $t_1 \cup \{\langle x, b\rangle\} \notin$ NoGoods(I)), Good($I, t_2 \cup \{\langle x, a\rangle\}$) (and hence $t_2 \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)), $t_1 \cup \{\langle x, a\rangle\} \in$ NoGoods(I), $t_2 \cup \{\langle x, b\rangle\} \in$ NoGoods(I) and hence we have a DGABT in I on a, b.

Since in each of the three possible cases, we produced a contradiction, this completes the proof. □

5.3 Tractability of DGABTP for a known variable ordering

Theorem 1. *A CSP instance I satisfying the DGABTP on a given variable ordering can be solved in polynomial time.*

Proof. Suppose that I satisfies the DGABTP for variable ordering $<$ and that x is the last variable according to this ordering. Lemma 1 tells us that DGA broken triangles cannot be introduced by merging all elements in $\mathcal{D}(x)$ to form a singleton domain $\{a\}$. At this point it may be that $\{\langle x, a\rangle\}$ is a nogood. In this case the instance is clearly unsatisfiable and the algorithm halts returning this result. If not then we simply delete $\langle x, a\rangle$ from all nogoods in which it occurs. This operation of variable elimination clearly preserves satisfiability. It is polynomial time to recursively apply this merging and variable elimination algorithm until a nogood corresponding to a singleton domain is discovered or until all variables have been eliminated (in which case I is satisfiable).

To complete the proof of correction of this algorithm, it only remains to show that elimination of the last variable x cannot introduce a DGA broken triangle on another variable y. For all tuples t, u and all values $c, d \in D(y)$, none of Good($I, t^{<y} \cup u^{<y}$), Good($I, t^{<y} \cup \{\langle y, c\rangle\}$) and Good($I, u^{<y} \cup \{\langle y, d\rangle\}$) can become true due to the variable elimination operation described above. On the other hand it is possible that $t \cup \{\langle y, d\rangle\}$ or $u \cup \{\langle y, c\rangle\}$ becomes a nogood due to variable elimination. Without loss of generality, suppose that $t \cup \{\langle y, d\rangle\}$ becomes a nogood and that $t' \cup \{\langle y, d\rangle\}$ is not a nogood for some t' such that $(t')^{<y} = t^{<y}$. Then by construction there was a nogood $t \cup \{\langle y, d\rangle\} \cup \{\langle x, a\rangle\}$ before the variable x (with singleton domain $\{a\}$) was eliminated, and $t' \cup \{\langle y, d\rangle\} \cup \{\langle x, a\rangle\}$ was not a nogood. But then there was a DGA broken triangle (given by tuples $t \cup \{\langle x, a\rangle\}$, u on values $c, d \in D(y)$) before elimination of x. This contradiction shows that variable elimination cannot introduce DGA broken triangles. □

5.4 Finding a DGABTP variable ordering is NP-hard

An important question is the tractability of the recognition problem of the class DGABTP when the variable order is not given, i.e. testing the existence of a

variable ordering for which a given instance satisfies the DGABTP. In the case of binary CSP, this test can be performed in polynomial time [5]. Unfortunately, as the following theorem shows, the problem becomes NP-complete in the general-arity case.

Theorem 2. *Testing the existence of a variable ordering for which a CSP instance satisfies the DGABTP is NP-complete (even if the arity of constraints is at most 5).*

Proof. The problem is in NP since verifying the DGABTP is polytime for a given order, so it suffices to give a polynomial-time reduction from the well-known NP-complete problem 3SAT. Let I_{3SAT} be an instance of 3SAT with variables $X_1, \ldots, X_N$ and clauses $C_1, \ldots, C_M$. We will create a CSP instance I_{CSP} which has a DGABTP variable-ordering if and only if I_{3SAT} is satisfiable. For each variable X_i of I_{3SAT}, we add two variables x_i, y_i to I_{CSP}. To complete the set of variables in I_{CSP}, we add three special variables v, w, z. We add constraints to I_{CSP} in such a way that each DGABTP ordering of its variables corresponds to a solution to I_{3SAT} (and vice versa). The role of the variable z is critical: a DGABTP ordering $>$ of the variables of I_{CSP} corresponds to a solution to I_{3SAT} in which $X_i = \texttt{true} \Leftrightarrow x_i > z$. The variables y_i are used to code $\overline{X_i}$: $y_i > z$ in a DGABTP ordering if and only if $X_i = \texttt{false}$ in the corresponding solution to I_{3SAT}. The variables v, w are necessary for our construction and will necessarily satisfy $v, w < z$ in a DGABTP ordering. Each clause $C = l_1 \vee l_2 \vee l_3$, where l_1, l_2, l_3 are literals in I_{3SAT}, is imposed in I_{CSP} by adding constraints which force one of $\overline{l_1}$, $\overline{l_2}$, $\overline{l_3}$ to be false. To give a concrete example, if $C = X_1 \vee X_2 \vee X_3$, then constraints are added to I_{CSP} to force $y_1 < z$ or $y_2 < z$ or $y_3 < z$ in a DGABTP ordering. If the clause C contains a negated variable $\overline{X_i}$ instead of X_i, it suffices to replace y_i by x_i.

We now give in detail the necessary gadgets in I_{CSP} to enforce each of the following properties in a DGABTP ordering:

1. $v, w < z$
2. $y_i < z \Leftrightarrow x_i > z$
3. $y_i < z$ or $y_j < z$ or $y_k < z$

We introduce broken triangles in order to impose these properties. However, it is important not to inadvertently introduce other broken triangles. This can be avoided by making all pairs of assignments $\langle x, a\rangle$, $\langle x', a'\rangle$ from two different gadgets incompatible (i.e. $\{\langle x, a\rangle, \langle x', a'\rangle\} \in$ NoGoods(I_{CSP})). We also assume that two gadgets which use the same variable x use distinct domain values in $\mathcal{D}(x)$. To avoid creating a trivial instance in which the gadgets disappear after establishing arc consistency, we can also add extra values in each domain which are compatible with all variable-value assignments in the gadgets.

We give the details of the three types of gadget:

1. The gadget to force $v, w < z$ in a DGABTP ordering consists of values $a_0 \in \mathcal{D}(z)$, $b_0, b_1 \in \mathcal{D}(v)$, $c_0, c_1 \in \mathcal{D}(w)$ and three nogoods $\{\langle z, a_0\rangle, \langle v, b_0\rangle\}$, $\{\langle z, a_0\rangle, \langle w, c_0\rangle\}$, $\{\langle v, b_1\rangle, \langle w, c_1\rangle\}$. The only way to satisfy the DGABTP on this triple of variables is to have $v, w < z$ since there are broken triangles on variables v and w.

2. To force $y_i < z \Leftrightarrow x_i > z$ in a DGABTP ordering we use two gadgets, the first to force $y_i > z \vee x_i > z$ and the second to force $y_i < z \vee x_i < z$.
   The first gadget is a broken triangle consisting of values $a_1, a_2 \in \mathcal{D}(z)$, $d_0 \in \mathcal{D}(x_i)$, $e_0 \in \mathcal{D}(y_i)$ and two nogoods $\{\langle z, a_1\rangle, \langle x_i, d_0\rangle\}$, $\{\langle z, a_2\rangle, \langle y_i, e_0\rangle\}$. In a DGABTP ordering we must have $y_i > z \vee x_i > z$.
   The second gadget consists of values $a_3, a_4 \in \mathcal{D}(z)$, $b_2 \in \mathcal{D}(v)$, $c_2 \in \mathcal{D}(w)$, $d_1 \in \mathcal{D}(x_i)$, $e_1 \in \mathcal{D}(y_i)$ and four nogoods $\{\langle z, a_3\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $\{\langle z, a_4\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $\{\langle z, a_4\rangle, \langle w, c_2\rangle, \langle y_i, e_1\rangle\}$, $\{\langle z, a_3\rangle, \langle w, c_2\rangle, \langle y_i, e_1\rangle\}$. We assume that we have forced $v, w < z$ using the gadget described in point (1). The tuples $t = \{\langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $u = \{\langle w, c_2\rangle, \langle y_i, e_1\rangle\}$ then form a DGA broken triangle on assignments $a_3, a_4 \in \mathcal{D}(z)$ if $x_i, y_i > z$. If either $x_i < z$ or $y_i < z$ then there is no DGA broken triangle; for example, if $x_i < z$, then we longer have Good(I_{CSP},$t^{<z} \cup \{\langle z, a_3\rangle\}$) since $t^{<z} \cup \{\langle z, a_3\rangle$ is precisely the nogood $\{\langle z, a_3\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$. Thus this gadget forces $y_i < z \vee x_i < z$ in a DGABTP ordering.
3. The gadget to force $y_i < z$ or $y_j < z$ or $y_k < z$ in a DGABTP ordering consists of values $a_5, a_6 \in \mathcal{D}(z)$, $b_3 \in \mathcal{D}(v)$, $c_3 \in \mathcal{D}(w)$, $e_2 \in \mathcal{D}(y_i)$, $e_3 \in \mathcal{D}(y_j)$, $e_4 \in \mathcal{D}(y_k)$ and five nogoods, namely $\{\langle z, a_6\rangle, \langle v, b_3\rangle, \langle y_i, e_2\rangle, \langle y_j, e_3\rangle, \langle y_k, e_4\rangle\}$, $\{\langle z, a_5\rangle, \langle w, c_3\rangle\}$, $\{\langle z, a_5\rangle, \langle y_i, e_2\rangle\}$, $\{\langle z, a_5\rangle, \langle y_j, e_3\rangle\}$, $\{\langle z, a_5\rangle, \langle y_k, e_4\rangle\}$. The tuples $t = \{\langle v, b_3\rangle, \langle y_i, e_2\rangle, \langle y_j, e_3\rangle, \langle y_k, e_4\rangle\}$, $u = \{\langle w, c_3\rangle\}$ form a DGA broken triangle on $a_5, a_6 \in \mathcal{D}(a)$ if $y_i, y_j, y_k > z$. If $y_i < z$ or $y_j < z$ or $y_k < z$, then there is no DGA broken triangle; for example, if $y_i < z$, then we longer have Good(I_{CSP},$t^{<z} \cup \{\langle z, a_5\rangle\}$) since $\{\langle z, a_5\rangle, \langle y_i, e_2\rangle\}$ is a nogood. Thus this gadget forces $y_i < z$ or $y_j < z$ or $y_k < z$ in a DGABTP ordering.

The above gadgets allow us to code I_{3SAT} as the problem of testing the existence of a DGABTP ordering in the corresponding instance I_{CSP}. To complete the proof it suffices to observe that this reduction is clearly polynomial. □

Our proof of Theorem 2 used large domains. The question still remains whether it is possible to detect in polynomial time whether a DGABTP variable ordering exists in the case of domains of bounded size, and in particular in the important case of SAT.

6 Conclusion

This paper described a novel reduction operation for binary CSP, called BTP-merging, which is strictly stronger than neighbourhood substitution. Experimental trials have shown that in several benchmark-domains applying BTP-merging until convergence can significantly reduce the total number of variable-value assignments. We gave a general-arity version of BTP-merging and demonstrated a theoretical link with resolution in SAT. From a theoretical point of view, we then went on to define a general-arity version of the tractable class defined by the broken-triangle property for a known variable ordering. Further research is required to find optimal algorithms for BTP-merging and to investigate the tractability of applying BTP-merging in instances containing global constraints.

References

1. David A. Cohen, Martin C. Cooper, Páidí Creed, Dániel Marx, and András Z. Salamon. The tractability of CSP classes defined by forbidden patterns. *Journal of Artificial Intelligence Research*, 45, 47–78, 2012.
2. David A. Cohen, Martin C. Cooper, Guillaume Escamocher and Stanislav Živný, Variable elimination in binary CSP via forbidden patterns, *Proceedings of IJCAI*, 517–523, 2013.
3. Martin C. Cooper. Fundamental properties of neighbourhood substitution in constraint satisfaction problems. *Artificial Intelligence*, 90(1-2), 1–24, 1997.
4. Martin C. Cooper and Guillaume Escamocher. A dichotomy for 2-constraint forbidden CSP patterns. In *Proceedings of AAAI*, 464–470, 2012.
5. Martin C. Cooper, Peter G. Jeavons, and András Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artificial Intelligence*, 174(9-10), 570–584, 2010.
6. Martin C. Cooper and Stanislav Živný, Tractable Triangles and Cross-Free Convexity in Discrete Optimisation, *Journal of Artificial Intelligence Research*, 44, 455–490, 2012.
7. Niklas Eén and Armin Biere, Effective Preprocessing in SAT Through Variable and Clause Elimination, In *Proceedings of SAT*, 61–75, 2005.
8. Achref El Mouelhi, Philippe Jégou and Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. In *Proceedings of ICTAI*, 947–954, 2013.
9. Philippe Jégou, Decomposition of Domains Based on the Micro-Structure of Finite Constraint-Satisfaction Problems In *Proceedings of AAAI*, 731–736, 1993.
10. Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI*, 227–233, 1991.
11. Chavalit Likitvivatanavong and Roland H.C. Yap. Eliminating redundancy in csps through merging and subsumption of domain values. *ACM SIGAPP Applied Computing Review*, 13(2), 2013.
12. András Z. Salamon and Peter G. Jeavons. Perfect Constraints Are Tractable. In *Proceedings of CP*, *LNCS* 5202, 524–528, 2008.

Constraints (2015) 20:383–413
DOI 10.1007/s10601-015-9185-y

A hybrid tractable class for non-binary CSPs

Achref El Mouelhi · Philippe Jégou · Cyril Terrioux

Published online: 11 March 2015

Abstract Find new islands of tractability, that is classes of CSP instances for which polytime algorithms exist, is a fundamental task in the study of constraint satisfaction problems. The concept of hybrid tractable class, which allows to deal simultaneously with the restrictions of languages and, for example, the satisfaction of structural properties, is an approach which has already shown its interest in this domain. Here we study a hybrid class for non-binary CSP instances. With this aim in view, we consider the Broken Triangle Property (BTP) introduced in Cooper et al. (*Artificial Intelligence*, *174*, 570–584 2010). While this tractable class has been defined for binary instances, the authors have suggested to extend it to instances with constraints of arbitrary arities, using the dual representation of such CSPs. We develop this idea by proposing a new definition without exploiting the dual representation, but using a semantic property associated to the compatibility relations of the constraints. This class is called DBTP for Dual Broken Triangle Property. We study it in depth, firstly to show that it is tractable. Then we compare it to some known classes. In particular, we prove that DBTP is incomparable with BTP and that it includes some well known tractable classes of CSPs such as β-acyclic CSPs. Then, we compare it with the Hyper-k-Consistency, which allows us to also present new results for BTP. Finally, we analyse DBTP from a practical viewpoint, by first highlighting that some benchmarks which are classically used to compare the solvers are included in DBTP and then by explaining the efficiency of solvers of the state of the art on such instances thanks to their membership of the DBTP class.

Keywords Constraint satisfaction · Tractable class

A. El Mouelhi · P. Jégou · C. Terrioux (✉)
LSIS UMR 7296, Aix-Marseille Université, CNRS, ENSAM, Université de Toulon,
13397 Marseille, France
e-mail: cyril.terrioux@univ-amu.fr

A. El Mouelhi
e-mail: achref.elmouelhi@lsis.org

P. Jégou
e-mail: philippe.jegou@lsis.org

1 Introduction

Constraint Satisfaction Problems (CSPs, see [25] for a state of the art) provide an efficient way of formulating problems in computer science, especially in Artificial Intelligence. Numerous and various problems can be modelled as CSPs like, for instance, graph coloring, frequency assignment problem, scheduling problem or Boolean satisfiability problem (SAT). The CSP problem can be considered as the problem of checking whether a finite set X of variables can be assigned in their domains of values given by D, while satisfying simultaneously a set C of constraints.

Formally, a *CSP instance* $P = (X, D, C)$ is defined by a set X of n variables (denoted $x_1, ..., x_n$), a set of domains $D = \{d_1, ..., d_n\}$ (d_i is the set of the possible values for the variable x_i) and a set C of e constraints (denoted $c_1, ..., c_e$). Each constraint c_i involves a set of variables called the *scope* of c_i and denoted $S(c_i)$. A constraint c_i allows a set of tuples over $\prod_{x_j \in S(c_i)} d_j$ defined by the relation $R(c_i)$ (i.e. we assume here that the relations are represented by sets of allowed tuples). $r_i = |S(c_i)|$ is the *arity* of the constraint c_i while r denotes the largest arity and $\rho = max_{c_i \in C}\{|R(c_i)|\}$ the size of the largest relation. In the following, without loss of generality, we assume that the CSP are normalized (i.e. $\forall c_i, c_j \in C, c_i \neq c_j, S(c_i) \neq S(c_j)$ [3]). Usually, we distinguish binary constraints whose arity is equal to 2 from non-binary ones. Likewise, binary CSPs (CSPs for which all the constraints are binary) are considered separately from CSPs with constraints of arbitrary arities. For binary CSPs, we will denote by c_{ij} the constraint involving x_i and x_j. For both binary CSPs and CSPs of arbitrary arity, the problem of deciding whether a solution (i.e. an assignment of a value to each variable which satisfies all the constraints) exists is NP-complete.

Although the problem CSP is NP-complete, there exist classes of instances that can be solved (and often recognized) in polynomial time. These classes are called "tractable classes" and rely on some properties that can be verified by the instances. There are two main kinds of such properties. The first one concerns the properties of the structure of the CSP instance which is represented by a hypergraph (a graph in the binary case), called the *constraint (hyper)graph*, whose vertices correspond to variables and edges to the constraint scopes. For example, it is well known that solving a tree-structured binary CSP (i.e. deciding whether it has a solution) can be achieved in linear time [14]. Another kind of properties is related to restrictions on the language defining the constraints. These restrictions concern the domains and/or the compatibility relations associated with the constraints. For example, it is the case for the class of "0-1-all constraints" (ZOA [7]). More recently, some tractable classes have been proposed which are related to these two kinds of properties, such as the Broken Triangle Property (BTP [8]). Their interest is that they are able to take into account both language and structure restrictions. They are thus sometimes called "hybrid classes".

In this paper, we study a hybrid tractable class called *DBTP* for *Dual Broken Triangle Property*. So, this class is based on the concept of "Broken Triangle" which is the basis of BTP. While BTP is only defined for binary constraints, DBTP is defined for CSPs whose constraints have arbitrary arities. Using the dual representation of CSPs, we can consider that this class has been firstly (and briefly) proposed in [8], as a non-binary version of BTP. However, we can also define DBTP by a semantic property related to the compatibility of tuples appearing in triples of relations associated to constraints, without an explicit link to the dual representation. But we show that these two definitions are equivalent (see Theorem 3). Nevertheless, DBTP is a tractable class quite different from BTP. For example, we prove that DBTP is not a generalization of BTP to constraints of arbitrary arity since in the case of

Constraints (2015) 20:383–413 385

binary CSPs, BTP and DBTP are formally different (see Theorem 12). Another example of these differences is related to the fact that DBTP is a conservative property for the filtering of domains and for the filtering of relations while BTP is conservative only for the filtering of domains. Moreover, we show that this tractable class includes simultaneously, structural classes such as $\beta\text{-}acyclic$ CSPs but also classes defined by language restrictions. We also establish that DBTP is incomparable with many well known tractable classes (e.g. ZOA [7], row-convex [1] or max-closed [19]).

As mentioned above, we prove that DBTP is a conservative property for many classical filterings like arc-consistency or pairwise consistency. It ensues that DBTP seems to have a real practical interest since any instance satisfying DBTP can be solved in polytime using algorithms similar to MAC (Maintaining Arc-Consistency [26]) or RFL (Real Full Look-ahead [23]).

This paper is organized as follows. In Section 2, we introduce the class DBTP and provide its main features. Then in Section 3, we study the relationship between BTP and DBTP and show that DBTP includes β-acyclic CSPs. Section 4 examines the relationship between DBTP and other well known tractable classes. Then, due to the relationship between BTP and hyper-3-consistency, we study the links between (D)BTP and (hyper)-k-consistency. Finally, we study DBTP from a practical viewpoint by showing that some benchmarks of the CSP 2008 Competition have the DBTP property and by making the links with their solving efficiency before concluding and giving some perspectives in Section 7.

2 The tractable class DBTP

In this section, we first define the DBTP class and present some of its properties (notably its tractability) before studying the effects of filtering on DBTP instances and their consequences on the efficiency of their solving by solvers of the state of the art.

2.1 Definition and properties

First, we recall the BTP property on which the DBTP property relies:

Definition 1 (Broken Triangle Property [8]) A CSP instance (X, D, C) satisfies the **Broken Triangle Property** (BTP) w.r.t. the variable ordering $<$ if, for all triples of variables (x_i, x_j, x_k) s.t. $x_i < x_j < x_k$, s.t. $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$ and $(v_j, v'_k) \in R(c_{jk})$, then either $(v_i, v'_k) \in R(c_{ik})$ or $(v_j, v_k) \in R(c_{jk})$. If neither of these two tuples exist, $\langle (v_i, v_j), (v_i, v_k), (v_j, v'_k) \rangle$ is called a **Broken Triangle on** x_k.

Let BTP be the set of the instances for which BTP holds w.r.t. some variable ordering.

The BTP property is relative to the compatibility between the values of domains which can be graphically visualized on the micro-structure graph.

Definition 2 (Micro-structure [21]) The **micro-structure** of a binary CSP instance $P = (X, D, C)$ is the undirected graph $\mu(P) = (V, E)$ where $V = \{(x_i, v_i) : x_i \in X, v_i \in d_i\}$ and $E = \{ \{(x_i, v_i), (x_j, v_j)\} : i \neq j, c_{ij} \notin C \text{ or } (v_i, v_j) \in R(c_{ij})\}$.

As each of these compatibilities involves as many values as the arity of the considered constraint, such a property cannot be easily generalized to non-binary CSPs.

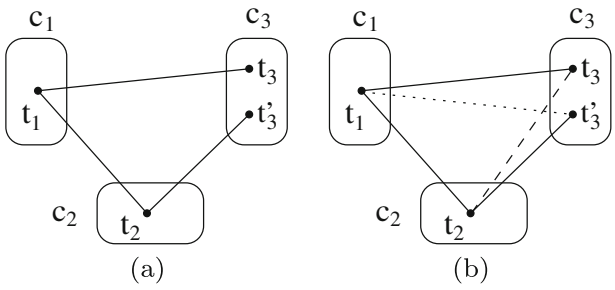


Fig. 1 Illustration of DBTP on the constraints c_1, c_2 and c_3

So a natural alternative[1] consists in considering the compatibilities between the relations through the notion of dual of a CSP instance. Before defining formally the notion of dual, we introduce the notation $t[Y']$, which represents the restriction of the tuple t of $\prod_{x_i \in Y} d_i$ (where Y is a subset of variables) to the variables of the subset $Y' \subseteq Y$.

Definition 3 (Dual) The **dual** of the CSP instance $P = (X, D, C)$ is the binary CSP instance $P^d = (X^d, D^d, C^d)$ where each constraint c_i of C is associated to the variable x_i^d of X^d whose domain d_i^d is defined by the tuples t_i of $R(c_i)$ s.t. $\forall x_j \in S(c_i), t_i[\{x_j\}] \in d_j$, and a constraint c_{ij}^d of C^d links two variables x_i^d and x_j^d of X^d if the corresponding constraints c_i and c_j of C share at least a variable (i.e. $S(c_i) \cap S(c_j) \neq \emptyset$). The relation $R(c^d)$ is defined by the tuples $(t_i, t_j) \in d_i^d \times d_j^d$ s.t. $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$.

It is well known that, for any CSP P, P has a solution iff P^d has a solution.
We now define the DBTP property:

Definition 4 (Dual Broken-Triangle Property) A CSP instance P satisfies the **Dual Broken Triangle Property** (DBTP) w.r.t. the constraint ordering $\prec$ iff the dual of P satisfies BTP w.r.t. $\prec$.
Let $DBTP$ be the set of the instances for which the DBTP property holds for some constraint ordering.

We can observe graphically the DBTP property on the micro-structure of the dual of the original instance. For instance, Fig. 1a represents the micro-structure of the dual instance of a CSP P with three constraints c_1, c_2 and c_3. In this example, we consider four tuples, $t_1 \in R(c_1)$, $t_2 \in R(c_2)$ and $t_3, t_3' \in R(c_3)$ s.t. $t_1[S(c_1) \cap S(c_2)] = t_2[S(c_1) \cap S(c_2)]$, $t_1[S(c_1) \cap S(c_3)] = t_3[S(c_1) \cap S(c_3)]$, $t_2[S(c_2) \cap S(c_3)] = t_3'[S(c_2) \cap S(c_3)]$, $t_1[S(c_1) \cap S(c_3)] \neq t_3'[S(c_1) \cap S(c_3)]$ and $t_2[S(c_2) \cap S(c_3)] \neq t_3[S(c_2) \cap S(c_3)]$. If we consider the ordering $c_1 \prec c_2 \prec c_3$, P does not satisfy DBTP w.r.t. $\prec$. Now, if we have P' (see Fig. 1b) s.t. either t_1 and t_3' (dotted edge), or t_2 and t_3 (dashed edge) or both are compatible, then P' satisfies DBTP according to $\prec$.

[1] Such an idea has already been introduced in [8] but it was just mentioned briefly and thus, it was not studied in depth.

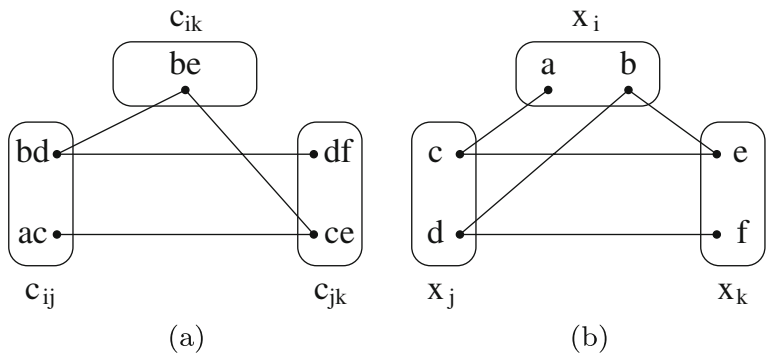


Fig. 2 An instance which satisfies DBTP (**a**) but not BTP (**b**)

The class $DBTP$ differs necessarily from the class BTP since $DBTP$ may contain non-binary instances while BTP is restricted to binary instances. It follows a natural question about the comparison of these two classes in the particular case of binary CSPs. In particular, a binary instance may satisfy DBTP while not satisfying BTP. For instance, Fig. 2b depicts the micro-structure of a binary instance while Fig. 2a represents the micro-structure of its dual. This instance is DBTP w.r.t. the ordering $c_{ij} \prec c_{jk} \prec c_{ik}$ but not BTP since $\langle (c,e),(c,a),(e,b)\rangle$, $\langle (b,e),(b,d),(e,c)\rangle$ and $\langle (b,d),(b,e),(d,f)\rangle$ are broken triangles respectively on x_i, x_j and x_k. Note that, given three variables (respectively constraints), the existence of a broken triangle on each of them with respect to the two other is a sufficient condition in order to prevent the existence of a suitable ordering w.r.t. to BTP (resp. DBTP). Conversely, a binary instance can satisfy BTP but not DBTP. This case is illustrated in Fig. 3. Indeed, BTP holds w.r.t. the ordering $x_i < x_j < x_k$ while DBTP does not hold because $\langle (bg,dg),(bg,bf),(dg,ad)\rangle$, $\langle (bd,dg),(bd,bi),(dg,cg)\rangle$ and $\langle (bd,bg),(bd,dh),(bg,eg)\rangle$ are broken triangles (in broken lines in Fig. 3) respectively on c_{ij}, c_{ik} and c_{jk}. Theorem 1 is deduced from these examples.

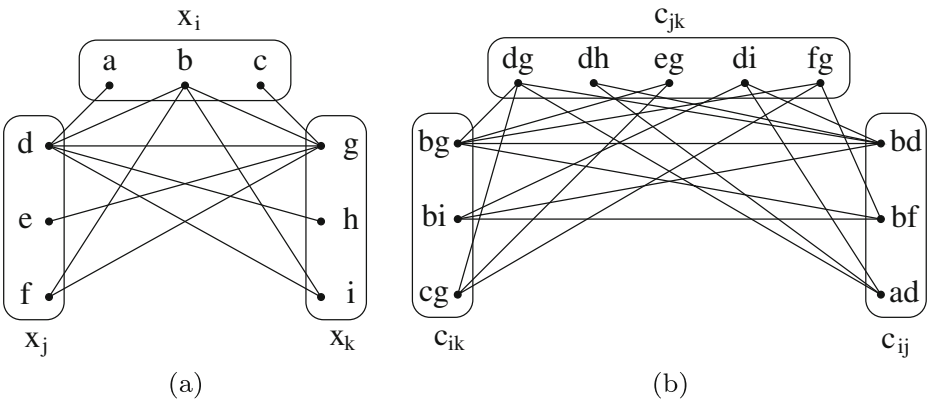


Fig. 3 An instance satisfying BTP (**a**) but not DBTP (**b**)

Theorem 1 *Let P be a binary CSP instance.*

- *P satisfies DBTP $\not\Rightarrow$ P satisfies BTP,*
- *P satisfies BTP $\not\Rightarrow$ P satisfies DBTP.*

This first theorem shows that DBTP is then not a generalization of BTP to non-binary CSPs.

We now prove that the class of CSP instances which satisfy DBTP is tractable, thanks to the two next lemmas, whose proofs exploit the approach proposed in [8].

Lemma 1 *Any CSP instance P which satisfies DBTP w.r.t. the constraint ordering $\prec$ can be solved in $O(e^2.r.\rho^2)$.*

Proof The first step consists in building the dual of P, what can be achieved in $O(e^2.r.\rho^2)$. Then, as the dual of P is BTP, we know that it can be solved in $O(e^2.\rho^2)$ [8]. Hence, the overall complexity is $O(e^2.r.\rho^2)$. □

Lemma 2 expresses that the constraint ordering $\prec$ related to DBTP may be computed (if any) in polynomial time.

Lemma 2 *Given any CSP instance P, determining if a constraint ordering $\prec$ s.t. P is DBTP w.r.t. $\prec$ exists (and finding it if any) can be achieved in polynomial time.*

Proof A possible algorithm consists in computing first the dual of P and then determining if an ordering $\prec$ s.t. the dual of P is BTP exists like in [8]. Both steps are polynomial (see the previous proof and [8]). Hence, the overall complexity is polynomial. □

The two previous lemmas allow to establish the tractability of DBTP.

Theorem 2 *DBTP is a tractable class.*

We now present an alternative and equivalent characterization of DBTP:

Theorem 3 *A CSP instance $P = (X, D, C)$ satisfies the DBTP property w.r.t. the constraint ordering $\prec$ iff for all triples of constraints (c_i, c_j, c_k) s.t. $c_i \prec c_j \prec c_k$, for all $t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$ s.t.*

- $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$
- $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$
- $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$

then

- *either* $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$
- *or* $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$.

Proof P satisfies DBTP w.r.t. $\prec$
$\Leftrightarrow$ P^d satisfies BTP w.r.t. $\prec$

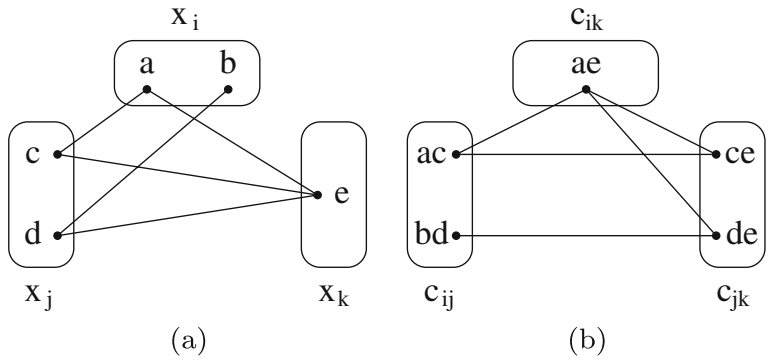


Fig. 4 An instance satisfying DBTP with its micro-structure (**a**) and the micro-structure of its dual (**b**)

$\Leftrightarrow$ for all triples of variables (x_i^d, x_j^d, x_k^d) s.t. $x_i^d \prec x_j^d \prec x_k^d$, for all $t_i \in d_i^d$, $t_j \in d_j^d$ and $t_k, t'_k \in d_k^d$ s.t. $(t_i, t_j) \in R(c_{ij}^d)$, $(t_i, t_k) \in R(c_{ik}^d)$ and $(t_j, t'_k) \in R(c_{jk}^d)$ then either $(t_i, t'_k) \in R(c_{ik}^d)$ or $(t_j, t_k) \in R(c_{jk}^d)$

$\Leftrightarrow$ for all triples of constraints (c_i, c_j, c_k) s.t. $c_i \prec c_j \prec c_k$, for all $t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$ s.t. $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$, $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$ and $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$ then either $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$ or $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$. □

In order to illustrate this characterization, let us consider the CSP depicted in Fig. 4a. Clearly, from the micro-structure of its dual (see Fig. 4b), we can deduce that this instance is DBTP w.r.t. the ordering $c_{ij} \prec c_{ik} \prec c_{jk}$ but not w.r.t. $c_{jk} \prec c_{ik} \prec c_{ij}$. Now, if we take into account the alternative characterization, we obtain the same conclusions. Indeed, regarding the ordering $c_{ij} \prec c_{ik} \prec c_{jk}$, there is a single quadruple of tuples $\left(t_{ij}, t_{ik}, t_{jk}, t'_{jk}\right)$ such that $t_{ij}[\{x_i\}] = t_{ik}[\{x_i\}]$, $t_{ij}[\{x_j\}] = t_{jk}[\{x_j\}]$ and $t_{ik}[\{x_k\}] = t'_{jk}[\{x_k\}]$, namely $t_{ij} = (a, c)$, $t_{ik} = (a, e)$, $t_{jk} = (c, e)$ and $t'_{jk} = (d, e)$. As we have $t_{ik}[\{x_k\}] = t_{jk}[\{x_k\}]$, the instance is DBTP w.r.t. $c_{ij} \prec c_{ik} \prec c_{jk}$. In contrast, for the ordering $c_{jk} \prec c_{ik} \prec c_{ij}$, the quadruple of tuples $\left(t_{jk}, t_{ik}, t_{ij}, t'_{ij}\right)$ with $t_{jk} = (d, e)$, $t_{ik} = (a, e)$, $t_{ij} = (b, d)$ and $t'_{ij} = (a, c)$ is such that $t_{jk}[\{x_k\}] = t_{ik}[\{x_k\}]$, $t_{jk}[\{x_j\}] = t_{ij}[\{x_j\}]$ and $t_{ik}[\{x_i\}] = t'_{ij}[\{x_i\}]$ but we have $t_{jk}[\{x_j\}] \neq t'_{ij}[\{x_j\}]$ and $t_{ik}[\{x_i\}] \neq t_{ij}[\{x_i\}]$. So the instance cannot be DBTP w.r.t. $c_{jk} \prec c_{ik} \prec c_{ij}$.

The alternative characterization introduced in Theorem 3 makes possible the recognition of DBTP instances directly by exploiting the tuples of relations without building the dual instance, what may be of significant interest from a practical viewpoint. We discuss this issue in Section 6.

2.2 Conservation by filtering and its consequences on solving

A filtering consistency ϕ is a function which associates to each CSP instance P an equivalent instance $\phi(P)$ (i.e. an instance which has the same solution set as P) by deleting

values and/or tuples which cannot appear in a solution of P (see [3] for more details). Filtering consistencies are commonly exploited before or during the solving in order to simplify the instances. At present, we wonder what the DBTP property becomes when applying a filtering consistency.

Definition 5 A class $\mathcal{C}$ of CSP instances is said **conservative** w.r.t. a filtering consistency ϕ if it is closed under ϕ, that is, if the instance obtained after the application of ϕ belongs to the class $\mathcal{C}$.
A property is said **conservative** if it defines a conservative class of instances.

Property 1 *DBTP is conservative for any filtering consistency which only removes values from domains or tuples from existing relations.*

Proof Let us consider a CSP instance P satisfying DBTP w.r.t. a given constraint ordering. The removal of a value from the domain of a variable x of P induces the deletion of some tuples for the constraints whose scope contains x. In other words, it implies the deletion of some values for the variables of the dual of P. On the other part, the deletion of some tuples is equivalent to remove some values from domains of some dual variables. Therefore, in both cases, the deletions of values or tuples in the original instance lead to remove values of the dual variables. As BTP is conservative w.r.t. domain filtering consistencies, the dual of P after these removals still satisfies BTP. Hence, P is still DBTP. □

For instance, this property holds for any domain filtering consistency (e.g (Generalized) Arc-Consistency or Path Inverse Consistency [4]) applied on the original instances or their dual. In particular, it is the case for the pairwise-consistency [18] (introduced in the field of Relational Databases Theory [2]) which is equivalent to applying arc-consistency on the dual instance [18]. We now recall the definitions of arc and pairwise-consistency.

Definition 6 (Arc-Consistency) Given a CSP instance $P = (X, D, C)$, a value $v_i \in d_i$ is arc-consistent w.r.t. $c \in C$ iff there exists a valid tuple $t \in R(c)$ s.t. $t[\{x_i\}] = v_i$. A domain d_i is arc-consistent w.r.t. c iff $d_i \neq \emptyset$ and $\forall v_i \in d_i$, v_i is arc-consistent w.r.t. c. P is arc-consistent iff $\forall d_i \in D$, d_i is arc-consistent w.r.t. all $c \in C$.

For example, the instance described in Fig. 5a is arc-consistent while one depicted in Fig. 4a is not arc-consistent since, notably, the value b of d_i is not arc-consistent w.r.t. the constraint c_{ik}. Hence, enforcing the arc-consistency on this instance will delete the values b and d in order to make the instance arc-consistent.

Definition 7 (Pairwise-Consistency [18]) A CSP instance $P = (X, D, C)$ is **pairwise-consistent** iff $\forall\ 1 \leq i \leq e$, $R(c_i) \neq \emptyset$ and $\forall\ 1 \leq i < j \leq e$, $R(c_i)[S(c_i) \cap S(c_j)] = R(c_j)[S(c_i) \cap S(c_j)]$.

For example, the instance of Fig. 5a is pairwise-consistent while one depicted in Fig. 4a is not pairwise-consistent since, notably, $R(c_{ij})[\{x_i\}] \neq R(c_{ik})[\{x_i\}]$ ($R(c_{ij})[\{x_i\}] = \{a, b\}$ and $R(c_{ik})[\{x_i\}] = \{a\}$). Hence, enforcing the pairwise-consistency on this instance will delete the tuples (b, d) and (d, e).

We now investigate the consequence of Property 1 on the solving of instances which are DBTP. Like MAC [26] maintains the arc-consistency (denoted AC) at each step of

the search, we define MPWC (for Maintaining PairWise Consistency) as the algorithm corresponding to maintain the pairwise-consistency at each step.

Theorem 4 *If a CSP instance P satisfies DBTP, then MPWC solves P in polynomial time w.r.t. any ordering.*

Proof As the pairwise-consistency on P is equivalent to the arc-consistency on the dual of P [18], the application of MPWC on P is equivalent to the application of MAC on the dual of P. Moreover, as P is DBTP, P^d is BTP and so, according to Theorem 7.6 of [8], MPWC solves P in polynomial time. □

Finally, we can derive a similar result for MAC as soon as enforcing the arc-consistency is sufficient to entail the pairwise-consistency. We say that enforcing the arc-consistency on a CSP $P = (X, D, C)$ entails the pairwise-consistency if the instance $P' = (X, D', C)$ obtained after having achieved AC is pairwise-consistent for the remaining values in the domains of D' (i.e. $\forall\ 1 \leq i \leq e, \exists t \in R(c_i), \forall x_{i_k} \in S(c_i), t[\{x_{i_k}\}] \in d'_{i_k}$ and $\forall\ 1 \leq i, j \leq e, i \neq j, \forall t_i \in R(c_i)\ s.t.\ \forall x_{i_k} \in S(c_i), t_i[\{x_{i_k}\}] \in d'_{i_k}, \exists t_j \in R(c_j)\ s.t.\ \forall x_{j_k} \in S(c_j), t_j[\{x_{j_k}\}] \in d'_{j_k}, t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$). Note that we consider here a particular case where the pairwise-consistency is a logical consequence of the application of the arc-consistency. In other words, we obtain the pairwise-consistency simply by enforcing the arc-consistency and so without having to enforce explicitly the pairwise-consistency. For example, the application of the arc-consistency on the instance depicted in Fig. 4 deletes the values b and d, and indirectly the tuples (b, d) and (d, e), what makes the obtained instance pairwise-consistent.

Lemma 3 *Let P be an arc-consistent CSP instance. If the instance P' obtained from P by deleting some values and enforcing AC has no empty domain and if enforcing the arc-consistency entails the pairwise-consistency, then the dual of P' is arc-consistent.*

Proof Let us consider P and P' obtained from P by deleting some values and enforcing AC such that it has no empty domain. Since P' is arc-consistent and enforcing the arc-consistency entails the pairwise-consistency, P' is also pairwise-consistent. Thus, as the pairwise-consistency on a CSP is equivalent to the arc-consistency on its dual [18], the dual of P' is arc-consistent. □

Theorem 5 *If a CSP instance P satisfies DBTP and at each step of the search, enforcing the arc-consistency entails the pairwise-consistency, then MAC can solve P in polynomial time w.r.t. any ordering.*

Proof If, after having enforced arc-consistency, no domain, neither relation is empty, then the resulting instance is pairwise-consistent and has a solution. According to lemma 3, the instance obtained after deleting some values and enforcing arc-consistency still remains pairwise-consistent. Therefore, when applying MAC on the original instance, we also maintain the pairwise-consistency. Moreover, as pairwise-consistency is equivalent to arc-consistency on the dual problem [18], Theorem 7.6 of [8] implies that MAC can solve P in polynomial time w.r.t. any ordering since the dual is BTP. □

The entailment of the pairwise-consistency when enforcing the arc-consistency occurs notably when any pair of constraints share at most one variable, as stated by the following lemma.

Lemma 4 (Prop. 8.1, p. 146 in [20]) *Let $P = (X, D, C)$ be a CSP instance s.t. $\forall c_i, c_j \in C, |S(c_i) \cap S(c_j)| \leq 1$. If P is arc-consistent, then it is pairwise-consistent.*

Theorem 6 *If a CSP instance $P = (X, D, C)$ s.t. $\forall c_i, c_j \in C, |S(c_i) \cap S(c_j)| \leq 1$ is arc-consistent and satisfies DBTP, then MAC can solve P in polynomial time w.r.t. any ordering.*

Proof According to Lemma 4, when any pair of constraints share at most one variable, achieving the arc-consistency entails the pairwise-consistency. So, from Theorem 5, MAC can solve P in polynomial time w.r.t. any ordering. □

We can note that this theorem holds in particular for binary CSPs. Moreover, we can also remark that Theorem 7.6 of [8] also holds for the algorithm RFL [23] since the proof of this theorem is only based on the enforcement of the arc-consistency at each step of the search. So, it is the same for Theorems 5 and 6. Hence, any CSP instance for which DBTP holds can be solved in polynomial time by MAC or RFL without any additional work and whatever the considered variable ordering. As most solvers of the state of the art rely on either MAC or RFL, this result may explain why these solvers are efficient in practice for solving such instances as shown in Section 6.

Finally, if we consider the instances which do not satisfy the DBTP property, we can observe that the simplified instances obtained by enforcing a given filtering consistency may be DBTP. Indeed the given filtering consistency may delete all the values or tuples which prevent the original instances from being DBTP. This issue will be studied in the next section in which we compare DBTP and BTP and in Section 6 from a practical viewpoint.

3 DBTP vs BTP

We saw with Theorem 1, that even in the case of binary CSPs, BTP and DBTP classes are different. Such a result was foreseeable since, even if the original instance and its dual represent the same problem, their structure and micro-structure are quite different. This result relies on the presence of broken triangles in the micro-structure of the instance or of its dual instance. In both cases, these broken triangles often involve values which would be deleted by some filtering consistency like arc-consistency. So, as DBTP and BTP are conservative w.r.t. domain filtering consistencies, we focus our study on binary instances which satisfy arc-consistency and thus pairwise-consistency (by lemma 3). Under these assumptions, we can prove the following lemma:

Lemma 5 *Given an arc-consistent binary CSP instance $P = (X, D, C)$, if for some triple (x_i, x_j, x_k) of variables, we have a broken triangle on x_k, then we have a broken triangle on c_{ik} and one on c_{jk} for the triple (c_{ij}, c_{ik}, c_{jk}) in the dual.*

Proof Let $x_i, x_j, x_k \in X$ s.t. $(v_i, v_j) \in R(c_{ij})$, $(v_i, v_k) \in R(c_{ik})$, $(v_j, v'_k) \in R(c_{jk})$, $(v_i, v'_k) \notin R(c_{ik})$ and $(v_j, v_k) \notin R(c_{jk})$. As P is pairwise-consistent, there are some values

$v'_i \in d_i$ and $v'_j \in d_j$ s.t. $v_i \neq v'_i$, $v_j \neq v'_j$, $\left(v'_i, v'_k\right) \in R(c_{ik})$ and $\left(v'_j, v_k\right) \in R(c_{jk})$. So, $\left\langle((v_i, v_j), (v_i, v_k)), \left((v_i, v_j), \left(v_j, v'_k\right)\right), \left((v_i, v_k), \left(v'_j, v_k\right)\right)\right\rangle$ forms a broken triangle on c_{jk} for the triple (c_{ij}, c_{ik}, c_{jk}).
Likewise for $\left\langle\left((v_i, v_j), \left(v_j, v'_k\right)\right), ((v_i, v_j), (v_i, v_k)), \left(\left(v_j, v'_k\right), \left(v'_i, v'_k\right)\right)\right\rangle$ on c_{ik}. □

The presence of a broken triangle on x_k for a triple (x_i, x_j, x_k) imposes the condition $x_k < max(x_i, x_j)$ on the variable ordering $<$ (see the proof of Theorem 3.2 of [8]). Consequently, according to lemma 5, it corresponds to impose the two conditions $c_{jk} \prec max(c_{ij}, c_{ik})$ and $c_{ik} \prec max(c_{ij}, c_{jk})$ for the triple (c_{ij}, c_{ik}, c_{jk}) on the constraint ordering $\prec$. It ensues that any arc-consistent and pairwise-consistent binary instance which satisfies BTP and has two broken triangles for two different variables of a same triple of variables cannot satisfy DBTP since we will obtain all the possible broken triangles for the corresponding triple of constraints.

Conversely, we show now that a binary instance can be arc-consistent and DBTP but not BTP. For this purpose, let us consider a binary instance with 9 variables $\{x_a, x_b, \ldots, x_i\}$. We define this instance by reproducing several times a same pattern s.t. each value appearing in an instance of the pattern does not appear in any other instance. This pattern consists in a broken triangle on a variable z for a triple (x, y, z) (i.e. which imposes the condition $z < max(x, y)$ on $<$) and each value of the variables x, y and z is linked to a given value of any variable which is not involved in this triple. We reproduce this pattern 9 times s.t. the following conditions are imposed:

- $x_a < max(x_b, x_c)$,
- $x_b < max(x_e, x_h)$,
- $x_c < max(x_e, x_g)$,
- $x_d < max(x_a, x_g)$,
- $x_e < max(x_a, x_i)$,
- $x_f < max(x_d, x_e)$,
- $x_g < max(x_h, x_i)$,
- $x_h < max(x_b, x_d)$ and
- $x_i < max(x_c, x_f)$.

Figure 5b depicts this pattern for the triple (x_a, x_b, x_c), a broken triangle on x_a (corresponding to the condition $x_a < max(x_b, x_c)$) and an independent variable x_e while Fig. 5a describes the corresponding part of the dual instance. By doing this, the micro-structure of our binary CSP or one of its dual instance have 9 connected components. We can note that this instance is not BTP because the 9 conditions make impossible the construction of a suitable variable ordering. In contrast, it is DBTP (w.r.t the ordering $c_{ab} \prec c_{ac} \prec c_{ad} \prec c_{bf} \prec c_{bh} \prec c_{ci} \prec c_{df} \prec c_{dh} \prec c_{ef} \prec c_{ei} \prec c_{gh} \prec c_{gi} \prec c_{af} \prec c_{bc} \prec c_{bd} \prec c_{ce} \prec c_{cg} \prec c_{de} \prec c_{dg} \prec c_{fi} \prec c_{hi} \prec c_{ae} \prec c_{ag} \prec c_{be} \prec c_{bg} \prec c_{cd} \prec c_{cf} \prec c_{di} \prec c_{fh} \prec c_{ai} \prec c_{bi} \prec c_{eg} \prec c_{eh} \prec c_{fg} \prec c_{ah} \prec c_{ch}$), arc-consistent and pairwise-consistent.

Now, we focus our study on acyclic CSP instances. [8] has already proved that such binary CSP instances satisfy BTP. We are going to show that this is also true for DBTP. Let $TREE$ be the set of binary CSP instances whose constraint graph is acyclic.

Theorem 7 $TREE \subsetneq DBTP$.

Proof Let $DUAL\text{-}TREE$ be the set of binary instances which are the dual of instances from $TREE$. As shown in [8], $DUAL\text{-}TREE \subsetneq BTP$. Hence $TREE \subsetneq DBTP$. □

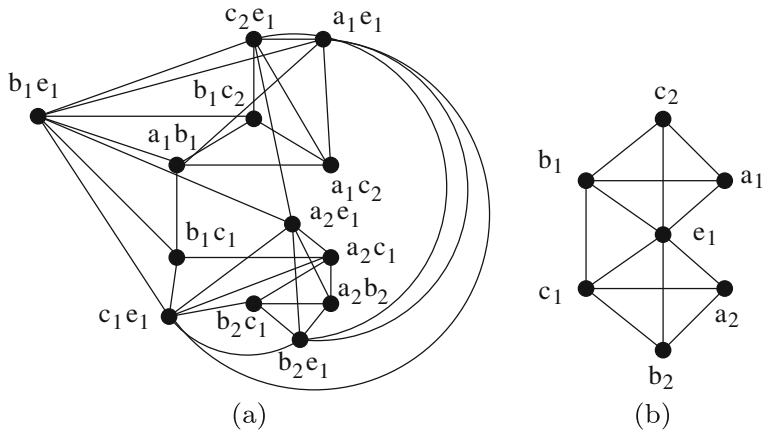


Fig. 5 Part of an instance satisfying DBTP, arc-consistency and pairwise-consistency but not BTP

This result can be extended to CSP instances of arbitrary arity. For this, we must consider the notion of cyclicity in hypergraphs, for which different degrees of cyclicity have been defined [2, 13]. Here, we are interested by α-acyclicity and β-acyclicity. We will first prove that β-acyclic CSPs satisfy DBTP and later, we will show it is not the case for α-acyclic CSPs. We first recall the definition of β-acyclicity of (constraint) hypergraph.

Definition 8 ([17]) $H = (X, C)$ is a β-acyclic hypergraph iff it has no Graham cycle. A sequence $(c_1, ..., c_m, c_{m+1})$ with $m \geq 3$ s.t. $(c_1, ..., c_m)$ are distinct and $c_1 = c_{m+1}$ is a Graham cycle if each $\Delta_i = S(c_i) \cap S(c_{i+1})$ $(1 \leq i \leq m)$ is nonempty, and whenever $i \neq j$, Δ_i and Δ_j are incomparable (i.e. $\Delta_i \not\subseteq \Delta_j$ and $\Delta_j \not\subseteq \Delta_i$).

Figure 6a depicts a β-acyclic hypergraph while we have two β-cyclic hypergraphs in b and c.

It has been recently shown in [9] that β-acyclic hypergraphs can be defined by applying the two following rules that yield the empty hypergraph:

(1) If a hyperedge is empty, we remove it from C.

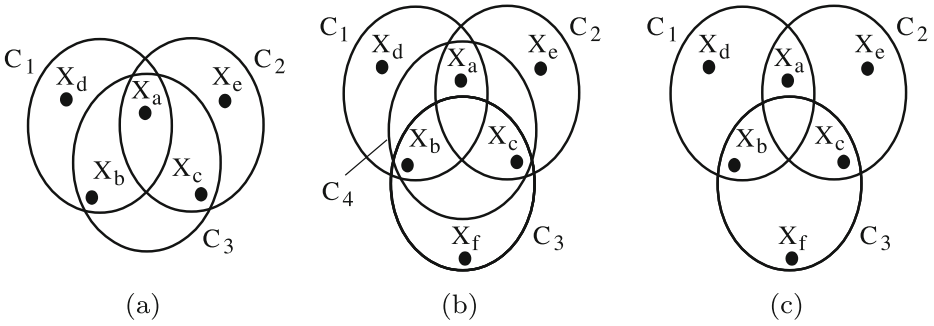


Fig. 6 A β-acyclic hypergraph (**a**), an α-acyclic and β-cyclic hypergraph (**b**) and an α and β-cyclic hypergraph (**c**)

(2) If a vertex is a nest point (i.e. the set of hyperedges containing it is a chain for the inclusion relation), then we remove it from H (i.e. from X and from the hyperedges that contain it).

Theorem 8 ([9]) *A hypergraph H is β-acyclic if and only if, after applying the two rules successively until none can be applied, we obtain the empty hypergraph.*

Using these definitions, we can now establish the next theorem:

Theorem 9 *Given a CSP instance (X, D, C), there exists a constraint ordering $\prec$, s.t. $\forall c_i, c_j, c_k \in C$ s.t. $c_i \prec c_j \prec c_k$, we have $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ or $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$ if and only if (X, D, C) has a β-acyclic constraint hypergraph.*

Proof ($\Rightarrow$) By contraposition. So we show that if the constraint hypergraph (X, C) is β-cyclic, then, there is no constraint ordering.

Consider a constraint hypergraph (X, C) which is β-cyclic. So, it has a Graham cycle, denoted by the sequence of hyperedges $(c_1, ..., c_m, c_{m+1})$. Consider an arbitrary constraint ordering $\prec$. Necessarily, among the constraints of this cycle, there is a maximum constraint c_k w.r.t. the ordering $\prec$. Consider its two neighbors in the cycle, denoted c_i and c_j (with $c_i, c_j \prec c_k$). By definition of Graham cycles, we know that $S(c_i) \cap S(c_k)$ and $S(c_j) \cap S(c_k)$ are incomparable. So, we have neither $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ nor $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$, and then no suitable constraint ordering $\prec$ exists.

($\Leftarrow$) Here, we use Theorem 8. So, given a CSP instance (X, D, C) which admits a constraint ordering $\prec$ and has a β-acyclic constraint hypergraph H, we will show that:

(1) A hyperedge $S(c_i)$ is empty iff H without $S(c_i)$ admits an ordering and is β-acyclic.
(2) A vertex x of H is a nest point such H admits an ordering iff H without x admits an ordering and is β-acyclic.

It is immediate to see that the property holds by applying the rule (1). So, consider the rule (2). Assume that for a hypergraph H we have an ordering $\prec$. So, $\forall c_i, c_j, c_k \in C$ s.t. $c_i \prec c_j \prec c_k$, we have $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ or $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$. We have five cases to consider:

1. $x \notin S(c_i) \cup S(c_j) \cup S(c_k)$: thus after the deletion of x, neither $S(c_i) \cap S(c_k)$ nor $S(c_j) \cap S(c_k)$ have changed. So, the property holds.
2. $x \in S(c_i) \cap S(c_j) \cap S(c_k)$: thus after the deletion of x, it disappears from each intersection $S(c_i) \cap S(c_k)$ and $S(c_j) \cap S(c_k)$, and thus, the property holds.
3. x belongs to only one set $S(c_i)$ or $S(c_j)$ or $S(c_k)$: so x belongs to no intersection and thus, the property holds after the deletion.
4. $x \in S(c_i) \cap S(c_j)$ and $x \notin S(c_k)$: so x belongs neither to $S(c_i) \cap S(c_k)$, nor to $S(c_i) \cap S(c_k)$ and thus, the property holds after the deletion.
5. $x \in S(c_i) \cap S(c_k)$ and $x \notin S(c_j)$ (or symmetrically $x \in S(c_j) \cap S(c_k)$ and $x \notin S(c_i)$): so before the deletion, we have necessarily $S(c_i) \cap S(c_k) \not\subseteq S(c_j) \cap S(c_k)$ and $S(c_j) \cap S(c_k) \subsetneq S(c_i) \cap S(c_k)$. Thus, after the deletion of x, we have at least $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$

So we have shown that if we can delete the whole hypergraph, which does not contradict the property on the ordering, necessarily, the first hypergraph is β-acyclic and admits a suitable constraint ordering. □

We can note that this theorem explains why the condition enunciated in lemma 4.6 of [8] (recalled below) holds independently from the scope of the constraints.

Lemma 4.6 of [8] *Let P be a CSP instance (of arbitrary arity) with constraint scopes $S(c_1), S(c_2), \ldots, S(c_e)$, where the constraints allow all combinations of values from some fixed domain D. The dual instance of P, with corresponding variables $1, 2, \ldots, e$, has the BTP property with respect to some ordering $<$ if, and only if, for all triples $S(c_i), S(c_j), S(c_k)$ with $i < j < k$ we have*

$$S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k) \text{ or } S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k).$$

Moreover, if this condition holds, then the dual of any instance P' with the same constraint scopes also has the BTP property with respect to $<$.

This lemma and the previous theorem allow us to obtain Theorem 10 where β-$ACYCLIC$ is the set of CSP instances whose constraint (hyper)graph is β-acyclic.

Theorem 10 $\beta\text{-}ACYCLIC \subsetneq DBTP$.

Proof According to Theorem 9, we know that for any β-acyclic CSP instance (X, D, C), there exists a constraint ordering $\prec$, s.t. $\forall c_i, c_j, c_k \in C$ s.t. $c_i \prec c_j \prec c_k$, we have $S(c_i) \cap S(c_k) \subseteq S(c_j) \cap S(c_k)$ or $S(c_j) \cap S(c_k) \subseteq S(c_i) \cap S(c_k)$. Moreover, according to lemma 4.6 of [8], any CSP instance satisfying the latter condition has a BTP dual. Hence $\beta\text{-}ACYCLIC \subsetneq DBTP$. □

Note that the equivalence in this lemma 4.6 only holds for the reverse direction. However this does not endanger the proof of Theorem 10. Clearly, it suffices to consider a binary instance with 3 monovalent variables (i.e. variables whose domain contains a single value) pairwise connected (each constraint allows the single tuple). Its dual satisfies BTP while the constraint graph is not β-acyclic.

Now, we show that if α-$ACYCLIC$ is the set of CSP instances whose constraint (hyper)graph is α-acyclic, then the sets α-$ACYCLIC$ and $DBTP$ are incomparable. So, we recall that α-acyclicity of (constraint) hypergraphs can be defined using the "running intersection property" [2], namely:

Definition 9 (X, C) is an α-acyclic hypergraph iff there exists an ordering $(c_1, ..., c_e)$ s.t. $\forall k, 1 < k \leq e, \exists j < k, \left(S(c_k) \cap \bigcup_{i=1}^{k-1} S(c_i) \right) \subseteq S(c_j)$.

Let us consider a CSP instance with six variables $x_a, \ldots x_f$ and four constraints whose scope are respectively $\{x_a, x_b, x_c\}$, $\{x_a, x_b, x_d\}$, $\{x_a, x_c, x_e\}$ and $\{x_b, x_c, x_f\}$. Figure 7 depicts its constraint hypergraph (a) and the micro-structure of its dual (b). We can note that this instance is α-acyclic but does not satisfy DBTP since no suitable constraint ordering exist. Moreover, it is well known that $\beta\text{-}ACYCLIC \subsetneq \alpha\text{-}ACYCLIC$ [13]. In this paper, several equivalent definitions of β-acyclicity are given, the most simple one indicating that a hypergraph is β-acyclic if and only if every one of its subhypergraphs is α-acyclic (a *subhypergraph* of a hypergraph is defined as a subset, not necessarily proper, of its set of hyperedges). For example, in Fig. 6b, we have a β-cyclic hypergraph which is α-acyclic while in (c), we have an α-cyclic hypergraph which is necessary β-cyclic.

Constraints (2015) 20:383–413 397

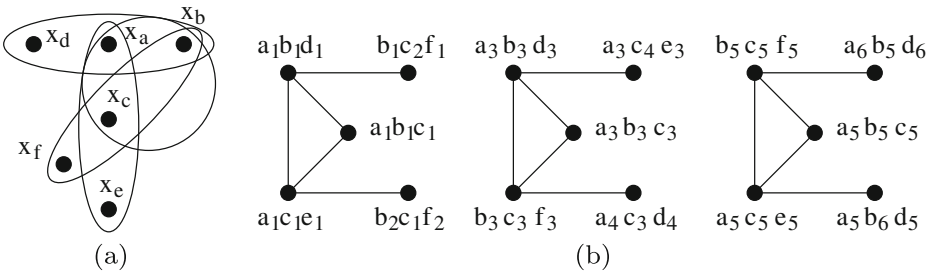


Fig. 7 An α-acyclic CSP instance (**a**) but which does not satisfy DBTP (**b**)

Hence, if we denote $A \perp B$ two tractable classes which are incomparable (i.e. neither $A \subseteq B$, nor $B \subseteq A$), we obtain the following theorem:

Theorem 11 *α-ACYCLIC $\cap$ DBTP $\neq \emptyset$ and α-ACYCLIC $\perp$ DBTP.*

Through this section, we have established:

Theorem 12 *BTP $\cap$ DBTP $\neq \emptyset$ and BTP $\perp$ DBTP.*

In the next section, we study the link between DBTP and some other well known tractable classes.

4 DBTP vs some tractable classes

4.1 For binary CSPs

As DBTP and BTP are two different classes, we first focus on some tractable classes included in BTP. These classes whose definitions are recalled below rely on restricted constraint languages.

Definition 10 (Row-convex [1]) A binary CSP instance $P = (X, D, C)$ is said **row-convex** w.r.t. a variable ordering $<$ and a value ordering, if, for each constraint c_{ij} of C with $x_i < x_j$, $\forall v_i \in d_i$, $\{v_j \in d_j | (v_i, v_j) \in R(c_{ij})\} = [a_j..b_j]$ for some a_j, $b_j \in d_j$ where $[a_j..b_j]$ denotes the values belonging to d_j between a_j and b_j w.r.t. the value ordering. We denote RC the set of row-convex instances.

Definition 11 (0-1-all [7]) A binary CSP instance $P = (X, D, C)$ is said **0-1-all** if for each constraint c_{ij} of C, for each value $v_i \in d_i$, c_{ij} satisfies one of the following conditions:

- (ZERO) for any value $v_j \in d_j$, $(v_i, v_j) \notin R(c_{ij})$,
- (ONE) there is a unique value $v_j \in d_j$ such as $(v_i, v_j) \in R(c_{ij})$,
- (ALL) for any value $v_j \in d_j$, $(v_i, v_j) \in R(c_{ij})$.

We denote ZOA the set of instances which are 0-1-all.

Definition 12 (Renamable right monotone [8]) A binary CSP instance $P = (X, D, C)$ is said **renamable right monotone** w.r.t. a variable ordering $<$ if, for $2 \leq j \leq n$, each domain

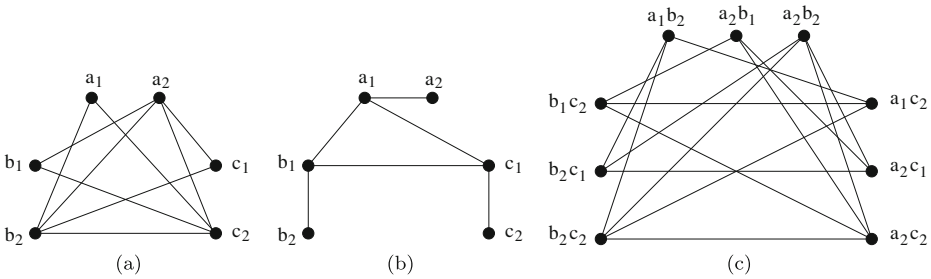


Fig. 8 A CSP instance which is RC, ZOA and RRM and has a chordal micro-structure (**a**) and a chordal complement of micro-structure (**b**), but which is not DBTP (**c**)

d_j can be ordered by $\lessdot_j$ s.t. for each constraint c_{ij} of C with $x_i < x_j$, $\forall v_i \in d_i$, $v_j, v'_j \in d_j$, if $(v_i, v_j) \in R(c_{ij})$ and $v_j \lessdot_j v'_j$ then $(v_i, v'_j) \in R(c_{ij})$.
We denote RRM the set of these instances.

The next theorem shows that these tractable classes share some instances with DBTP but are different.

Theorem 13 *$RC \cap DBTP \neq \emptyset$ and $RC \perp DBTP$.*
$ZOA \cap DBTP \neq \emptyset$ and $ZOA \perp DBTP$.
$RRM \cap DBTP \neq \emptyset$ and $RRM \perp DBTP$.

Proof If we consider the binary CSP instance of Fig. 8a, it is 0-1-all, row-convex, and renamable right monotone w.r.t. the lexicographic value and variable orderings. However, as shown in Fig. 8c, this instance is not DBTP. Conversely, any non-binary DBTP instance cannot belong to RC, ZOA or RRM.

In order to prove that $DBTP$ intersects RC, ZOA and RRM, it is sufficient to consider a monovalent and consistent binary CSP instance with three variables and three constraints since such an instance satisfies both $DBTP$, RC, ZOA and RRM. □

Some tractable classes are related to some graphical features of their micro-structure. This is the case of the class of instances which have a chordal micro-structure [15, 21]:

Definition 13 (Chordal micro-structure) A graph is said **chordal** [16] if it has no cycle of length greater than 3 without a chord (i.e. an edge joining two non-consecutive vertices in the cycle).
We denote CM the set of instances which have a **chordal micro-structure**.

Theorem 14 *$CM \cap DBTP \neq \emptyset$ and $CM \perp DBTP$.*

Proof Any monovalent and consistent binary CSP instance has a chordal micro-structure and is DBTP. So the intersection is not empty.

Consider the instance depicted in Fig. 8a. It has a chordal micro-structure but is not DBTP. Conversely, any non-binary DBTP instance cannot belong to CM. □

Likewise, the instances for which the complement of their micro-structure is chordal also form a tractable class [6]:

Definition 14 (Chordal complement of micro-structure) The complement of a graph $G = (X, E)$ is the graph (X, E') with $E' = \{\{x, y\} | x, y \in X \; s.t. \; \{x, y\} \notin E\}$.
We denote CCM the set of instances for which the complement of their micro-structure is chordal.

Theorem 15 *$CCM \cap DBTP \neq \emptyset$ and $CCM \perp DBTP$.*

Proof Any monovalent and consistent binary CSP instance has a chordal complement of micro-structure and is DBTP. So the intersection is not empty.

Consider the instance depicted in Fig. 8a. The complement of its micro-structure, depicted in Fig. 8b, is chordal but this instance is not DBTP. Conversely, any non-binary DBTP instance cannot belong to CCM. □

As chordal graphs are also perfect graphs, we can derive a similar result for the class of instances whose micro-structure is a perfect graph [27]:

Definition 15 (Perfect micro-structure) A graph is said **perfect** [16] if it contains no cycle, neither the complement of a cycle with an odd length greater than 4.
We denote PM the set of instances which have a **perfect micro-structure**.

Theorem 16 *$PM \cap DBTP \neq \emptyset$ and $PM \perp DBTP$.*

The next class relies on the number of maximal cliques of the micro-structure:

Definition 16 (Maximal clique bounded [10]) A CSP instance P is said **maximal clique bounded** if the number of maximal cliques in its micro-structure is polynomial w.r.t the size of P.
We denote CL the set of these instances.

Theorem 17 *$CL \cap DBTP \neq \emptyset$ and $CL \perp DBTP$.*

Proof Any monovalent and consistent binary CSP has a single maximal clique and is DBTP. So the intersection is not empty.

Consider any binary CSP instance s.t. its micro-structure has a polynomial number of maximal cliques. We add to such a CSP instance additional variables with additional values and additional constraints corresponding to the instance depicted in Fig. 1, s.t. these values are not compatible with those of the first part of this instance. So, it has a polynomial number of maximal cliques in its micro-structure but is not DBTP. Conversely, any non-binary DBTP instance cannot belong to CL. □

Regarding classes based on restricted structures, we have proved in Theorem 7 that $TREE \subsetneq DBTP$.

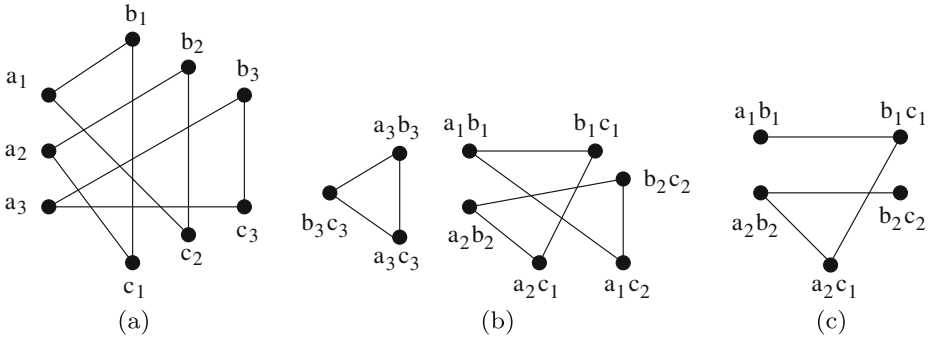


Fig. 9 An incrementally functional instance (**a**) but which does not satisfy DBTP (**b**). An instance which is DBTP but not triangular (**c**)

4.2 For CSPs of arbitrary arity

We first consider some known tractable classes based on restricted constraint languages like the max-closed class.

Definition 17 (Max-closed [19]) A CSP instance $P = (X, D, C)$ is said **max-closed** if for each constraint c of arity r_c, $\forall (v_1, v_2, \ldots, v_{r_c}), (v'_1, v'_2, \ldots, v'_{r_c}) \in R(c)$, $(max(v_1, v'_1), max(v_2, v'_2), \ldots, max(v_{r_c}, v'_{r_c})) \in R(c)$.
We denote MC the set of max-closed instances.

Theorem 18 *$MC \cap DBTP \neq \emptyset$ and $MC \perp DBTP$.*

Proof The proof of $MC \cap DBTP \neq \emptyset$ and $MC \not\subseteq DBTP$ is similar to one of Theorem 13. Regarding $DBTP \not\subseteq MC$, any CSP instance having two variables and one binary constraint is DBTP but not necessarily max-closed. □

Definition 18 (Incrementally functional [5]) A CSP instance $P = (X, D, C)$ is said **incrementally functional** if there exists a variable ordering $<$ s.t. for $1 \leq i < n$, each solution of $P[\{x_1, \ldots, x_i\}]$ extends to at most one solution of $P[\{x_1, \ldots, x_{i+1}\}]$ where, for $X' \subseteq X$, $P[X']$ denotes the CSP instance (X', D', C') where $D' = \{d_i | x_i \in X'\}$ and $C' = \{c' | c \in C \; s.t. \; S(c) \cap X' \neq \emptyset, S(c') = S(c) \cap X' \; and \; R(c') = \{t[S(c')] | t \in R(c)\}\}$.
We denote $IFUN$ the set of these instances.

Theorem 19 *$IFUN \cap DBTP \neq \emptyset$ and $IFUN \perp DBTP$.*

Proof In order to prove that the intersection is not empty, we consider a CSP instance with four monovalent variables $x_1, \ldots, x_4$ and three ternary constraints c_1, c_2 and c_3 s.t. $S(c_1) = \{x_1, x_2, x_3\}$, $R(c_1) = \{(v_1, v_2, v_3)\}$, $S(c_2) = \{x_1, x_2, x_4\}$, $R(c_2) = \{(v_1, v_2, v_4)\}$, $S(c_3) = \{x_2, x_3, x_4\}$ and $R(c_3) = \{(v_2, v_3, v_4)\}$. This instance is incrementally functional (using the numeration of variables as ordering) and DBTP.

The instance of Fig. 9a is incrementally functional but not DBTP (b). Conversely, any DBTP instance having several solutions cannot be incrementally functional. □

Definition 19 (Dual maximal clique bounded [10]) A CSP instance P is said **dual maximal clique bounded (DMCB)** if the number of maximal cliques in the micro-structure of its dual instance is polynomial w.r.t. the size of P.
We denote DCL the set of these instances.

Theorem 20 *$DCL \cap DBTP \neq \emptyset$ and $DCL \perp DBTP$.*

Proof Let us consider the first instance defined in the proof of Theorem 19. The micro-structure of its dual instance has a single maximal clique and the instance is DBTP. So, the intersection is not empty. Regarding the instance depicted in Fig. 9b, it has a polynomial number of maximal cliques in the micro-structure of its dual instance but is not DBTP. Conversely, a binary instance whose constraint graph is a star and for which each domain has several values is DBTP but has an unbounded number of maximal cliques in the micro-structure of its dual. □

Now we introduce a new tractable class based on a constraint language restriction.

Definition 20 (Triangular) A CSP instance $P = (X, D, C)$ is said **triangular** w.r.t. a constraint ordering $\prec$ iff $\forall c_i, c_j, c_k$, $c_i \prec c_j \prec c_k$, $\forall t_i \in R(c_i), t_j \in R(c_j), t_k \in R(c_k)$, if $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$ and $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$ then, $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$.
We denote TR the set of triangular instances.

Theorem 21 *If a CSP instance is triangular w.r.t. $\prec$, then it satisfies DBTP w.r.t. $\prec$.*

Proof Assume that P is triangular but not DBTP. So, there exist three constraints c_i, c_j and c_k, $c_i \prec c_j \prec c_k$, $t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$ s.t. $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$, $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$, $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$, $t'_k[S(c_i) \cap S(c_k)] \neq t_i[S(c_i) \cap S(c_k)]$ and $t_j[S(c_j) \cap S(c_k)] \neq t_k[S(c_j) \cap S(c_k)]$.
As P is triangular w.r.t. $\prec$, we must have $t'_k[S(c_i) \cap S(c_k)] = t_i[S(c_i) \cap S(c_k)]$ and $t_j[S(c_j) \cap S(c_k)] = t_k[S(c_j) \cap S(c_k)]$, what is not possible since P does not satisfy DBTP. □

Theorem 22 *$TR \subsetneq DBTP$.*

Proof Theorem 21 shows that $TR \subseteq DBTP$. The instance depicted in Fig. 9c satisfies DBTP but is not triangular. □

Regarding classes based on restricted structures, we have proved in Theorems 10 and 11 that $\beta\text{-}ACYCLIC \subsetneq DBTP$, $\alpha\text{-}ACYCLIC \cap DBTP \neq \emptyset$ and $\alpha\text{-}ACYCLIC \perp DBTP$. Another important tractable class based on restricted structure is related to the tree-width. We first recall the notion of tree-decomposition of graphs [24].

Definition 21 (Tree-decomposition) A **tree-decomposition** of a graph $G = (X, E)$ is a pair (N, T) where $T = (I, F)$ is a tree with nodes I and edges F and $N = \{N_i : i \in I\}$ is a family of subsets of X, s.t. each subset N_i is a node of T and verifies:

(i) $\cup_{i \in I} N_i = X$,
(ii) for each edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subseteq N_i$, and
(iii) for all $i, j, k \in I$, if k is in a path from i to j in T, then $N_i \cap N_j \subseteq N_k$.

The width w of a tree-decomposition (N, T) is equal to $max_{i \in I}|N_i| - 1$. The **tree-width** w of G is the minimal width over all the tree-decompositions of G.

Classically, this definition is extended to hypergraphs by considering the notion of primal graphs. The primal graph of a hypergraph (X, E) is the graph (X, E') where $E' = \{\{x, y\} | \exists e \in E \; s.t. \; x, y \in e\}$.

Definition 22 (Bounded tree-width) Let k be a fixed positive integer. The class BTW_k is the set of the instances whose tree-width is bounded by k.

Theorem 23 $BTW_1 \subsetneq DBTP$.
For $k > 1$, $BTW_k \cap DBTP \neq \emptyset$ *and* $BTW_k \perp DBTP$.

Proof It is well known that BTW_1 is the set of tree-structured binary CSP instances. So according to Theorem 7, we have $BTW_1 \subsetneq DBTP$.

For $k > 1$, as $BTW_1 \subsetneq BTW_k$, the intersection $BTW_k \cap DBTP$ is not empty. Now, let us consider an instance having n variables with $n \geq 3$, whose tree-width is bounded by a constant $k \geq 2$ and which contains the subproblem depicted in Fig. 9a. This instance has a bounded tree-width but does not satisfy DBTP. Conversely, any instance having n variables and one constraint of arity n is DBTP but has an unbounded tree-width. Hence $BTW_k \perp DBTP$. □

5 Links between (D)BTP and directional (Hyper-)k-consistency

In this section, we study the links existing between (D)BTP and Directional (Hyper-)k-Consistency. Such a study is quite natural, since, as pointed in [8], BTP is a weaker form of hyper-3-consistency [22]. We recall the definition of the Hyper-k-Consistency:

Definition 23 (Hyper-**k**-Consistency [22]) Given a CSP instance $P = (X, D, C)$ and an integer k s.t. $1 \leq k \leq e$, P satisfies the hyper-k-consistency if, for every subset $\{c_1, c_2, \ldots c_{k-1}, c_k\}$ of k constraints, we have

$$\bowtie_{i=1}^{k-1} R(c_i) \left[\left(\bigcup_{i=1}^{k-1} S(c_i) \right) \cap S(c_k) \right] \subseteq R(c_k) \left[\left(\bigcup_{i=1}^{k-1} S(c_i) \right) \cap S(c_k) \right]$$

As the hyper-3-consistency implies BTP, we can consider that the hyper-k-consistency is a too strong property. Hence, we define a weaker form by taking into account an ordering over the constraints:

Definition 24 (Directional Hyper-k-Consistency) Given a CSP instance $P = (X, D, C)$, a constraint ordering $\prec$ and an integer k s.t. $1 \leq k \leq e$, P satisfies the directional hyper-k-consistency if for every subset of k constraints s.t. $c_1 \prec c_2 \prec \cdots \prec c_{k-1} \prec c_k$,

$$\bowtie_{i=1}^{k-1} R(c_i)\left[\left(\bigcup_{i=1}^{k-1} S(c_i)\right) \cap S(c_k)\right] \subseteq R(c_k)\left[\left(\bigcup_{i=1}^{k-1} S(c_i)\right) \cap S(c_k)\right]$$

It is easy to see that directional hyper-k-consistency is a weaker form of hyper-k-consistency. For example, let us consider an instance with three constraints, c_1, c_2 and c_3 such that:

- $S(c_1) = \{x_1, x_2, x_3\}$ and $R(c_1) = \{(a, b, c)\}$,
- $S(c_2) = \{x_2, x_3, x_4\}$ and $R(c_2) = \{(b, c, d)\}$,
- $S(c_3) = \{x_1, x_4, x_5\}$ and $R(c_3) = \{(a, d, e), (a, f, e)\}$.

With the ordering $c_1 \prec c_2 \prec c_3$, this instance satisfies directional hyper-3-consistency since $R(c_1) \bowtie R(c_2)[(S(c_1) \cup S(c_2)) \cap S(c_3)] = R(c_1) \bowtie R(c_2)[\{x_1, x_4\}] = \{(a, d)\} \subseteq R(c_3)[\{x_1, x_4\}] = \{(a, d), (a, f)\}$, while hyper-3-consistency is not verified since $R(c_1) \bowtie R(c_3)[(S(c_1) \cup S(c_3)) \cap S(c_2)] = R(c_1) \bowtie R(c_3)[\{x_2, x_3, x_4\}] = \{(b, c, d), (b, c, f)\}$ which is not a subset of $R(c_2)[(S(c_1) \cup S(c_3)) \cap S(c_2)] = \{(b, c, d)\}$.

Theorem 24 *If a CSP instance P satisfies DBTP w.r.t. a constraint ordering $\prec$ and is directional hyper-k-consistent w.r.t. $\prec$ for $2 \leq k < e$, then P is directional hyper-$(k+1)$-consistent w.r.t. $\prec$.*

Proof Assume that P is not hyper-$(k+1)$-consistent. So, there is a subset of $k+1$ constraints s.t. $c_1 \prec \cdots \prec c_k \prec c_{k+1}$, $\exists(t_1, \ldots, t_k) \in R(c_1) \times \ldots \times R(c_k)$, $\forall t_{k+1} \in R(c_{k+1})$, $\bowtie_{i=1}^{k} t_i[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})] \neq t_{k+1}[(\bigcup_{i=1}^{k} S(c_i)) \cap S(c_{k+1})]$.
Let us consider the k subsets of k constraints $\{c_1, \ldots c_{j-1}, c_{j+1}, \ldots c_k, c_{k+1}\}$, for $1 \leq j \leq k$. As P is directional hyper-k-consistent, for $1 \leq j \leq k$, there exists a tuple t_{k+1}^{j} of $R(c_{k+1})$ s.t. $\bowtie_{i=1, i\neq j}^{k} t_i[(\bigcup_{i=1, i\neq j}^{k} S(c_i)) \cap S(c_{k+1})] = t_{k+1}^{j}[(\bigcup_{i=1, i\neq j}^{k} S(c_i)) \cap S(c_{k+1})]$.
Consider $1 \leq j < j' \leq k$. We have two cases:

(1) $t_{k+1}^{j} = t_{k+1}^{j'}$. Then $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] = t_{k+1}^{j}[S(c_{j'}) \cap S(c_{k+1})]$ and $t_j[S(c_j) \cap S(c_{k+1})] = t_{k+1}^{j'}[S(c_j) \cap S(c_{k+1})]$.
So $\bowtie_{i=1}^{k} t_i\left[\left(\bigcup_{i=1}^{k} S(c_i)\right) \cap S(c_{k+1})\right] = t_{k+1}^{j}\left[\left(\bigcup_{i=1}^{k} S(c_i)\right) \cap S(c_{k+1})\right]$ and we have a contradiction.

(2) $t_{k+1}^{j} \neq t_{k+1}^{j'}$. We have $t_j[S(c_j) \cap S(c_{j'})] = t_{j'}[S(c_j) \cap S(c_{j'})]$, $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] = t_{k+1}^{j}[S(c_{j'}) \cap S(c_{k+1})]$, $t_j[S(c_j) \cap S(c_{k+1})] = t_{k+1}^{j'}[S(c_j) \cap S(c_{k+1})]$, $t_{j'}[S(c_{j'}) \cap S(c_{k+1})] \neq t_{k+1}^{j'}[S(c_{j'}) \cap S(c_{k+1})]$ and $t_j[S(c_j) \cap S(c_{k+1})] \neq t_{k+1}^{j}[S(c_j) \cap S(c_{k+1})]$. Hence P does not satisfy DBTP w.r.t. $\prec$ and we have again a contradiction.

So, P is directional hyper-$(k+1)$-consistent w.r.t. the order $\prec$. □

As the pairwise-consistency corresponds to the hyper-2-consistency, we can deduce this corollary:

Corollary 1 *If a CSP instance P satisfies DBTP w.r.t. a constraint ordering $\prec$ and is directional pairwise-consistent w.r.t. $\prec$, then P is directional hyper-$(k+1)$-consistent w.r.t. $\prec$ for $1 \leq k < e$.*

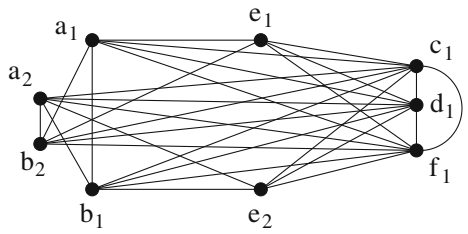


Fig. 10 Part of an instance satisfying directional hyper-3-consistency but not BTP

So a CSP instance which is both DBTP and directional pairwise-consistent w.r.t. a given constraint ordering is consistent. It also ensues that such a CSP instance satisfies the directional hyper-3-consistency.

Moreover, as the hyper-k-consistency corresponds to the k-consistency on the dual problem, we can also derive the following theorem and corollary by achieving a similar reasoning.

Theorem 25 *If a binary CSP instance P satisfies BTP w.r.t. a variable ordering $<$ and is directional k-consistent w.r.t. $<$ for $2 \leq k < n$, then P is directional $(k+1)$-consistent w.r.t. $<$.*

Corollary 2 *If a binary CSP instance P satisfies BTP w.r.t. a variable ordering $<$ and is directional arc-consistent (DAC) w.r.t. $<$, then P is directional $(k+1)$-consistent w.r.t. $<$ for $1 \leq k < n$.*

As a consequence, a binary CSP instance which is both BTP and DAC w.r.t. a given variable ordering is consistent.

Regarding the positioning of the directional hyper-3-consistency w.r.t. to BTP, we can prove that the directional hyper-3-consistency does not necessarily imply BTP. For example, we can consider a binary instance with 6 variables $\{x_a, x_b, \ldots, x_f\}$. We define this instance by reproducing several times a same pattern s.t. each value appearing in an instance of the pattern does not appear in any other instance. This pattern consists in a broken triangle on a variable z for a triple (x, y, z) (i.e. which imposes the condition $z < max(x, y)$ on $<$) and each value of the variables x, y and z is linked to a given value of any variable which is not involved in this triple. We reproduce this pattern 6 times s.t. the following conditions are imposed:

- $x_a < max(x_b, x_c)$,
- $x_b < max(x_d, x_e)$,
- $x_c < max(x_e, x_f)$,
- $x_d < max(x_a, x_b)$,
- $x_e < max(x_a, x_b)$ and
- $x_f < max(x_a, x_b)$.

Figure 10 depicts this pattern for the triples (x_a, x_b, x_e), a broken triangle on x_e (corresponding to the condition $x_e < max(x_a, x_b)$) and the independent variables x_c, x_d and x_f. By doing this, the micro-structure of our binary CSP instance has 6 connected components. We can note that this instance is not BTP because the 6 conditions make impossible the construction of a suitable variable ordering. Nevertheless, it is directional hyper-3-consistent and arc-consistent.

Constraints (2015) 20:383–413 405

6 DBTP from a practical viewpoint

In this section, we study the DBTP property from a practical viewpoint. First, we show that some benchmarks which are classically used for solver comparisons are DBTP. Then, for these benchmarks, we highlight the consequences on the solving efficiency.

6.1 Benchmarks which are DBTP

Checking whether an instance P has the DBTP property can be achieved by testing the existence of a constraint ordering $\prec$ s.t. P is DBTP w.r.t. $\prec$. To do this, the proof of Lemma 2 suggests a method which consists in computing first the dual of P and then checking if a constraint ordering s.t. the dual of P is BTP exists like in [8]. However, in practice, computing the dual may require a prohibitive amount of memory. It is especially the case if the constraints allow a large number of tuples. Moreover, if the constraints are expressed in intention (e.g. by predicates), the memory space required for representing the dual instance may be greatly larger than one for representing the original instance. We can avoid building the dual by considering it in a virtual manner and by exploiting the alternative characterization of DBTP provided in Theorem 3. Indeed, in this characterization, we only need to manage the interactions between tuples of constraints. These interactions are directly expressed in the dual instance but can also be deduced on the fly by taking into account the intersection between tuples of different constraints. The second step can then be achieved by building and solving a max-closed instance P^o with a variable o_i (with domain $\{1, \ldots, e\}$) per constraint c_i of P and a constraint imposing $o_k < max(o_i, o_j)$ for all triples of constraints (c_i, c_j, c_k) s.t. $\exists t_i \in R(c_i)$, $t_j \in R(c_j)$ and $t_k, t'_k \in R(c_k)$, $t_i[S(c_i) \cap S(c_j)] = t_j[S(c_i) \cap S(c_j)]$, $t_i[S(c_i) \cap S(c_k)] = t_k[S(c_i) \cap S(c_k)]$, $t'_k[S(c_j) \cap S(c_k)] = t_j[S(c_j) \cap S(c_k)]$, $t'_k[S(c_i) \cap S(c_k)] \neq t_i[S(c_i) \cap S(c_k)]$ and $t_j[S(c_j) \cap S(c_k)] \neq t_k[S(c_j) \cap S(c_k)]$.

Algorithm 1 implements this method. Given a CSP instance P, it returns *true* if P satisfies the DBTP property, false otherwise. Lines 1-8 are devoted to the construction of the max-closed instance P^o while Line 9 consists in solving this instance in polynomial time. Note that, if we consider the recognition method used in the proof of Lemma 2, checking whether the dual of P is BTP requires to build exactly the same max-closed instance as P^o, what ensures the validity of Algorithm 1.

Algorithm 1 Is_DBTP

```
   Input: a CSP instance P = (X, D, C)
   Output: Boolean
 1 X^o ← {x^o_1, ..., x^o_e}
 2 D^o ← {d^o_1, ..., d^o_e | ∀1 ≤ i ≤ e, d^o_i = {1, ..., e}}
 3 C^o ← ∅
 4 foreach c_i, c_j, c_k ∈ C s.t. c_i ≠ c_j, c_i ≠ c_k and c_j ≠ c_k do
 5     if ∃t_i ∈ R(c_i), t_j ∈ R(c_j), t_k, t'_k ∈ R(c_k) s.t. t_i[S(c_i) ∩ S(c_j)] = t_j[S(c_i) ∩ S(c_j)],
 6     t_i[S(c_i) ∩ S(c_k)] = t_k[S(c_i) ∩ S(c_k)], t'_k[S(c_j) ∩ S(c_k)] = t_j[S(c_j) ∩ S(c_k)],
 7     t'_k[S(c_i) ∩ S(c_k)] ≠ t_i[S(c_i) ∩ S(c_k)] and t_j[S(c_j) ∩ S(c_k)] ≠ t_k[S(c_j) ∩ S(c_k)] then
 8         C^o ← C^o ∪ {c|S(c) = {o_i, o_j, o_k} and R(c) = {o_k < max(o_i, o_j)}}
 9 if the CSP instance P^o = (X^o, D^o, C^o) has a solution then
10     return true
11 else
12     return false
```

We are interested here in the 7,272 benchmark instances from the CSP 2008 Competition[2]. These instances have binary or non-binary constraints which are represented in extension (by list of allowed tuples or list of forbidden tuples) or in intention (by predicate or global constraints). Most of the tractable classes we consider require that the constraints are expressed in extension by lists of allowed tuples in order to guarantee a polynomial recognition. Their recognition (including one of DBTP) may take from a few seconds to several hours (notably for instances having constraints in intention with large arity). So, for our experimentations, we exclude the instances containing constraints in intention with large arity (to ensure a reasonable runtime) or global constraints (because our CSP library does not take them into account yet). At the end, we have considered 2,800 instances for which we have checked the belonging to the classes $DBTP$ and BTP but also to some of the classes previously introduced, namely $TREE$, β-$ACYCLIC$, ZOA, CM, CCM, $IFUN$, MC, TR and BTW_k.

If we check the property directly on the original instances, the only instances that are detected as DBTP are those which are acyclic or β-acyclic. We find 8 binary instances with an acyclic constraint graph (e.g. all the instances of the `hanoi` family) and 23 non-binary instances with a β-acyclic constraint hypergraph.

Tables 1 and 2-3 provide respectively the list of the binary and non-binary instances which are detected as DBTP after having enforced the arc-consistency. They also indicate the value of some parameters of the instances and their tree-width[3] w or a range of values when the exact value is unknown (we recall that computing w is NP-hard). Table 1 presents the belonging of the original instances to the classes $DBTP$, MC, ZOA and CM (i.e. before enforcing arc-consistency) while Tables 2 and 3 do the same for the classes $DBTP$ and MC before the filtering and for the class MC after having enforced the arc-consistency.

The absence of instances for which DBTP holds thanks to relational properties is partially due to the presence of useless values in the domains and so to useless tuples in relations. To avoid this problem, a possible solution consists in simplifying the instances by enforcing some level of consistency (in the same spirit of *hidden tractable classes* introduced in [12]), namely here the arc-consistency. The choice of the arc-consistency is quite natural since most solvers exploit a level of consistency at least as powerful as the arc-consistency. By so doing, 362 instances belong trivially to DBTP or to any other relational or hybrid tractable classes since they are detected as inconsistent and so all the domains are empty (we assume that the filtering process is not stopped as soon as a domain becomes empty). We have found 105 instances which are arc-consistent and DBTP.

For binary instances, 37 instances have been detected as DBTP. We can note that all the instances of families `domino` and `hanoi` are DBTP after an AC filtering. Moreover, for the instances from families `hanoi` and the instances `graph12-w0` and `graph13-w0`, the DBTP property holds because these instances have an acyclic constraint graph. Finally, we have observed that, once the arc-consistency has been enforced, all the instances in Table 1 also belong to BTP, ZOA and TR while all of them except `graph12-w0` and `graph13-w0` are in CM, CC, $IFUN$ and MC. However, all the instances of the competition which are BTP are not necessarily DBTP. For example, the instance `fapp17-0300-10` is BTP, but not DBTP. Regarding the class BTW_k, the instances `large-*` can be assimilated to instances having an unbounded tree-width (even if formally,

[2] See http://www.cril.univ-artois.fr/CPAI08formoredetails.

[3] We recall that this notion is extended to hypergraphs by considering the notion of primal graphs.

Table 1 List of binary instances which are detected as DBTP after having enforced the arc-consistency and for each instance, the values of some parameters, its tree-width w and its belonging or not to the classes $DBTP$, MC, ZOA and CM before enforcing the arc-consistency

Instance	n	e	d	w	$DBTP$	MC	ZOA	CM
domino-100-100	100	100	100	2	no	no	no	yes
domino-100-200	100	100	200	2	no	no	no	yes
domino-100-300	100	100	300	2	no	no	no	yes
domino-1000-100	1,000	1,000	100	2	no	no	no	yes
domino-1000-1000	1,000	1,000	1,000	2	no	no	no	yes
domino-1000-200	1,000	1,000	200	2	no	no	no	yes
domino-1000-300	1,000	1,000	300	2	no	no	no	yes
domino-1000-500	1,000	1,000	500	2	no	no	no	yes
domino-1000-800	1,000	1,000	800	2	no	no	no	yes
domino-2000-2000	2,000	2,000	2,000	2	no	no	no	yes
domino-300-100	300	300	100	2	no	no	no	yes
domino-300-200	300	300	200	2	no	no	no	yes
domino-300-300	300	300	300	2	no	no	no	yes
domino-3000-3000	3,000	3,000	3,000	2	no	no	no	yes
domino-500-100	500	500	100	2	no	no	no	yes
domino-500-200	500	500	200	2	no	no	no	yes
domino-500-300	500	500	300	2	no	no	no	yes
domino-500-500	500	500	500	2	no	no	no	yes
domino-5000-5000	5,000	5,000	5,000	2	no	no	no	yes
domino-800-100	800	800	100	2	no	no	no	yes
domino-800-200	800	800	200	2	no	no	no	yes
domino-800-300	800	800	300	2	no	no	no	yes
domino-800-500	800	800	500	2	no	no	no	yes
domino-800-800	800	800	800	2	no	no	no	yes
hanoi-3_ext	6	5	27	1	yes	no	no	no
hanoi-4_ext	14	13	81	1	yes	no	no	no
hanoi-5_ext	30	29	243	1	yes	no	no	no
hanoi-6_ext	62	61	729	1	yes	no	no	no
hanoi-7_ext	126	125	2,187	1	yes	no	no	no
large-80-sat_ext	80	3,160	80	79	no	no	no	no
large-84-sat_ext	84	3,486	84	83	no	no	no	no
large-88-sat_ext	88	3,828	88	87	no	no	no	no
large-92-sat_ext	92	4,186	92	91	no	no	no	no
large-96-sat_ext	96	4,560	96	95	no	no	no	no
mps-diamond	2	1	2	1	yes	yes	yes	yes
graph12-w0	680	340	44	1	yes	yes	no	no
graph13-w0	916	458	44	1	yes	yes	no	no

Table 2 List of non-binary instances which are detected as DBTP after having enforced the arc-consistency and for each instance, the values of some parameters, its tree-width w (or a range) and its belonging or not to the classes $DBTP$ and MC before enforcing the arc-consistency and to the class MC after having enforced the arc-consistency

Instance	n	e	d	r	w	$DBTP$	MC	MC after AC
mknap-1-0	6	1	2	6	5	yes	yes	yes
mknap-1-2	15	1	2	15	14	yes	yes	yes
mknap-1-3	20	1	2	20	19	yes	yes	yes
mknap-1-4	28	1	2	28	27	yes	yes	yes
mknap-1-5	39	1	2	39	38	yes	yes	yes
mknap-1-6	50	1	2	50	49	yes	yes	yes
primes-10-20-2-1	100	20	28	3	2	yes	no	no
primes-10-20-3-1	100	20	28	4	3	yes	no	no
primes-10-40-2-1	100	40	28	3	2	no	no	no
primes-10-40-3-1	100	40	28	4	[3,9]	no	no	no
primes-10-60-2-1	100	60	28	3	[2,5]	no	no	no
primes-10-60-2-3	100	60	28	5	[4,21]	no	no	no
primes-10-60-2-5	100	60	28	7	[6,35]	no	no	no
primes-10-60-3-1	100	60	28	4	[3,20]	no	no	no
primes-10-80-2-1	100	80	28	3	[2,8]	no	no	yes
primes-10-80-2-3	100	80	28	5	[4,27]	no	no	no
primes-10-80-2-5	100	80	28	7	[6,42]	no	no	yes
primes-10-80-3-1	100	80	28	4	[3,27]	no	no	no
primes-10-80-3-3	100	80	28	6	[5,42]	no	no	yes
primes-15-20-2-1	100	20	46	3	2	yes	no	no
primes-15-20-3-1	100	20	46	4	3	yes	no	no
primes-15-40-2-1	100	40	46	3	2	no	no	no
primes-15-40-2-3	100	40	46	5	[4,10]	no	no	no
primes-15-40-3-1	100	40	46	4	[3,9]	no	no	no
primes-15-60-2-1	100	60	46	3	[2,5]	no	no	yes
primes-15-60-2-3	100	60	46	5	[4,21]	no	no	no
primes-15-60-3-1	100	60	46	4	[3,20]	no	no	no
primes-15-60-3-3	100	60	46	6	[5,34]	no	no	no
primes-15-80-2-1	100	80	46	3	[2,8]	no	no	yes
primes-15-80-2-3	100	80	46	5	[4,27]	no	no	no
primes-15-80-3-1	100	80	46	4	[3,27]	no	no	no
primes-15-80-3-3	100	80	46	6	[5,42]	no	no	yes
primes-20-20-2-1	100	20	70	3	2	yes	no	no
primes-20-20-3-1	100	20	70	4	3	yes	no	no

A dash means that the computation cannot be achieved due to an expensive runtime

Table 3 List of non-binary instances (Table 2 continued) which are detected as DBTP after having enforced the arc-consistency and for each instance, the values of some parameters, its tree-width w (or a range) and its belonging or not to the classes $DBTP$ and MC before enforcing the arc-consistency and to the class MC after having enforced the arc-consistency

Instance	n	e	d	r	w	$DBTP$	MC	MC after AC
primes-20-40-2-1	100	40	70	3	2	no	no	no
primes-20-40-3-1	100	40	70	4	[3,9]	no	no	no
primes-20-60-2-1	100	60	70	3	[2,5]	no	no	no
primes-20-60-2-3	100	60	70	5	[4,21]	no	no	no
primes-20-60-3-1	100	60	70	4	[3,20]	no	no	no
primes-20-80-2-1	100	80	70	3	[2,8]	no	no	yes
primes-20-80-2-3	100	80	70	5	[4,27]	no	no	no
primes-20-80-3-1	100	80	70	4	[3,27]	no	no	no
primes-25-20-2-1	100	20	96	3	2	yes	no	no
primes-25-20-3-1	100	20	96	4	3	yes	no	no
primes-25-40-2-1	100	40	96	3	2	no	no	no
primes-25-40-3-1	100	40	96	4	[3,9]	no	no	no
primes-25-60-2-1	100	60	96	3	[2,5]	no	no	no
primes-25-60-2-3	100	60	96	5	[4,21]	no	no	no
primes-25-60-3-1	100	60	96	4	[3,20]	no	no	no
primes-25-80-2-1	100	80	96	3	[2,8]	no	no	yes
primes-25-80-2-3	100	80	96	5	[4,27]	no	no	no
primes-25-80-3-1	100	80	96	4	[3,27]	no	no	no
primes-30-20-2-1	100	20	112	3	2	yes	no	no
primes-30-20-3-1	100	20	112	4	3	yes	no	no
primes-30-40-2-1	100	40	112	3	[2,2]	no	no	no
primes-30-40-3-1	100	40	112	4	[3,9]	no	no	no
primes-30-60-2-1	100	60	112	3	[2,5]	no	no	no
primes-30-60-3-1	100	60	112	4	[3,20]	no	no	no
primes-30-80-2-1	100	80	112	3	[2,8]	no	no	yes
primes-30-80-3-1	100	80	112	4	[3,27]	no	no	no
mps-sentoy	60	1	2	60	59	yes	no	-
mps-red-markshare1-1	280	11	2	130	129	yes	no	-
mps-red-markshare1	230	6	2	80	79	yes	no	-
mps-red-markshare2	270	7	2	90	89	yes	no	-
mps-red-blend2	2,944	196	2	2,659	2,658	yes	no	-
mps-red-est	146	4	2	74	73	yes	no	-
mps-red-markshare2-1	330	13	2	150	149	yes	no	-

A dash means that the computation cannot be achieved due to an expensive runtime

Table 4 List of binary and non-binary families for which none instance is DBTP

binary		non binary
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		

it is not the case since they are instances) since their tree-width is equal to their number of variables minus one.

Regarding non-binary instances, the instances of the family `mknap` are trivially DBTP and β-acyclic since each one contains a single constraint. Then, we can observe that more than 33 % of instances from the `primes-*` families are DBTP. For ten of these instances, the DBTP property holds because they have a β-acyclic constraint hypergraph. It follows that the remaining instances are DBTP thanks to the features of their relations. So the `primes-*` families illustrate perfectly the fact that the class DBTP is hybrid. We note that all the DBTP instances are also triangular like in the binary case while none is incrementally functional. The instances of the `mknap` family belong to MC. It is the same for some instances of the `prime-*` families once the arc-consistency is enforced. Finally, the tree-width of the DBTP instances is not necessarily bounded due to the arbitrary arity of constraints. For example, the tree-width of the `mknap-1-6` is equal to its number of variables minus one.

Table 4 provides the list of families for which no instance is DBTP, what corresponds to 1,640 instances. The other instances which are not DBTP belong to families for which at least one instance is DBTP or has an unknown status. At the end, the DBTP instances only represent about 4 % of the considered instances (17 % if we also consider the instances which are arc-inconsistent and so trivially DBTP). However, on the one hand, this shows

that this tractable class is not artificial, and on the other hand, their membership of DBTP will make it possible to explain the efficiency of their solving in the next part.

6.2 Links with the solving

Generally, the instances of a given tractable class are solved thanks to a specific algorithm which is dedicated to this particular tractable class. Here, our aim is not to solve the DBTP instances with a specific algorithm, but to exploit the DBTP property to explain why some instances are solved efficiently by classical algorithms like MAC [26] or RFL [23], on which most solvers of the state of the art rely. Of course, we focus our study on instances which are both DBTP and arc-consistent.

For the binary instances which are DBTP and arc-consistent, Theorem 5 states that MAC solves these instances in polynomial time. In practice, MAC turns to be very efficient since it solves all the instances listed in Table 1 in a backtrack-free manner.

Regarding the non-binary instances, the 54 DBTP instances of the `primes-*` families are also solved efficiently by MAC in a backtrack-free manner (except two which require a single backtrack). For 10 instances, the size of the largest intersection between the scopes of two constraints does not exceed one. So, Theorem 6 holds, what explains the solving efficiency we observe. However, this theorem concerns a particular case where the arc-consistency of the instance entails its pairwise-consistency. Other cases may exist depending on the features of relations. For 38 instances, enforcing the arc-consistency entails the pairwise-consistency at each step of the search and so MAC solves them in polynomial time as stated by Theorem 5. For 6 instances, enforcing the arc-consistency is unable to entail the pairwise-consistency, the membership of the class DBTP is not a sufficient reason to explain the efficient solving.

The `mps-sentoy` instance and the three first instances of the `mknap` family are solved efficiently in a backtrack-free manner. As they have a single constraint, Theorem 5 holds, what explains the efficient solving. In contrast, for the other instances of the `mknap` family or the instances of the `mps` family, MAC does not succeed in solving them efficiently. The explanation of this phenomenon is related to the assumption that the relations are expressed in extension and to the large value of the arity of the constraint. If, the violation of this assumption often has no consequence on the efficiency of MAC applied on DBTP instances (all the instances of Tables 2 and 3 have constraints defined by predicates), it is not the case here.

Finally, note that we have made the same observations when using RFL instead of MAC, what was foreseeable since Theorems 5 and 6 also hold for RFL.

7 Conclusion and future works

In this paper, we have studied a hybrid tractable class whose instances can be solved in polynomial time by MAC-like algorithms. We have then proved that it is incomparable with several known tractable classes (notably BTP) and that it captures both structural and relational tractable classes (namely β-acyclic CSPs and Triangular CSPs). We have also compared DBTP with the Directional Hyper-k-Consistency, which led us to present new results for BTP. Finally, we have analysed DBTP from a practical point of view. This study has shown that DBTP is not an artificial tractable class since several classical benchmarks among the ones used in the CSP 2008 Competition belong to DBTP. Moreover, for most of

these instances, their membership of DBTP allows us to explain their ability to be efficiently solved by solvers of the state of the art based on algorithms like MAC or RFL.

A first extension consists in studying the link between DBTP and other tractable classes we have not mentioned in this paper. Another one consists in considering other properties and then in extending other tractable classes to non-binary CSPs using a similar approach, using the dual representation. One interesting candidate could be the *min-of-max extendable* property also introduced in [8].

Then, in the same spirit, we can also explore the possibility of defining new tractable classes, taking properties like BTP (or some others) and exploiting other encodings of non-binary CSPs.

Acknowledgements This work was supported by the French National Research Agency under grant TUPLES (ANR-2010-BLAN-0210).

The authors would like to thank Martin C. Cooper for his useful comments.

References

1. van Beek, P., & Dechter, R. (1995). On the minimality and decomposability of row-convex constraint networks. *Journal of the ACM*, *42*(3), 543–561.
2. Beeri, C., Fagin, R., Maier, D., & Yannakakis, M. (1983). On the desirability of acyclic database schemes. *Journal of the ACM*, *30*(3), 479–513.
3. Bessière, C. (2006). *Constraint Propagation, chap. 3. Handbook of Constraint Programming*: Elsevier.
4. Bessière, C., Stergiou, K., & Walsh, T. (2008). Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, *172*, 800–822.
5. Cohen, D., Cooper, M., Green, M., & Marx, D. (2011). On guaranteeing polynomially bounded search tree size. In *Proceedings of CP* (pp. 160–171).
6. Cohen, D.A. (2003). A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *Proceedings of CP 2003* (pp. 807–811).
7. Cooper, M., Cohen, D., & Jeavons, P. (1994). Characterising Tractable Constraints. *Artificial Intelligence*, *65*(2), 347–361.
8. Cooper, M., & Jeavons, P. (2010). Salamon, A.: Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination. *Artificial Intelligence*, *174*, 570–584.
9. Duris, D. (2012). Some characterizations of γ and β-acyclicity of hypergraphs. *Information Processing Letters*, *112*(16), 617–620.
10. El Mouelhi, A., Jégou, P., Terrioux, C., & Zanuttini, B. (2013). Some New Tractable Classes of CSPs and their Relations with Backtracking Algorithms. In *Proceedings of CP-AI-OR* (pp. 61–76).
11. El Mouelhi, A., Jégou, P., & Terrioux, C. (2013). A Hybrid Tractable Class for Non-Binary CSPs. In *Proceedings of ICTAI* (pp. 947–954).
12. El Mouelhi, A., Jégou, P., & Terrioux, C. (2014). Hidden Tractable Classes: from Theory to Practice. In *Proceedings of ICTAI* (pp. 437–445).
13. Fagin, R. (1983). Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. *Journal of the ACM*, *30*(3), 514–550.
14. Freuder, E. (1982). A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, *29*(1), 24–32.
15. Gavril, F. (1972). 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.
16. Golumbic, M. (1980). *Algorithmic Graph Theory and Perfect Graphs*. New York: Academic Press.
17. Graham, M.H. (1979). On the universal relation. Tech. rep., University of Toronto.
18. Janssen, P., Jégou, P., Nouguier, B., & Vilarem, M.C. (1989). 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* (pp. 420–427).
19. Jeavons, P., & Cooper, M. (1995). Tractable constraints on ordered domains. *Artificial Intelligence*, *79*(2), 327–339.

20. Jégou, P. (1991). Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution – Propagation de contraintes dans les réseaux dynamiques. Ph.D. thesis, Université des Sciences et Techniques du Languedoc.
21. Jégou, P. (1993). Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *Proceedings of AAAI* (pp. 731–736).
22. Jégou, P. (1993). On the Consistency of General Constraint-Satisfaction Problems. In *Proceedings of AAAI* (pp. 114–119).
23. Nadel, B. (1988). Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms, pp. 287–342. In *Search in Artificial Intelligence*: Springer-Verlag.
24. Robertson, N., & Seymour, P. (1986). Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, *7*, 309–322.
25. Rossi, F., van Beek, P., & Walsh, T. (2006). *Handbook of Constraint Programming*: Elsevier.
26. Sabin, D., & Freuder, E. (1994). Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI* (pp. 125–129).
27. Salamon, A., & Jeavons, P. (2008). Perfect Constraints Are Tractable. In *Proceedings of CP* (pp. 524–528).

Artificial Intelligence 234 (2016) 196–218

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Broken triangles: From value merging to a tractable class of general-arity constraint satisfaction problems

Martin C. Cooper [a,*], Aymeric Duchein [a], Achref El Mouelhi [b], Guillaume Escamocher [c], Cyril Terrioux [b], Bruno Zanuttini [d]

[a] *IRIT, Université de Toulouse, CNRS, INPT, UPS, UT1, UT2J, France*
[b] *Aix-Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, Marseille, France*
[c] *INSIGHT Centre for Data Analytics, University College Cork, Cork, Ireland*
[d] *GREYC, UMR 6072, Normandie Université, UNICAEN, CNRS, ENSICAEN, Caen, France*

ARTICLE INFO

Article history:
Received 17 September 2015
Received in revised form 1 February 2016
Accepted 3 February 2016
Available online 4 February 2016

Keywords:
CSP
Constraint satisfaction
Domain reduction
Tractable class
Hybrid tractability
NP-completeness
Global constraints

ABSTRACT

A binary CSP instance satisfying the broken-triangle property (BTP) can be solved in polynomial time. Unfortunately, in practice, few instances satisfy the BTP. We show that a local version of the BTP allows the merging of domain values in arbitrary instances of binary CSP, thus providing a novel polynomial-time reduction operation. Extensive experimental trials on benchmark instances demonstrate a significant decrease in instance size for certain classes of problems. We show that BTP-merging can be generalised to instances with constraints of arbitrary arity and we investigate the theoretical relationship with resolution in SAT. A directional version of general-arity BTP-merging then allows us to extend the BTP tractable class previously defined only for binary CSP. We investigate the complexity of several related problems including the recognition problem for the general-arity BTP class when the variable order is unknown, finding an optimal order in which to apply BTP merges and detecting BTP-merges in the presence of global constraints such as AllDifferent.

1. Introduction

At first sight one could assume that the discipline of constraint programming has come of age. On the one hand, efficient solvers are regularly used to solve real-world problems in diverse application domains while, on the other hand, a rich theory has been developed concerning, among other things, global constraints, tractable classes, reduction operations and symmetry. However, there often remains a large gap between theory and practice, which is perhaps most evident when we look at the large number of deep results concerning tractable classes which have yet to find any practical application. The research reported in this paper is part of a long-term project to bridge the gap between theory and practice. Our aim is not only to develop new tools but also to explain why present tools work so well.

Most research on tractable classes has been based on classes defined by placing restrictions either on the types of constraints [1,2] or on the constraint hyper-graph whose vertices are the variables and whose hyper-edges are the constraint scopes [3,4]. Another way of defining classes of binary CSP instances consists of imposing conditions on the microstruc-

* Corresponding author.
E-mail addresses: cooper@irit.fr (M.C. Cooper), Aymeric.Duchein@irit.fr (A. Duchein), achref.elmouelhi@lsis.org (A. El Mouelhi), guillaume.escamocher@insight-centre.org (G. Escamocher), cyril.terrioux@lsis.org (C. Terrioux), bruno.zanuttini@unicaen.fr (B. Zanuttini).

http://dx.doi.org/10.1016/j.artint.2016.02.001
0004-3702/

ture, a graph whose vertices are the possible variable-value assignments with an edge linking each pair of compatible assignments [5,6]. If each vertex of the microstructure, corresponding to a variable-value assignment $\langle x, a\rangle$, is labelled (or coloured) by the variable x, then this so-called coloured microstructure retains all information from the original instance. The broken-triangle property (BTP) is a simple local condition on the coloured microstructure which defines a tractable class of binary CSP [7]. The BTP corresponds to forbidding a simple pattern, known as a broken triangle, in the coloured microstructure for a given variable order. Inspired by the BTP, investigation of other forbidden patterns in the coloured microstructure has led to the discovery of new tractable classes [8–10] as well as new reduction operations based on variable or value elimination [11,12]. The BTP itself has also been directly generalised in several different ways. For example, it has been shown that under an assumption of strong path consistency, the BTP can be considerably relaxed since not all broken triangles need be forbidden to define a tractable class [13–15]. Indeed, even without any assumptions of consistency, it is not necessary to forbid all broken triangles [12]. Imposing the BTP in the dual problem leads directly to a tractable class of general-arity CSPs [16]. The BTP has also been generalised to the Broken Angle Property which defines a tractable class of Quantified Constraint Satisfaction Problems [17].

In this paper we show that the absence of broken triangles on a pair of values in a domain allows us to merge these two values while preserving the satisfiability of the instance. Furthermore, given a solution to the reduced instance, it is possible to find a solution to the original instance in linear time (Section 3). We then investigate the interactions between arc consistency and BTP-merging operations (Section 4) and show that it is NP-hard to find the best sequence of BTP-merging (and arc consistency) operations (Section 5). The effectiveness of BTP-merging in reducing domains in binary CSP benchmark problems is investigated in Section 6. In the second half of the paper we consider general-arity CSPs. Section 7 shows how to generalise BTP-merging to instances containing constraints of any arity (where all constraints are given in the form of either tables, lists of compatible tuples or lists of incompatible tuples). We then go on to consider global constraints, and in particular the AllDifferent constraint, in Section 8. Finally, a directional version of the general-arity BTP allows us to define a tractable class of general-arity CSP instances which is incomparable with the tractable class obtained by directly imposing the BTP in the dual [16] (Section 9). However, on the negative side, we then show that it is NP-complete to determine the existence of a variable order for which an instance falls into this tractable class. The results of Sections 3, 7, 9 and Sections 4, 5 first appeared in two conference papers (respectively [18] and [19]).

2. The Constraint Satisfaction Problem

For simplicity of presentation we use two different representations of constraint satisfaction problems. In the binary case, our notation is fairly standard, whereas in the general-arity case we use a notation close to the representation of SAT instances. This is for presentation only, though, and our algorithms do *not* need instances to be represented in this manner.

Definition 1. A *binary CSP instance* I consists of

- a set X of n variables,
- a domain $\mathcal{D}(x)$ of possible values for each variable $x \in X$,
- a relation $R_{xy} \subseteq \mathcal{D}(x) \times \mathcal{D}(y)$, for each pair of distinct variables $x, y \in X$, which consists of the set of compatible pairs of values (a, b) for variables (x, y).

A *partial solution* to I on $Y = \{y_1, \ldots, y_r\} \subseteq X$ is a set $\{\langle y_1, a_1\rangle, \ldots, \langle y_r, a_r\rangle\}$ such that $\forall i, j \in [1, r]$, $(a_i, a_j) \in R_{y_i y_j}$. A *solution* to I is a partial solution on X.

For simplicity of presentation, Definition 1 assumes that there is exactly one constraint relation for each pair of variables. The number of constraints e is the number of pairs of variables x, y such that $R_{xy} \neq \mathcal{D}(x) \times \mathcal{D}(y)$. An instance I is *arc consistent* if for each pair of distinct variables $x, y \in X$, each value $a \in \mathcal{D}(x)$ has an AC-support at y, i.e. a value $b \in \mathcal{D}(y)$ such that $(a, b) \in R_{xy}$.

In our representation of general-arity CSP instances, we require the notion of *tuple* which is simply a set of variable-value assignments. For example, in the binary case, the tuple $\{\langle x, a\rangle, \langle y, b\rangle\}$ is *compatible* if $(a, b) \in R_{xy}$ and *incompatible* otherwise.

Definition 2. A *(general-arity) CSP instance* I consists of

- a set X of n variables,
- a domain $\mathcal{D}(x)$ of possible values for each variable $x \in X$,
- a set NoGoods(I) consisting of incompatible tuples.

A *partial solution* to I on $Y = \{y_1, \ldots, y_r\} \subseteq X$ is a tuple $t = \{\langle y_1, a_1\rangle, \ldots, \langle y_r, a_r\rangle\}$ such that no subset of t belongs to NoGoods(I). A *solution* is a partial solution on X.

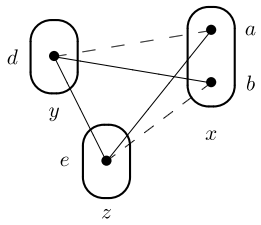


Fig. 1. A broken triangle on two values a, b for a given variable x.

3. Value merging in binary CSP based on the BTP

In this section we consider a method, based on the BTP, for reducing domain size while preserving satisfiability. Instead of eliminating a value, as in classic reduction operations such as arc consistency or neighbourhood substitution, we merge two values. We show that the absence of broken-triangles [7] on two values for a variable x in a binary CSP instance allows us to merge these two values in the domain of x while preserving satisfiability. This rule generalises the notion of virtual interchangeability [20] as well as neighbourhood substitution [21].

It is known that if for a given variable x in an arc-consistent binary CSP instance I, the set of (in)compatibilities (known as a broken-triangle) shown in Fig. 1 occurs for no two values $a, b \in \mathcal{D}(x)$ and no two assignments to two other variables, then the variable x can be eliminated from I without changing the satisfiability of I [7,11]. In figures, each bullet represents a variable-value assignment, assignments to the same variable are grouped together within the same oval and compatible pairs of assignments are linked by solid lines. In Fig. 1 (and in other figures illustrating forbidden patterns) incompatible pairs of assignments are linked by broken lines. Even when this variable-elimination rule cannot be applied, it may be the case that for a given pair of values $a, b \in \mathcal{D}(x)$, no broken triangle occurs. We will show that if this is the case, then we can perform a domain-reduction operation which consists in merging the values a and b.

Definition 3. *Merging* values $a, b \in \mathcal{D}(x)$ in a binary CSP consists in replacing a, b in $\mathcal{D}(x)$ by a new value c which is compatible with all variable-value assignments compatible with at least one of the assignments $\langle x, a\rangle$ or $\langle x, b\rangle$. A *value-merging condition* is a polytime-computable property $P(x, a, b)$ of assignments $\langle x, a\rangle$, $\langle x, b\rangle$ in a binary CSP instance I such that when $P(x, a, b)$ holds, the instance I' obtained from I by merging $a, b \in \mathcal{D}(x)$ is satisfiable if and only if I is satisfiable.

We now formally define the value-merging condition based on the BTP.

Definition 4. A *broken triangle* on the pair of variable-value assignments $a, b \in \mathcal{D}(x)$ consists of a pair of assignments $d \in \mathcal{D}(y)$, $e \in \mathcal{D}(z)$ to distinct variables $y, z \in X \setminus \{x\}$ such that $(a, d) \notin R_{xy}$, $(b, d) \in R_{xy}$, $(a, e) \in R_{xz}$, $(b, e) \notin R_{xz}$ and $(d, e) \in R_{yz}$. The pair of values $a, b \in \mathcal{D}(x)$ is *BT-free* if there is no broken triangle on a, b.

Proposition 5. *In a binary CSP instance, being BT-free is a value-merging condition. Furthermore, given a solution to the instance resulting from the merging of two values, we can find a solution to the original instance in linear time.*

Proof. Let I be the original instance and I' the new instance in which a, b have been merged into a new value c. Clearly, if I is satisfiable then so is I'. It suffices to show that if I' has a solution s which assigns c to x, then I has a solution. Let s_a, s_b be identical to s except that s_a assigns a to x and s_b assigns b to x. Suppose that neither s_a nor s_b are solutions to I. Then, there are variables $y, z \in X \setminus \{x\}$ such that $\langle a, s(y)\rangle \notin R_{xy}$ and $\langle b, s(z)\rangle \notin R_{xz}$. By definition of the merging of a, b to produce c, and since s is a solution to I' containing $\langle x, c\rangle$, we must have $(b, s(y)) \in R_{xy}$ and $(a, s(z)) \in R_{xz}$. Finally, $(s(y), s(z)) \in R_{yz}$ since s is a solution to I'. Hence, $\langle y, s(y)\rangle$, $\langle z, s(z)\rangle$, $\langle x, a\rangle$, $\langle x, b\rangle$ forms a broken-triangle, which contradicts our assumption. Hence, the absence of broken triangles on assignments $\langle x, a\rangle$, $\langle x, b\rangle$ allows us to merge these assignments while preserving satisfiability.

Reconstructing a solution to I from a solution s to I' simply requires checking which of s_a or s_b is a solution to I. Checking if s_a or s_b is a solution only requires checking the (at most) $n - 1$ binary constraints that include x. Thus finding a solution to the original instance can be achieved in linear time. □

We can see that the BTP-merging rule, given by Proposition 5, generalises neighbourhood substitution [21]: if b is neighbourhood substitutable by a, then no broken triangle occurs on a, b and merging a and b produces a CSP instance which is identical (except for the renaming of the value a as c) to the instance obtained by simply eliminating b from $\mathcal{D}(x)$. BTP-merging also generalises the merging rule proposed by Likitvivatanavong and Yap [20]. The basic idea behind their rule is that if the two assignments $\langle x, a\rangle$, $\langle x, b\rangle$ have identical compatibilities with all assignments to all other variables except concerning at most one other variable, then we can merge a and b. This is clearly subsumed by BTP-merging.

The BTP-merging operation is not only satisfiability-preserving but, from Proposition 5, we know that we can also reconstruct a solution in polynomial time to the original instance I from a solution to an instance I^m to which we have applied

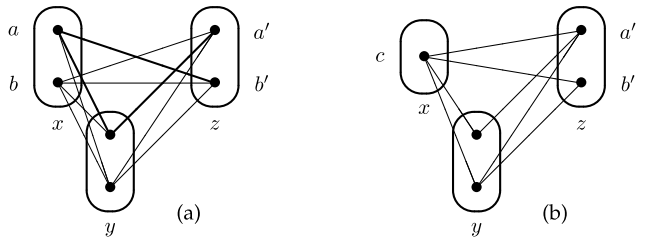


Fig. 2. (a) A broken triangle (shown in bold) exists on values a', b' at variable z. (b) After BTP-merging of values a and b in $\mathcal{D}(x)$, this broken triangle has disappeared.

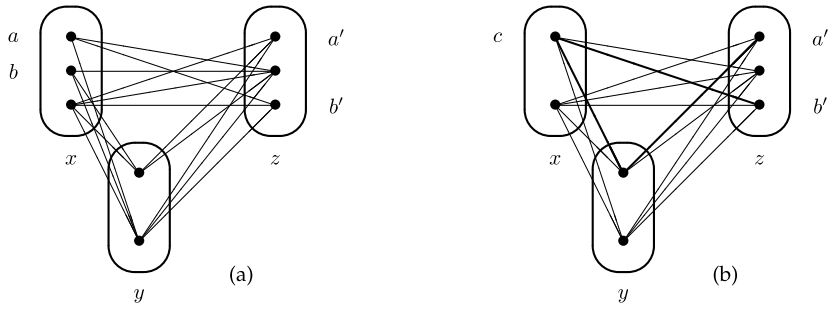


Fig. 3. (a) This instance contains no broken triangle. (b) After BTP-merging of values a and b in $\mathcal{D}(x)$, a broken triangle (shown in bold) has appeared on values $a', b' \in \mathcal{D}(z)$.

a sequence of merging operations until convergence. It is known that for the weaker operation of neighbourhood substitutability, all solutions to the original instance can be generated in $O(N(de+n^2))$ time, where N is the number of solutions to the original instance, n is the number of variables, d the maximum domain size and e the number of constraints [22]. We now show that a similar result also holds for the more general rule of BTP-merging.

Proposition 6. *Let I be a binary CSP instance and suppose that we are given*

- *a sequence of m triples of the form $(x_i, a_i, b_i)_{i=0}^{m-1}$, implicitly defining a sequence of instances $I^0 = I, I^1, \cdots, I^m$ such that I^{i+1} is obtained from I^i by BTP-merging values a_i, b_i for x_i $(i = 0, \ldots, m-1)$,*
- *the set of all N solutions to the instance I^m.*

All solutions to I can then be enumerated with delay $O(mn)$ after a preprocessing step in $O(mnd^2)$ (hence in total time $O(n^2d^3 + Nn^2d)$).

Proof. We start by computing, for each constraint R_{xy} in the original instance I, its successive versions $R_{xy}^{t_1}, \ldots, R_{xy}^{t_{m_{xy}}}$, where $t_1, \ldots, t_{m_{xy}} \in \{1, \ldots, m\}$ record by which BTP-merging operation this version was produced. Since each BTP-merging operation can change only $O(n)$ constraints (those involving x_i), this preprocessing step requires time $O(mnd^2)$.

Now given a solution s to I^i we proceed inductively as follows. If $i = 0$ then we output s, otherwise we test whether s_a or s_b (or both) are solutions to I^{i-1}, where s_a (resp. s_b) is obtained from s by setting x_i to a_i (resp. to b_i), as in the proof of Proposition 5. For each of them found to be a solution to I^{i-1}, we recurse with I^{i-1}. This requires $O(n)$ time per step, since again there are at most $n-1$ constraints to be checked (those involving x_i) and these have been precomputed. Finally, since at each step either s_a or s_b is guaranteed to be a solution to I^{i-1}, we indeed generate solutions to I with delay $O(mn)$. □

The weaker operation of neighbourhood substitution has the property that two different convergent sequences of eliminations by neighbourhood substitution necessarily produce isomorphic instances I_1^m, I_2^m [22]. This is not the case for BTP-merging. Firstly, and perhaps rather surprisingly, BTP-merging can have as a side-effect to eliminate broken triangles. This is illustrated in the 3-variable instance shown in Fig. 2. In order to avoid cluttering up figures with broken lines linking each pair of incompatible assignments, in all figures illustrating binary CSP instances, we use the convention that those pairs of assignments which are not explicitly linked with a solid line are incompatible. The instance in Fig. 2(a) contains a broken triangle on values a', b' for variable z, but after BTP-merging of values $a, b \in \mathcal{D}(x)$ into a new value c, as shown in Fig. 2(b), there are no broken triangles in the instance. Secondly, BTP-merging of two values in $\mathcal{D}(x)$ can introduce a broken triangle on a variable $z \neq x$, as illustrated in Fig. 3. The instance in Fig. 3(a) contains no broken triangle, but after the BTP-merging of $a, b \in \mathcal{D}(x)$ into a new value c, a broken triangle has been created on values $a', b' \in \mathcal{D}(z)$.

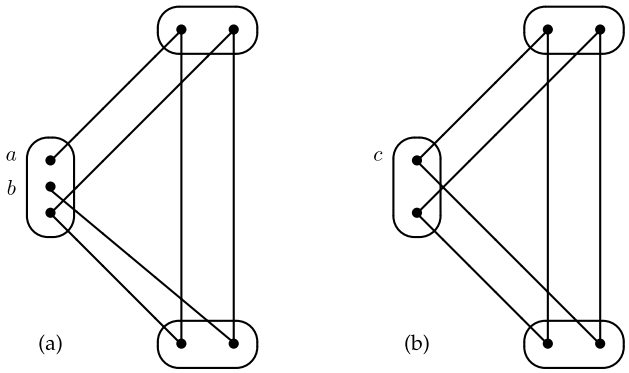


Fig. 4. (a) An instance in which applying AC leads to the elimination of all values (starting with the values a and b), but applying BTP merging leads to just one elimination, namely the merging of a with b (with the resulting instance shown in (b)).

4. Mixing arc consistency and BTP-merging

Given the omnipresence of arc consistency in constraint solvers, it is natural to investigate its relationship and interaction with BTP-merging. Values which can be BTP-merged may or may not be arc consistent. Trivially, two values $a, b \in \mathcal{D}(x)$ which are compatible with all assignments to all other variables can be BTP-merged, but cannot be eliminated by arc consistency. Conversely, if $a \in \mathcal{D}(x)$ has no AC-support at y but otherwise is compatible with all assignments to all other variables, $b \in \mathcal{D}(x)$ has no AC-support at $z \neq y$ but otherwise is compatible with all assignments to all other variables, and $R_{yz} \neq \emptyset$, then a, b can both be eliminated by arc consistency but a, b cannot be BTP-merged. Having established the incomparability of arc consistency and BTP-merging, we now investigate their possible interactions.

We have already observed that BTP-merging is a generalisation of neighbourhood substitutability, since if $a \in \mathcal{D}(x)$ is neighbourhood substitutable for $b \in \mathcal{D}(x)$ then a, b can be BTP-merged. The possible interactions between arc consistency (AC) and neighbourhood substitution (NS) are relatively simple and can be summarised as follows [22]:

1. The fact that $a \in \mathcal{D}(x)$ is AC-supported or not at variable y remains invariant after the elimination of any other value b (in $\mathcal{D}(x) \setminus \{a\}$ or in the domain $\mathcal{D}(z)$ of any variable $z \neq x$) by neighbourhood substitution.
2. An arc-consistent value $a \in \mathcal{D}(x)$ that is neighbourhood substitutable remains neighbourhood substitutable after the elimination of any other value by arc consistency.
3. On the other hand, a value $a \in \mathcal{D}(x)$ may become neighbourhood substitutable after the elimination of a value $c \in \mathcal{D}(y)$ ($y \neq x$) by arc consistency.

Indeed, it has been shown that the maximum cumulated number of eliminations by arc consistency and neighbourhood substitution can be achieved by first establishing arc consistency and then applying any convergent sequence of NS eliminations (i.e. any valid sequence of eliminations by neighbourhood substitution until no more NS eliminations are possible) [22].

The interaction between arc consistency and BTP-merging is not so simple and can be summarised as follows:

1. The fact that $a \in \mathcal{D}(x)$ is AC-supported or not at variable y remains invariant after the BTP-merging of any other pair of other values b, c (in $\mathcal{D}(x) \setminus \{a\}$ or in the domain $\mathcal{D}(z)$ of any variable $z \neq x$). However, after the BTP-merging of two arc-inconsistent values the resulting merged value may be arc consistent. An example is given in Fig. 4(a). In this 3-variable instance, the two values $a, b \in \mathcal{D}(x)$ can be eliminated by arc consistency (which in turn leads to the elimination of all values), or alternatively they can be BTP-merged (to produce the new value c) resulting in the instance shown in Fig. 4(b) in which no more eliminations are possible by AC or BTP-merging.
2. A single elimination by AC may prevent one or more BTP-mergings. An example is given in Fig. 5(a). In this 4-variable instance, if the value b is eliminated by AC, then no other eliminations are possible by AC or BTP-merging in the resulting instance (shown in Fig. 5(b)), whereas if a and b are BTP-merged into a new value d (as shown in Fig. 5(c)) this destroys a broken triangle thus allowing c to be BTP-merged with d (as shown in Fig. 5(d)).
3. On the other hand, two values in the domain of a variable x may become BTP-mergeable after an elimination of a value $d \in \mathcal{D}(z)$ ($z \neq x$) by arc consistency. An example is given in Fig. 6. In this 4-variable instance, initially a and b cannot be BTP-merged (Fig. 6(a)), but after value d is eliminated from $\mathcal{D}(z)$ by AC, the broken triangle has disappeared and a, b can be BTP merged (Fig. 6(b)).

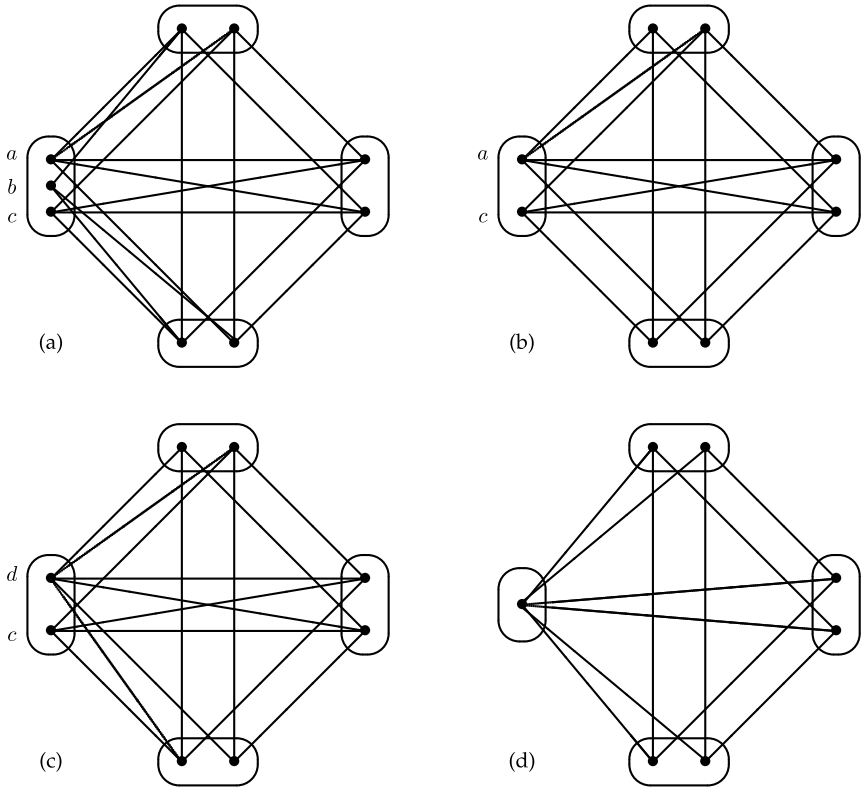


Fig. 5. (a) An instance in which applying AC leads to one elimination (the value b) (as shown in (b)), but applying BTP merging leads to two eliminations, namely a with b (shown in (c)) and then d with c (shown in (d)).

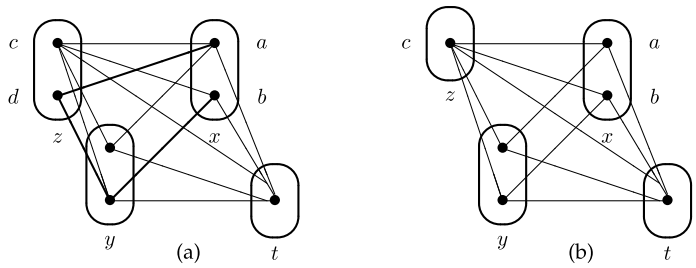


Fig. 6. (a) A broken triangle (shown in bold) exists on values a, b at variable x. (b) After removing value d from $\mathcal{D}(z)$ by AC, this broken triangle has disappeared.

5. The order of BTP-mergings

We saw in Section 3 that BTP-merging can both create and destroy broken triangles. This implies that the choice of the order in which BTP-mergings are applied may affect the total number of merges that can be performed. Unfortunately, maximising the total number of merges in a binary CSP instance turns out to be NP-hard, even when bounding the maximum size of the domains d by a constant as small as 3. For simplicity of presentation, we first prove this for the case in which the instance is not necessarily arc consistent. We will then prove a tighter version, namely NP-hardness of maximising the total number of merges even in arc-consistent instances.

Theorem 7. *The problem of determining if it is possible to perform k BTP-mergings in a boolean binary CSP instance is NP-complete.*

Proof. For a given sequence of k BTP-mergings, verifying if this sequence is correct can be performed in $\mathcal{O}(kn^2d^2)$ time because looking for broken triangles for a given couple of values takes $\mathcal{O}(n^2d^2)$. As we can verify a solution in polynomial time, the problem of determining if it is possible to perform k BTP-mergings in a binary CSP instance is in NP. So to complete the proof of NP-completeness it suffices to give a polynomial-time reduction from the well-known 3-SAT problem.

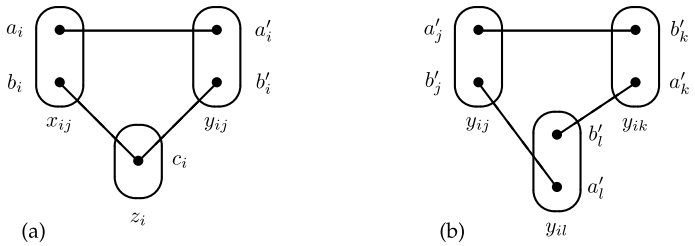


Fig. 7. (a) Representation of the variable X_i and its negation (by the possibility of performing a merge in $\mathcal{D}(x_{ij})$ or $\mathcal{D}(y_{ij})$, respectively, according to rules (1), (2)). (b) Representation of the clause $(X_j \vee X_k \vee X_l)$. Pairs of points joined by a solid line are compatible and incompatible otherwise.

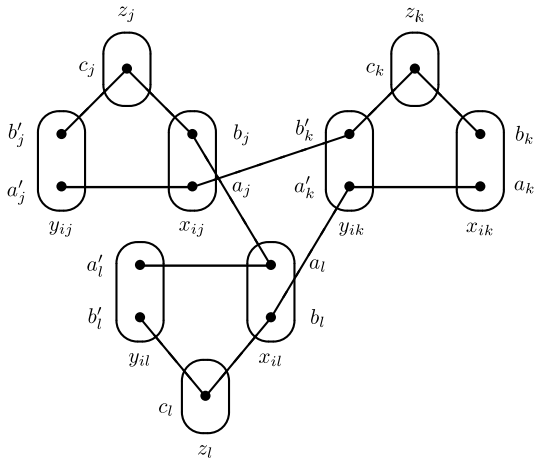


Fig. 8. Gadget representing the clause $(\overline{X_j} \vee X_k \vee \overline{X_l})$.

Let I_{3SAT} be an instance of 3-SAT (SAT in which each clause contains exactly 3 literals) with variables $X_1, \ldots, X_N$ and clauses $C_1, \ldots, C_M$. We will create a boolean binary CSP instance I_{CSP} which has a sequence of $k = 3 \times M$ mergings if and only if I_{3SAT} is satisfiable.

For each variable X_i of I_{3SAT}, we add a new variable z_i to I_{CSP}. For each occurrence of X_i in the clause C_j of I_{3SAT}, we add two more variables x_{ij} and y_{ij} to I_{CSP}. Each $\mathcal{D}(z_i)$ contains only one value c_i and each $\mathcal{D}(x_{ij})$ (resp. $\mathcal{D}(y_{ij})$) contains only two values a_i and b_i (resp. a'_i and b'_i). The roles of variables x_{ij} and y_{ij} are the following:

$$X_i = \mathit{true} \Leftrightarrow \forall j,\ a_i, b_i \text{ can be merged in } \mathcal{D}(x_{ij}) \tag{1}$$

$$X_i = \mathit{false} \Leftrightarrow \forall j,\ a'_i, b'_i \text{ can be merged in } \mathcal{D}(y_{ij}) \tag{2}$$

In order to prevent the possibility of merging both (a_i, b_i) and (a'_i, b'_i), we define the following constraints for z_i, x_{ij} and y_{ij}: $\forall j\ R_{x_{ij}z_i} = \{(b_i, c_i)\}$ and $R_{y_{ij}z_i} = \{(b'_i, c_i)\}$; $\forall j\ \forall k\ R_{x_{ij}y_{ik}} = \{(a_i, a'_i)\}$. These constraints are shown in Fig. 7(a) for a single j (where a pair of points not joined by a solid line are incompatible). By this gadget, we create a broken triangle on each y_{ij} when merging values in the x_{ij} and vice versa.

The idea is that BTP-merging a_i and b_i in any $\mathcal{D}(x_{ij})$ $(1 \leq j \leq N)$ prevents us from BTP-merging a'_i and b'_i in any $\mathcal{D}(y_{ik})$ $(1 \leq k \leq N)$, thus ensuring that the value of X_i is the same in each clause in which it occurs. If X_i is prevented from being assigned either false or true according to the rules (1) and (2) (because of the clause gadgets described below), then I_{3SAT} will be detected as unsatisfiable since the total number of merges will be less than $3 \times M$.

For each clause $C_i = (X_j, X_k, X_l)$, we add the following constraints in order to have at least one of the literals X_j, X_k, X_l true: $R_{y_{ij}y_{ik}} = \{(a'_j, b'_k)\}$, $R_{y_{ik}y_{il}} = \{(a'_k, b'_l)\}$ and $R_{y_{il}y_{ij}} = \{(a'_l, b'_j)\}$. This construction, shown in Fig. 7(b), is such that it allows two mergings on the variables y_{ij}, y_{ik}, y_{il} before a broken triangle is created. For example, merging a'_j, b'_j and then a'_k, b'_k creates a broken triangle on a'_i, b'_i. So a third merging is not possible.

If the clause C_i contains a negated literal $\overline{X_j}$ instead of X_j, it suffices to replace y_{ij} by x_{ij}. Indeed, Fig. 8 shows the construction for the clause $(\overline{X_j} \vee X_k \vee \overline{X_l})$ together with the gadgets for each variable.

The maximum number of mergings that can be performed are one per occurrence of each variable in a clause, which is exactly $3 \times M$. Given a sequence of $3 \times M$ mergings in the CSP instance, there is a corresponding solution to I_{3SAT} given by (1) and (2). To give a concrete example, consider the gadget shown in Fig. 8 representing the clause $\overline{X_j} \vee X_k \vee \overline{X_l}$. This gadget is made up of three triangles of the type shown in Fig. 7(a). To perform three merges in this gadget, we must perform exactly one merging in each of these triangles. For example, if we merge the pairs of values (a_j, b_j), (a_k, b_k) and (a_l, b_l),

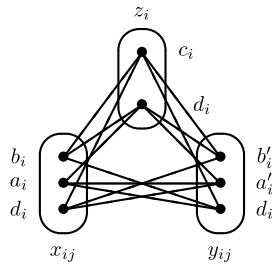


Fig. 9. Ensuring arc consistency between the variables z_i, y_{ij}, x_{ij} by addition of new values d_i.

then this sequence of merges corresponds to the assignment $(X_j, X_k, X_l) = (true, true, true)$ which satisfies the clause. On the other hand, the assignment $(X_j, X_k, X_l) = (true, false, true)$, which does not satisfy the clause, is impossible due to the central triangle of variables (of the type shown in Fig. 7(b)) which prevents us from simultaneously merging the three pairs of values (a_j, b_j), (a'_k, b'_k) and (a_l, b_l).

The above reduction allows us to code I_{3SAT} as the problem of testing the existence of a sequence of $k = 3 \times M$ mergings in the corresponding instance I_{CSP}. This reduction being polynomial, we have proved the NP-completeness of the problem of determining whether k BTP merges are possible in a boolean binary CSP instance. □

The reduction in the proof of Theorem 7 supposes that no arc-consistency operations are used. We will now show that it is possible to modify the reduction so as to prevent the elimination of any values in the instance I_{CSP} by arc-consistency, even when the maximum size of the domains d is bounded by a constant as small as 3. Recall that an arc-consistent instance remains arc-consistent after any number of BTP-mergings.

Theorem 8. *The problem of determining if it is possible to perform k BTP-mergings in an arc-consistent binary CSP instance is NP-complete, even when only considering binary CSP instances where the size of the domains is bounded by 3.*

Proof. In order to ensure arc-consistency of the instance I_{CSP}, we add a new value d_i to the domain of each of the variables x_{ij}, y_{ij}, z_i. However, we cannot simply make d_i compatible with all values in all other domains, because this would allow all values to be merged with d_i, destroying in the process the semantics of the reduction.

In the three binary constraints concerning the triple of variables x_{ij}, y_{ij}, z_i, we make d_i compatible with all values in the other two domains *except* d_i. In other words, we add the following tuples to constraint relations, as illustrated in Fig. 9:

- $\forall i \forall j$, (a_i, d_i), (b_i, d_i), $(d_i, c_i) \in R_{x_{ij}z_i}$
- $\forall i \forall j$, (a'_i, d_i), (b'_i, d_i), $(d_i, c_i) \in R_{y_{ij}z_i}$
- $\forall i \forall j$, (a_i, d_i), (b_i, d_i), (d_i, a'_i), $(d_i, b'_i) \in R_{x_{ij}y_{ij}}$

This ensures arc consistency, without creating new broken triangles on a_i, b_i or a'_i, b'_i, while at the same time preventing BTP-merging with the new value d_i. It is important to note that even after BTP-merging of one of the pairs a_i, b_i or a'_i, b'_i, no BTP-merging is possible with d_i in $\mathcal{D}(x_{ij})$, $\mathcal{D}(y_{ij})$ or $\mathcal{D}(z_i)$ due to the presence of broken triangles on this triple of variables. For example, the pair of values $a_i, d_i \in \mathcal{D}(x_{ij})$ belongs to a broken triangle on $c_i \in \mathcal{D}(z_i)$ and $d_i \in \mathcal{D}(y_{ij})$, and this broken triangle still exists if the values $a'_i, b'_i \in \mathcal{D}(y_{ij})$ are merged.

We can then simply make d_i compatible with all values in the domain of all variables outside this triple of variables. With these constraints we ensure arc consistency without changing any of the properties of I_{CSP} used in the reduction from 3-SAT described in the proof of Theorem 7. For each pair of values $a_i, b_i \in \mathcal{D}(x_{ij})$ and $a'_i, b'_i \in \mathcal{D}(y_{ij})$, no new broken triangle is created since these two values always have the same compatibility with all the new values d_k. As we have seen, the constraints shown in Fig. 9 prevent any merging of the new values d_k. □

Corollary 9. *The problem of determining if it is possible to perform k value eliminations by arc consistency and BTP-merging in a binary CSP instance is NP-complete, even when only considering binary CSP instances where the size of the domains is bounded by 3.*

A related question concerns the complexity of finding the optimal order of BTP-mergings within the domain of a single variable. It turns out that this too is NP-complete. The proof of this theorem [19] is based on a similar technique to that used in the proof of Theorem 7.

Theorem 10. *The problem of determining if it is possible to perform k BTP-mergings within a same domain in a binary CSP instance is NP-complete.*

6. Experimental trials

In this section, we study BTP-merging from a practical viewpoint.

6.1. Experimental protocol

To test the utility of BTP-merging we performed extensive experimental trials on CSP benchmark instances available from the International CP Competition.[1] Among the 7272 CSP benchmark instances, we consider all the instances including only binary constraints (namely 3795 instances). For each of these instances, we performed BTP-mergings until convergence with a time-out of one hour. In total, we obtained results for 2944 instances out of 3795 benchmark instances. In the other instances, the search for all BTP-mergings did not terminate within the time-out. Note that some of the considered instances have constraints defined by predicates. In such cases, these constraints are first expressed in extension before applying the BTP-merging algorithm. The runtime of this transformation is included in the reported runtime.

BTP-mergings are performed by checking first for virtual interchangeability and then by looking for BTP-mergeable pairs of values. These two steps are repeated until a fixed point is reached. By so doing, the virtual interchangeability step allows us to merge more quickly some pairs of values since the virtual interchangeability rule is easier to check than the BTP rule. Our experiments (not reported here) have shown that this version of BTP-merging is significantly faster than one presented in [18] while leading to a similar number of mergings.

For the BTP-merging step, we consider the variables according to a given ordering. Among the different variable orderings we tried, we opted in our experimental trials for one which orders the variables according to increasing degree (the degree of a variable being the number of constraints whose scope contains the variable). Note that this ordering differs from the one used in [18] which corresponds to a lexicographical ordering. In practice, we obtain a similar number of mergings with these two orderings but the algorithm is significantly faster with the first one. In general, we observed that the different variable orderings we tried had more impact on runtime than on the number of mergings performed.

For a given variable x, we check for each pair of values $a, b \in D(x)$ whether a, b are BTP-mergeable. If a broken triangle on a, b is found, we save it in a data structure. Then if, later, we have to check again the BTP-mergeability of a, b, we start with this saved broken triangle. If this triangle is still broken, we can immediately deduce that a, b are not BTP-mergeable, thus avoiding some useless checks. On the other hand, if this triangle is no longer broken, we check whether a, b are BTP-mergeable. If no broken triangle occurs on a, b (that is a, b are BTP-mergeable), we immediately merge a, b. This greedy algorithm is a natural choice since by Theorem 8 it is NP-hard to optimise the order of BTP-merges. For efficiency reasons, when merging a, b, we keep one value (assume without loss of generality that a is this value) and delete the other one instead of creating a new value c and removing a and b as evoked in Definition 3. Then a is made compatible with each variable-value assignment compatible with the assignment $\langle x, b\rangle$.

We also implemented the deletion of values by neighbourhood substitution, by virtual interchangeability or by arc-consistency (which is enforced by the AC-2001 algorithm [23]). In the remainder of this section, we denote AC+P the application of AC followed by merging according to the property P where P may be BTP-merging, neighbourhood substitution (NS) or virtual interchangeability (VI). For solving, we use MAC (for Maintaining Arc-Consistency [24]) based on AC-2001 together with the variable ordering heuristic dom/wdeg [25]. The choice of MAC is a natural choice since most state-of-the-art solvers rely on it. All the algorithms are implemented in C++.

The experimentations were performed on 8 Dell PowerEdge M820 blade servers with two processors (Intel Xeon E5-2609 v2 2.5 GHz and 32 GB of memory) under Linux Ubuntu 14.04.

6.2. Comparisons between BTP-merging and AC+BTP-merging

We compare in this subsection the results obtained by BTP-merging and by AC+BTP-merging. First, as shown in Fig. 10, AC+BTP-merging is able to process (i.e. find a fixed point in which no more BTP-mergings are possible) more instances within the time-out than BTP-merging alone. More precisely, AC+BTP-merging succeeds in terminating within the time-out for 2944 instances against 2856 for BTP-merging. In both cases, for more than one third of these instances, some mergings occur. Fig. 11 compares the percentages of values removed by BTP-merging and AC+BTP-merging for the instances for which both BTP-merging and AC+BTP-merging terminate. Clearly, AC+BTP-merging outperforms BTP-merging since the percentage of values removed by AC+BTP-merging is always greater than or equal to the number of values removed by BTP-merging. We can see that for certain types of problem, (AC+)BTP-merging is very effective (more than 90% of deleted values), whereas for others hardly any merging of values occurred. In particular, we have observed that often the instances for which no merging is possible have some disequality constraints (which makes sense, since even a conjunction of disequality constraints as simple as $(x \neq y) \wedge (x \neq z) \wedge (y \neq z)$ with $a, b \in D(x), a \in D(y), b \in D(z)$, induces a broken triangle on a, b). For instance, for the graph colouring instances, (AC+)BTP-mergings only occur when the instances have variables with degree 0 or 1. In contrast, (AC+)BTP-merging is very effective for some real-world instances from frequency assignment problems (`fapp*`, `graph*` or `scen*`) or for some patterned instances (like `BlackHole*` or `os-taillard*`). Note that at best, BTP-merging reduces

[1] http://www.cril.univ-artois.fr/CPAI08.

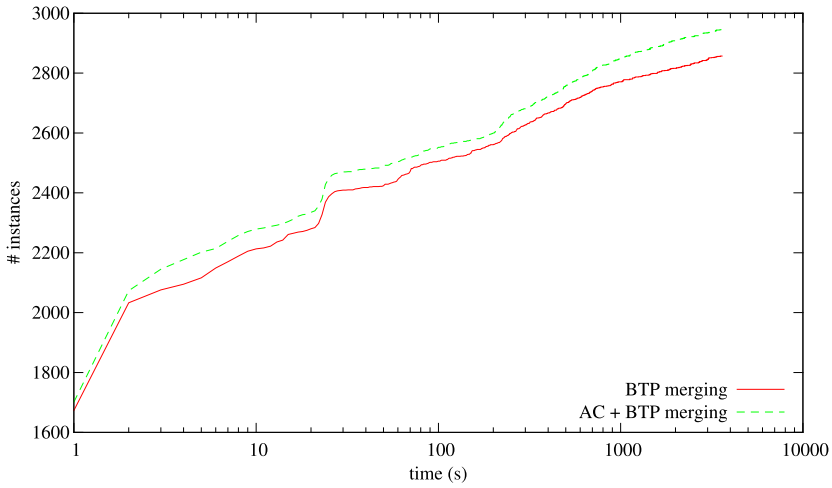


Fig. 10. Number of instances processed by BTP-merging with and without AC preprocessing depending on the elapsed time (in seconds).

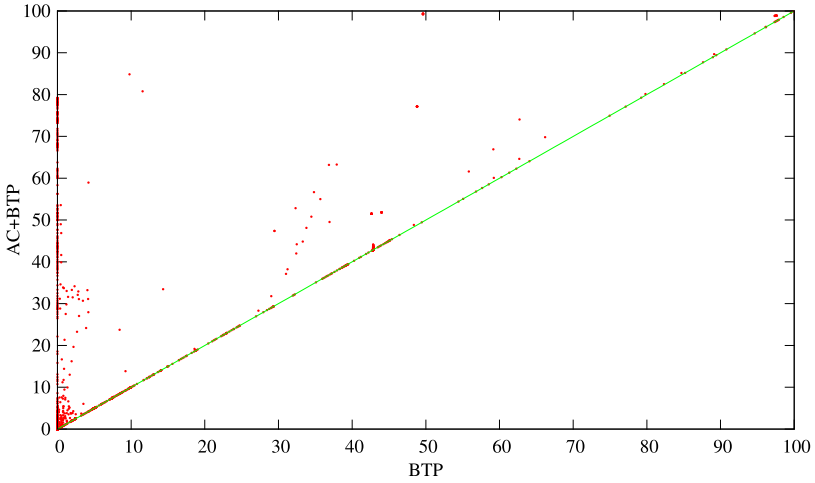


Fig. 11. Percentage of values removed by BTP vs percentage of values removed by AC and BTP-merging (AC+BTP) for each considered instance.

all variable domains to singletons (and so cannot remove all the values in a domain). For example, this is the case for all instances `hanoi*` which satisfy the broken-triangle property [26]. Tables 1 and 2 provide some detailed results for some selected instances. These instances have been selected in such a way that all observed trends are represented.

Regarding runtime, BTP-merging and AC+BTP-merging are often close as shown in Fig. 12. However, for a few instances, such as `langford-4-14`, AC+BTP-merging requires more time than BTP-merging. Such a result is often explained by the fact that the values used to quickly find broken triangles in BTP-merging have been removed by AC in AC+BTP-merging. In contrast, in most cases, achieving an AC preprocessing is useful since it saves time. Moreover, sometimes, it turns out to be very useful since it makes it possible to process more instances. For example, for the instance `fapp25-2230-8`, the values removed by AC make it possible for the BTP-merging step to terminate. Finally, we can note that a large part of the considered instances are processed quickly. Indeed, for about 57% of instances, achieving (AC+)BTP-merging requires less than one second.

6.3. Comparisons with neighbourhood substitution and virtual interchangeability

As shown in Section 3, the BTP-merging rule generalises the notion of neighbourhood substitution as well as virtual interchangeability. Hence, when we compare the percentage of values removed by BTP-merging with the number of values removed by neighbourhood substitution or virtual interchangeability, BTP-merging is always better than or equivalent to neighbourhood substitution or virtual interchangeability. The same trend is observed when the instances are preprocessed by AC. Figs. 13 and 14 compare the percentage of values removed by BTP-merging and by neighbourhood substitution or virtual interchangeability after having enforced AC. Nevertheless, even when the percentages are equal, we have no guarantee that BTP-merging removes the same values as neighbourhood substitution or virtual interchangeability. So, in order to

Table 1
For each selected instance, the number n of variables, the number e of constraints, the total number of values, the number of values removed by neighbourhood substitution (NS) or by virtual interchangeability (VI), the number of values removed by BTP-merging, the number of values for which neighbourhood substitution or virtual interchangeability hold among the values removed by BTP-merging and the runtime in seconds of BTP-merging. A dash means that the information is unknown because the runtime of BTP-merging exceeds the time-out of one hour.

Instance	n	e	# values	NS	VI	BTP			
				# del.	# del.	# del.	# NS	# VI	Time
bqwh-15-106-18_ext	106	597	385	0	0	0	0	0	<0.01
le-450-5a-2-ext	450	5714	900	0	0	0	0	0	0.04
geo50-20-d4-75-100_ext	50	393	1000	0	0	0	0	0	0.05
rand-24-24-276-139-53021_ext	24	276	576	0	0	0	0	0	0.03
langford-4-14	56	1540	3136	0	0	0	0	0	8.18
haystacks-21	441	4430	9261	0	20	20	0	20	6.44
rand-2-40-180-84-900-56_ext	40	84	7200	0	358	358	0	358	23.47
mulsol-i-4-31	185	3946	5735	300	300	300	300	300	8.01
e0ddr2-10-by-5-7	50	265	6215	366	0	366	218	0	224.11
inithx-i-2-28	645	13,979	18,060	2349	2349	2349	2349	2349	220.76
scen1-f9	916	5548	28,596	368	532	1200	251	588	669.72
fapp01-0200-8	200	1108	26,963	0	113	113	0	113	2528.42
ehi-85-297-33_ext	297	4094	2079	0	0	891	0	0	1.18
os-taillard-4-105-5	16	48	2500	812	0	812	406	0	102.73
scen10-w1-f3	680	1138	25,192	893	6626	7419	2411	6770	345.57
BlackHole-4-7-e-0_ext	112	1261	2102	697	887	896	463	887	5.71
BlackHole-4-13-e-1_ext	208	4217	7334	2541	3209	3226	1691	3209	151.31
os-taillard-4-95-7	16	48	2508	1558	0	1573	777	49	123.43
scen4	680	3967	26,856	0	268	3103	214	455	445.89
graph13-w0	916	458	35,176	0	34,260	34,260	17,130	34,260	0.36
large-92-unsat_ext	92	4186	8464	8280	8275	8280	4233	8279	2.83
lard-92-92	92	4186	8556	7163	5303	8347	840	5314	448.00
fapp25-2230-8	2230	11,974	610,084	44,168	–	–	–	–	–
hanoi-5_ext	30	29	6808	0	27	6778	41	6752	0.45

Table 2
For each selected instance, the total number of values, the number of values removed by AC, by AC and neighbourhood substitution (NS) or by virtual interchangeability (VI) after AC preprocessing, the number of values removed by BTP-merging after AC preprocessing, the number of values for which neighbourhood substitution or virtual interchangeability hold among the values removed by BTP-merging and the runtime in seconds of BTP-merging.

Instance	# values	AC	NS	VI	BTP			
		# del.	# del.	# del.	# del.	# NS	# VI	Time
bqwh-15-106-18_ext	385	0	0	0	0	0	0	<0.01
le-450-5a-2-ext	900	0	0	0	0	0	0	0.04
geo50-20-d4-75-100_ext	1000	0	0	0	0	0	0	0.05
rand-24-24-276-139-53021_ext	576	0	0	0	0	0	0	0.03
langford-4-14	3136	1428	0	0	0	0	0	35.18
haystacks-21	9261	0	0	20	20	0	20	6.45
rand-2-40-180-84-900-56_ext	7200	0	0	358	358	0	358	23.50
mulsol-i-4-31	5735	0	300	300	300	300	300	7.93
e0ddr2-10-by-5-7	6215	0	366	0	366	218	0	225.09
inithx-i-2-28	18,060	0	2349	2349	2349	2349	2349	225.93
scen1-f9	28,596	7604	0	383	390	168	383	329.40
fapp01-0200-8	26,963	9155	21	159	174	6	159	1003.46
ehi-85-297-33_ext	2079	2	0	0	889	0	0	1.19
os-taillard-4-105-5	2500	287	810	0	818	404	22	102.44
scen10-w1-f3	25,192	5134	0	6321	6808	2138	6451	190.51
BlackHole-4-7-e-0_ext	2102	280	634	802	802	427	802	2.93
BlackHole-4-13-e-1_ext	7334	793	2422	3007	3007	1623	3007	79.57
os-taillard-4-95-7	2508	451	1346	4	1406	656	122	106.43
scen4	26,856	19,534	0	774	2161	406	924	44.44
graph13-w0	35,176	0	0	34,260	34,260	17,130	34,260	0.36
large-92-unsat_ext	8464	0	8280	8275	8280	4233	8279	2.82
lard-92-92	8556	4350	4114	3878	4114	2494	3886	3.93
fapp25-2230-8	610,084	590,850	5294	14,469	16,449	4209	15,840	226.09
hanoi-5_ext	6808	6778	0	0	0	0	0	<0.01

make a finer comparison, we check, for each BTP-mergeable pair of values, whether neighbourhood substitution or virtual interchangeability may also hold. Table 1 gives these results for a selection of the considered instances. For a few instances, all the values removed by BTP-merging can also be deleted by neighbourhood substitution or virtual interchangeability. In most cases (e.g. for the instances `inithx-i-2-28` or `mulsol-i-4-31`), the removed values belong to domains of variables having a degree 0 or 1. At the opposite extreme, for some instances, such as `ehi-85-297-33_ext`, none of

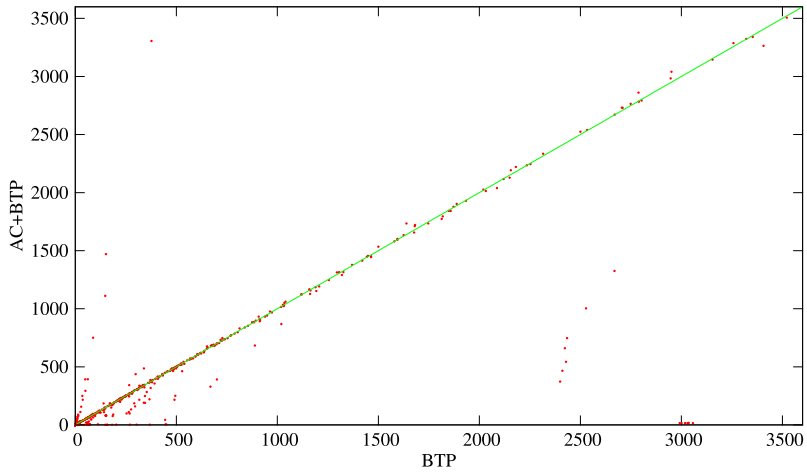


Fig. 12. Runtime of BTP-merging vs runtime of AC+BTP-merging for each considered instance.

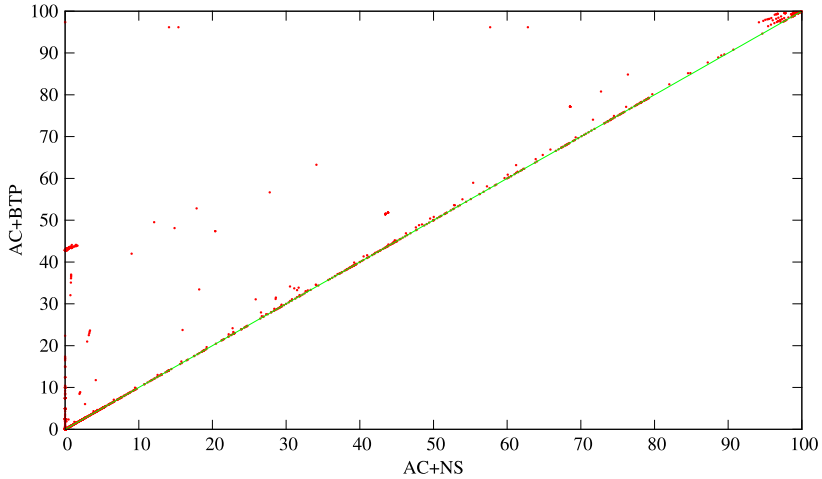


Fig. 13. Percentage of values removed by AC and neighbourhood substitution (AC+NS) vs percentage of values removed by AC and BTP-merging (AC+BTP) for each considered instance.

the values removed by BTP-merging can be removed by neighbourhood substitution or virtual interchangeability. For the majority of instances, BTP-merging removes some values which are removed neither by neighbourhood substitution nor by virtual interchangeability. We observe the same trends when the instances are preprocessed by AC (Table 2).

6.4. Impact on solving

In this subsection, we investigate the impact of removed values on the solving performed by MAC. For these experiments, we only consider those 828 instances which are arc-consistent and for which AC+BTP-merging removes at least one value. First, we observe that MAC with AC+BTP-merging solves 697 instances against 688 for MAC alone within the time-out of one hour. Note that the runtime of MAC with AC+BTP-merging includes the runtime of the solving and the AC+BTP-merging. Fig. 15 provides a comparison of the runtimes of MAC and MAC with AC+BTP-merging for the selected instances. Clearly, for most instances, MAC outperforms MAC with AC+BTP-merging with respect to runtime. This result is clearly due to the cost of achieving AC+BTP-merging which sometimes turns out to be too expensive with respect to the runtime of solving. However, in some cases, MAC with AC+BTP-merging is faster than MAC alone and, overall, is able to solve more instances.

In order to better assess the impact on solving, we now consider the number of nodes developed by MAC and MAC with AC+BTP-merging. We can see in Fig. 16 that solving by MAC with AC+BTP-merging turns out to be more efficient than we would have thought by just studying the total runtime. Indeed, thanks to the values removed by AC+BTP-merging, MAC with AC+BTP-merging is often able to develop less nodes than MAC alone. The total number of nodes developed by MAC with AC+BTP-merging is 27% less than the total number of nodes developed by MAC (32 millions compared to 44 millions). These preliminary results concerning solving are promising. However, in order to make MAC with AC+BTP-merging competitive, we have now to look for better algorithms for achieving AC+BTP-merging or techniques for identifying which instances could

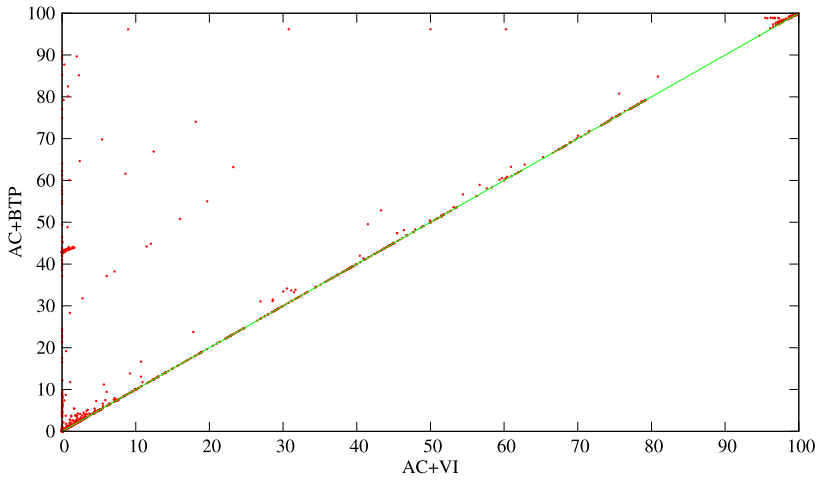


Fig. 14. Percentage of values removed by AC and virtual interchangeability (AC+VI) vs percentage of values removed by AC and BTP-merging (AC+BTP) for each considered instance.

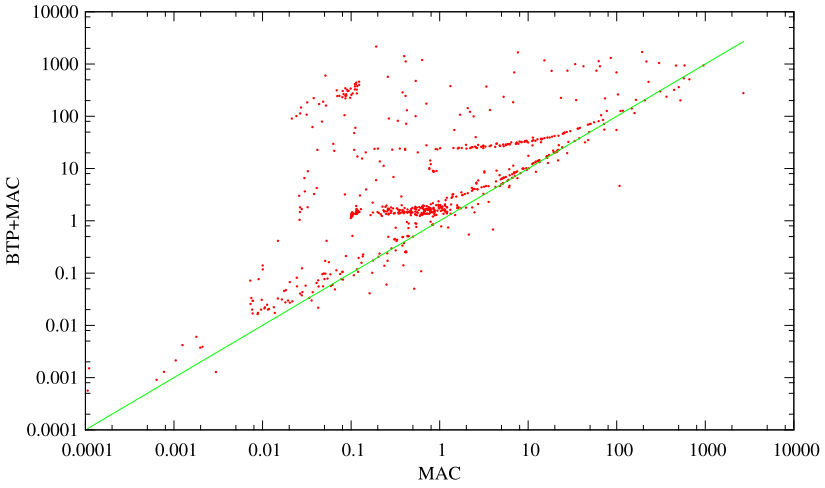


Fig. 15. Runtime of MAC vs runtime of MAC after BTP-merging. Runtimes are given in seconds.

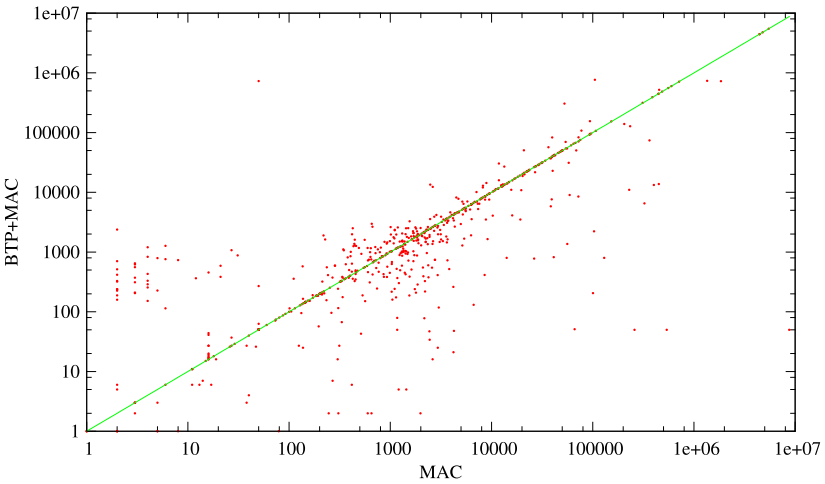


Fig. 16. Number of nodes developed by MAC vs number of nodes developed by MAC after BTP-merging.

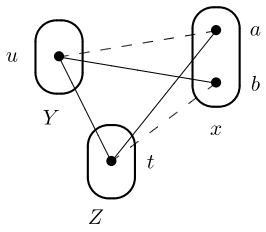


Fig. 17. A general-arity broken triangle on values $a, b \in \mathcal{D}(x)$.

best profit from BTP-merging during preprocessing. Note that the cost of searching for broken triangles precludes using BTP-merging during search.

An interesting phenomenon which is worthy of further investigation is that the number of nodes in the search tree may actually increase due to merging (since it tends to make constraints less tight) even though domain size has decreased. Likitvivatanavong and Yap [20] mention that search runtime may increase after merging virtual interchangeable values and indeed they observed that the number of instances for which search runtime increased was approximately the same as the number of instances in which search runtime decreased. An open theoretical question concerning the performance of MAC with or without BTP-merging is the existence of conditions under which BTP-merging is guaranteed not to increase the number of nodes in the search tree. Similarly, further experimental trials would be necessary to uncover relationships between the expected gain by BTP-merging and parameters such as average domain size, constraint density and constraint tightness.

7. Generalising BTP-merging to constraints of arbitrary arity

In the remainder of the paper, we assume that the constraints of a general-arity CSP instance I are given in the form described in Definition 2, i.e. as a set of incompatible tuples NoGoods(I), where a tuple is a set of variable-value assignments. For simplicity of presentation, we use the predicate Good(I, t) which is true iff the tuple t is a partial solution, i.e. t does not contain any pair of distinct assignments to the same variable and $\nexists t' \subseteq t$ such that $t' \in$ NoGoods(I). We first generalise the notion of broken triangle and merging to the general-arity case, before showing that absence of broken triangles allows merging.

Definition 11. A *general-arity broken triangle* (GABT) on values $a, b \in \mathcal{D}(x)$ consists of a pair of tuples t, u (containing no assignments to variable x) satisfying the following conditions:

1. Good($I, t \cup u$) $\wedge$ Good($I, t \cup \{\langle x, a\rangle\}$) $\wedge$ Good($I, u \cup \{\langle x, b\rangle\}$)
2. $t \cup \{\langle x, b\rangle\} \in$ NoGoods(I) $\wedge$ $u \cup \{\langle x, a\rangle\} \in$ NoGoods(I)

The pair of values $a, b \in \mathcal{D}(x)$ is *GABT-free* if there is no broken triangle on a, b.

A general-arity broken triangle is illustrated in Fig. 17. This figure is identical to Fig. 1 except that Y, Z are now sets of variables and t, u are tuples. Note that the sets Y and Z may overlap. As in the binary case, a dashed line represents a nogood (i.e. a tuple not in the constraint relation on its variables). A solid line now represents a partial solution.

If the constraints are represented by nogoods, as in our Definition 2, then to decide whether there is a GABT on a, b in a CSP instance, one can use the second condition in Definition 11 and explore all pairs $t \cup \{\langle x, b\rangle\}, u \cup \{\langle x, a\rangle\} \in$ NoGoods(I). On the other hand, if the constraints are represented as lists of allowed tuples, then one can use the first condition in Definition 11 and explore all pairs $t \cup \{\langle x, a\rangle\}, u \cup \{\langle x, b\rangle\}$ of tuples explicitly allowed by the constraints in I (since the second condition implies that under this representation, there is a constraint over the variables of t and x, and one over the variables of u and x). Whatever the representation, a pair t, u can be checked to be a GABT on a, b by evaluating the properties of Definition 11, all of which involve only constraint checks. Hence deciding whether a pair a, b is GABT-free is polytime for constraints given in extension (as the set of satisfying assignments) as well as for those given by nogoods (the set of assignments violating the constraint).

Definition 12. *Merging* values $a, b \in \mathcal{D}(x)$ in a general-arity CSP instance I consists of replacing a, b in $\mathcal{D}(x)$ by a new value c which is compatible with all variable-value assignments compatible with at least one of the assignments $\langle x, a\rangle$ or $\langle x, b\rangle$, thus producing an instance I' with the new set of nogoods defined as follows:

$$\begin{aligned}\text{NoGoods}(I') = &\{t \in \text{NoGoods}(I) \mid \langle x, a\rangle, \langle x, b\rangle \notin t\}\\ &\cup \{t \cup \{\langle x, c\rangle\} \mid t \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I) \ \wedge\\ &\quad \exists t' \in \text{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle x, b\rangle\}\}\end{aligned}$$

$$\cup \{t \cup \{\langle x, c\rangle\} \mid t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I) \ \wedge$$
$$\exists t' \in \text{NoGoods}(I) \text{ s.t. } t' \subseteq t \cup \{\langle x, a\rangle\}\}$$

A *value-merging condition* is a polytime-computable property $P(x, a, b)$ of assignments $\langle x, a\rangle$, $\langle x, b\rangle$ in a CSP instance I such that when $P(x, a, b)$ holds, the instance I' is satisfiable if and only if I is satisfiable.

This merging operation can be performed in polynomial time whether constraints are represented positively in extension or negatively as nogoods. For representations using nogoods this is clear from Definition 12. For representations in extension, simply observe that as in the binary case, the operation amounts to gathering together tuples which satisfy Good$(I, \cdot)$ and containing $\langle x, a\rangle$ or $\langle x, b\rangle$, and setting x to c in them.

Proposition 13. *In a general-arity CSP instance, being GABT-free is a value-merging condition. Furthermore, given a solution to the instance resulting from the merging of two values, we can find a solution to the original instance in linear time.*

Proof. In order to prove that satisfiability is preserved by this merging operation, it suffices to show that if s is a solution to I' containing $\langle x, c\rangle$, then either $s_a = (s \setminus \{\langle x, c\rangle\}) \cup \{\langle x, a\rangle\}$ or $s_b = (s \setminus \{\langle x, c\rangle\}) \cup \{\langle x, b\rangle\}$ is a solution to I. Suppose, for a contradiction that this is not the case. Then there are tuples $t, u \subseteq s \setminus \{\langle x, c\rangle\}$ such that $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I)$ and $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I)$. Since t, u are subsets of the solution s to I' and t, u contain no assignments to x, we have Good$(I, t \cup u)$. Since $t \cup \{\langle x, c\rangle\}$ is a subset of the solution s to I', we have $t \cup \{\langle x, c\rangle\} \notin \text{NoGoods}(I')$. By the definition of NoGoods(I') given in Definition 12, and since $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I)$, we know that $\nexists t' \in \text{NoGoods}(I)$ such that $t' \subseteq t \cup \{\langle x, a\rangle\}$. But then Good$(I, t \cup \{\langle x, a\rangle\})$. By a symmetric argument, we can deduce Good$(I, u \cup \{\langle x, b\rangle\})$. This provides the contradiction we were looking for, since we have shown that a general-arity broken triangle occurs on t, u, $\langle x, a\rangle$, $\langle x, b\rangle$.

Reconstructing a solution to the original instance can be achieved in linear time, since it suffices to verify which (or both) of s_a or s_b is a solution to I. □

7.1. Relationship with resolution in SAT

We now show that in the case of Boolean domains, there is a close relationship between merging two values a, b on which no GABT occurs and a common preprocessing operation used by SAT solvers. Given a propositional CNF formula φ in the form of a set of clauses (each clause C_i being represented as a set of literals) and a variable x occurring in φ, recall that *resolution* is the process of inferring the clause $(C_0 \cup C_1)$ from the two clauses $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1)$. Define the formula $Res(x, \varphi)$ to be the result of performing all such resolutions on φ, removing all clauses containing x or $\bar{x}$, and removing subsumed clauses:

$$Res(x, \varphi) = \min_{\subset}(\{C \mid C \in \varphi; x, \bar{x} \notin C\} \cup \{(C_0 \cup C_1) \mid (\{\bar{x}\} \cup C_0), (\{x\} \cup C_1) \in \varphi\})$$

It is a well-known fact that $Res(x, \varphi)$ is satisfiable if and only if φ is.

Eliminating variables in this manner from SAT instances, to get an equisatisfiable formula with less variables, is a common preprocessing step in SAT solving, and is typically performed provided it does not increase the size of the formula [27]. A particular case is when it amounts to simply removing all occurrences of x, which is the case, for instance, if x or $\bar{x}$ is unit or pure in φ, or if all resolutions on x yield a tautological clause.

Definition 14. A variable x is said to be *erasable* from a CNF φ if

$$Res(x, \varphi) \subseteq \{C \mid C \in \varphi; x, \bar{x} \notin C\} \ \cup \ \{C_0 \mid (\{\bar{x}\} \cup C_0) \in \varphi\} \ \cup \ \{C_1 \mid (\{x\} \cup C_1) \in \varphi\}$$

A CNF φ can be seen as the CSP instance I_φ on the set X of variables occurring in φ, with $\mathcal{D}(x) = \{\top, \bot\}$ for all $x \in X$, and NoGoods$(I_\varphi) = \{\overline{C} \mid C \in \varphi\}$, where $\overline{(\{x_1, \cdots x_p, \bar{x}_{p+1}, \cdots, \bar{x}_q\})} = \{\langle x_1, \bot\rangle, \ldots, \langle x_p, \bot\rangle, \langle x_{p+1}, \top\rangle, \ldots, \langle x_q, \top\rangle\}$.

Proposition 15. *Assume that no GABT occurs on values $\bot, \top$ for x in I_φ. Assume moreover that no clause in φ is subsumed by another one.*[2] *Then x is erasable from φ.*

Proof. Rephrasing Definition 11 (1) in terms of clauses, for any two clauses $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1) \in \varphi$ we have one of (i) $\exists C \in \varphi, C \subseteq (C_0 \cup C_1)$, (ii) $\exists C' \in \varphi, C' \subseteq (C_0 \cup \{x\})$, or (iii) $\exists C'' \in \varphi, C'' \subseteq (C_1 \cup \{\bar{x}\})$. Moreover, in Case (ii) C' must contain x, for otherwise the clause $(\{\bar{x}\} \cup C_0)$ would be subsumed in φ, contradicting our assumption. Similarly, in Case (iii) C'' must contain $\bar{x}$.

In Case (i) the resolvent $(C_0 \cup C_1)$ of $(\{\bar{x}\} \cup C_0), (\{x\} \cup C_1)$ is subsumed by C in $Res(x, \varphi)$, and hence does not occur in it. Similarly, in the second case $(C_0 \cup C_1)$ is subsumed by the resolvent of $(\{\bar{x}\} \cup C_0)$ and C', which is precisely C_0. The third

[2] This is without loss of generality since such clauses can be removed in polytime and such removal preserves logical equivalence.

case is dual. We finally have that the only resolvents added are of the form C_0 (resp. C_1) for some clause $(\{\bar{x}\} \cup C_0)$ (resp. $(\{x\} \cup C_1)$) of φ, as required. □

We can show the converse is also true provided that a very reasonable property holds.

Proposition 16. *Assume that φ satisfies: $\forall(\{x\} \cup C) \in \varphi, \nexists C' \subseteq C, (\{\bar{x}\} \cup C') \in \varphi$ and dually $\forall(\{\bar{x}\} \cup C) \in \varphi, \nexists C' \subseteq C, (\{x\} \cup C') \in \varphi$. If x is erasable from φ, then no GABT occurs on values $\bot, \top$ for x in I_φ.*

Proof. Assume for a contradiction that there is a GABT on values $\bot, \top$ for x in I_φ, let t, u be witnesses to this, and write $t \cup \{\langle x, \top\rangle\} = \overline{(\{\bar{x}\} \cup C_0)}$, $u \cup \{\langle x, \bot\rangle\} = \overline{(\{x\} \cup C_1)}$. Then the clause $(C_0 \cup C_1)$ is produced by resolution on x. Since x is erasable, $(C_0 \cup C_1)$ is equal to or subsumed by a clause $C \in Res(x, \varphi)$, where (applying Definition 14 in reverse) either C, or $(\{x\} \cup C)$, or $(\{\bar{x}\} \cup C)$ is in φ. The first case contradicts $\text{Good}(I_\varphi, t \cup u)$, so assume by symmetry $(\{x\} \cup C) \in \varphi$. From $C \notin \varphi$ and $C \in Res(x, \varphi)$ we get $\exists C' \subseteq C, (\{\bar{x}\} \cup C') \in \varphi$. But then the pair of clauses $(\{x\} \cup C), (\{\bar{x}\} \cup C') \in \varphi$ violates the assumption of the claim. □

8. BTP-merging in the presence of global constraints

Global constraints are an important feature of constraint programming. They not only facilitate modelling of complex problems but many global constraints also have dedicated efficient filtering algorithms [28]. In the presence of global constraints there are specific questions which need to be addressed to know whether BTP-merging is useful. The first thing to verify is that mergings are possible in the presence of one or more global constraints. A second important point is whether these BTP-mergings can be detected in polynomial time. A third point is to determine whether the semantics of the global constraint(s) are preserved by the operation of merging two values. For those global constraints that are decomposable into the conjunction of low-arity constraints, we can also ask whether BTP-merging applied to the decomposed version is equivalent to BTP-merging applied to the original global constraint(s). The answers to these questions depend on the global constraints. This section presents results concerning the important global constraint AllDifferent. These results are both negative and positive.

Proposition 17. *Determining whether two values can be GABTP-merged in a CSP instance consisting of two AllDifferent constraints is coNP-complete.*

Proof. It suffices to show that the problem of testing the existence of a general-arity broken triangle (GABT) in a CSP instance consisting of two AllDifferent constraints is NP-complete. We denote this problem by ∃GABT(2AllDiff). Clearly, the validity of a GABT can be checked in polynomial time. Testing the satisfiability of a CSP instance consisting of two AllDifferent constraints (a problem which we denote by CSP(2AllDiff)) is known to be NP-complete [29]. Thus to complete the proof it suffices to exhibit a polynomial reduction from CSP(2AllDiff) to ∃GABT(2AllDiff).

Let I be an instance, over variables X, consisting of two AllDifferent constraints with scopes S_1, S_2. Without loss of generality, we suppose that $S_1 \cup S_2 = X$. Let x, y, z be three variables not in X with domains containing only values not occurring in the domains of the variables in X, including $a, b \in \mathcal{D}(x)$ with $a \in \mathcal{D}(y)$, $a \notin \mathcal{D}(z)$, $b \in D(z)$, $b \notin D(z)$. We construct a new instance I' with variables $X \cup \{x, y, z\}$, with domains as in I for variables in X and the domains of variables x, y, z as just described. The instance I' has just two constraints: AllDifferent constraints with scopes $S_1 \cup \{y, x\}$ and $S_2 \cup \{z, x\}$. We will show that I' has a GABT on $a, b \in \mathcal{D}(x)$ if and only if I has a solution. A GABT on $a, b \in \mathcal{D}(x)$ consists of tuples t, u (containing no assignments to variable x) satisfying the following conditions: $\text{Good}(I', t \cup u)$, $\text{Good}(I', t \cup \{\langle x, a\rangle\})$, $\text{Good}(I', u \cup \{\langle x, b\rangle\})$, $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I')$ and $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I')$. Since $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I')$, but $\text{Good}(I', u)$, we must have $\langle y, a\rangle \in u$, since y is the only variable other than x containing a in its domain. Similarly, we can deduce that $\langle z, b\rangle \in t$. Now $\text{Good}(I', t \cup u)$ implies that $(t \setminus \{\langle z, b\rangle\}) \cup (u \setminus \{\langle y, a\rangle\})$ is a solution to I. On the other hand, suppose that s is a solution to I. Let $u = s[S_1] \cup \{\langle y, a\rangle\}$ and $t = s[S_2] \cup \{\langle z, b\rangle\}$ (where $s[S]$ represents the subset of s corresponding to assignments to variables in S). Then the tuples t and u satisfy the conditions: $\text{Good}(I', t \cup u)$, $\text{Good}(I', t \cup \{\langle x, a\rangle\})$, $\text{Good}(I', u \cup \{\langle x, b\rangle\})$, $t \cup \{\langle x, b\rangle\} \in \text{NoGoods}(I')$ and $u \cup \{\langle x, a\rangle\} \in \text{NoGoods}(I')$. Thus t, u form a GABT on $a, b \in \mathcal{D}(x)$.

We have shown that I' has a GABT on $a, b \in \mathcal{D}(x)$ if and only if I has a solution. Since the reduction from CSP(2AllDiff) to ∃GABT(2AllDiff) is clearly polynomial, this completes the proof. □

Another problem with merging values in the presence of global constraints is that the global constraint may lose its semantics when values are merged. To give an example, consider an instance I in which a variable x (with domain $\mathcal{D}(x) = A$) occurs in the scope of a single constraint, an AllDifferent constraint on variables X. Since there is only one constraint on variable x, there can be no GABT on any pair of values in $\mathcal{D}(x)$. It is easy to see that we can, in fact, GABTP-merge all the values in $\mathcal{D}(x)$. When the domain of x becomes a singleton, we can clearly eliminate x. However, the resulting constraint on the variables $X \setminus \{x\}$ combines both an AllDifferent constraint on $X \setminus \{x\}$ and a constraint which says that the set of values assigned to these variables does not contain all of A. This constraint clearly does not have the same semantics as an

AllDifferent constraint. In general, merging values can transform global constraints which have efficient filtering algorithms into new global constraints which do not have efficient filtering algorithms.

After these negative results, we now give some positive results. It turns out that we can take advantage of the semantics of (global) constraints to reduce the complexity of searching for broken triangles. Suppose that instance I contains only AllDifferent constraints. Instead of looking for GABTP-merges, we can decompose the AllDifferent constraints into binary constraints and look for BTP-merges in the resulting instance I_{bin}. The presence of a general-arity broken triangle on $a, b \in \mathcal{D}(x)$ in I implies the presence of a broken triangle on $a, b \in \mathcal{D}(x)$ in I_{bin}, but the converse is not true. Thus BT-merging in I_{bin} is a strictly weaker operation than GABT-merging in I. The advantages of BT-merging in I_{bin} is that (1) it can be detected in linear time, and (2) it conserves the semantics of the AllDifferent constraints, as we will now show.

Lemma 18. *Suppose that instance I contains only binary difference constraints $x \neq y$. For each variable x, let S_x denote the set of variables constrained by x. Two distinct values a, b in the domain of a variable x can be BTP-merged if and only if one of the following conditions holds:*

1. *there is at most one variable $y \in S_x$ such that $\{a, b\} \cap D_y \neq \emptyset$*
2. *either $\forall y \in S_x$, $a \notin D_y$ or $\forall y \in S_x$, $b \notin D_y$.*

Proof. Since I contains only difference constraints, if y, z are two distinct variables in S_x, then the pair of assignments $\langle y, a\rangle$, $\langle z, b\rangle$ are necessarily compatible. Furthermore, from Definition 4, a broken triangle on $a, b \in \mathcal{D}(x)$ necessarily consists of assignments $\langle y, a\rangle$, $\langle z, b\rangle$ where x, y, z are distinct variables. Absence of a broken triangle on $a, b \in \mathcal{D}(x)$ is thus equivalent to there being at most one variable $y \in S_x$ such that $\{a, b\} \cap D_y \neq \emptyset$, or $\forall y \in S_x$, $a \notin D_y$ or $\forall y \in S_x$, $b \notin D_y$. □

Lemma 19. *Suppose that instance I contains only binary difference constraints and that $a, b \in \mathcal{D}(x)$ are BT-free. After BT-merging of $a, b \in \mathcal{D}(x)$, the variable x can be eliminated without the introduction of new constraints, producing an instance I' which is satisfiable if and only if I is satisfiable.*

Proof. If $y \neq x$, then $\forall d \in \mathcal{D}(y)$, $\langle y, d\rangle$ is either compatible with $\langle x, a\rangle$ or $\langle x, b\rangle$, since the only possible constraint between y and x is $y \neq x$. Hence, once $a, b \in \mathcal{D}(x)$ are merged, the resulting new value c is compatible with all assignments to all other variables. It follows immediately that x and all binary constraints with x in their scope can be eliminated while preserving the satisfiability of the instance. □

Proposition 20. *If I is an instance containing only binary difference constraints, then the result of applying BTP-merges (and eliminating the corresponding variables) until convergence is unique and can be found in $O(n^2d^2)$ time and $O(nd^2)$ space, where d is the maximum domain size.*

Proof. For each variable x and for each pair of distinct values $a, b \in \mathcal{D}(x)$, we can establish in $O(n)$ time three counters $N^x_{\{a\}}$, $N^x_{\{b\}}$, $N^x_{\{ab\}}$, where $N^x_A = |\{y \mid y \in S_x \ \wedge \ A \cap \mathcal{D}(y) \neq \emptyset\}|$.

By Lemma 18, to determine whether a, b can be BTP-merged, it suffices to check whether $N^x_{\{a,b\}} \leq 1$ or $N^x_{\{a\}} = 0$ or $N^x_{\{b\}} = 0$. After each BTP-merge, and the elimination of the corresponding variable, the constraints on the remaining variables remain unchanged. Thus, when a variable y is eliminated, due to the BT-merging of two values in its domain, for each variable $x \in S_y$: for each $a \in \mathcal{D}(y) \cap \mathcal{D}(x)$, we decrement the counter $N^x_{\{a\}}$ and for each pair $a, b \in \mathcal{D}(x)$ such that $a \in \mathcal{D}(y)$ or $b \in \mathcal{D}(y)$, we decrement the counter $N^x_{\{ab\}}$. Updating these data structures can be achieved in $O(nd^2)$ each time a variable y is eliminated. Since at most n variables can be eliminated, the total time complexity is $O(n^2d^2)$. The space complexity required to store the counters is $O(nd^2)$.

We now show that all maximal sequences of BTP-merges result in the same instance. For this we observe that if $a, b \in \mathcal{D}(x)$ can be BTP-merged in an instance I, and c, d can also be BTP-merged in I, then a, b can be BTP-merged in the instance I' obtained from I by BTP-merging $c, d \in \mathcal{D}(y)$. Indeed, by Lemma 19, the BTP-merge of $c, d \in \mathcal{D}(y)$ leads immediately to the elimination of the variable y, and clearly, such elimination cannot invalidate the characterisation of Lemma 18. By symmetry it also holds that c, d can be BTP-merged in the instance obtained from I by BTP-merging a, b, hence the order of BTP-merges does not matter. □

We have seen that applying the definition of GABT-merging to CSP instances containing AllDifferent constraints is coNP-complete and can also alter the semantics of the global constraints. However, Lemma 18 provides a weaker form of merging (which is equivalent to BT-merging if the instance contains only AllDifferent constraints that have been decomposed into an equivalent set of binary difference constraints) which can be applied in $O(n^2d^2)$ time. It is worth pointing out that this is much more efficient than a brute-force application of the definition of BT-merging in a binary CSP instance until convergence, which has worst-case time complexity $O(n^4d^5)$.

9. A tractable class of general-arity CSP

In binary CSP, the broken-triangle property defines an interesting tractable class when broken triangles are forbidden according to a given variable ordering. Unfortunately, this tractable class was limited to binary CSPs [7]. Section 7 described a general-arity version of the broken-triangle property whose absence on two values allows these values to be merged while preserving satisfiability. An obvious question is whether GABT-freeness can be adapted to define a tractable class. In this section we show that this is possible for a fixed variable ordering, but not if the ordering is unknown.

Definition 11 defined a general-arity broken triangle (GABT). What happens if we forbid GABTs according to a given variable ordering? Absence of GABTs on two values a, b for the last variable x in the variable ordering allows us to merge a and b while preserving satisfiability. It is possible to show that if GABTs are absent on all pairs of values for x, then we can merge all values in the domain $\mathcal{D}(x)$ of x to produce a singleton domain. This is because (as we will show later) merging a and b, to produce a merged value c, cannot introduce a GABT on c, d for any other value $d \in \mathcal{D}(x)$. Once the domain $\mathcal{D}(x)$ becomes a singleton $\{a\}$, we can clearly eliminate x from the instance, by deleting $\langle x, a\rangle$ from all nogoods, without changing its satisfiability. It is at this moment that GABTs may be introduced on other variables, meaning that forbidding GABTs according to a variable ordering does not define a tractable class.

Nevertheless, we will show that strengthening the general-arity BTP allows us to avoid this problem. The resulting directional general-arity version of BTP (for a known variable ordering) then defines a tractable class which includes the binary BTP tractable class as a special case.

Note that the set of general-arity CSP instances whose dual instance satisfies the BTP also defines a tractable class which can be recognised in polynomial time even if the ordering of the variables in the dual instance is unknown [16]. This DBTP class is incomparable with the class we present in the present paper (which is equivalent to BTP in binary CSP) since DBTP is known to be incomparable with the BTP class already in the special case of binary CSP [16]. A general-arity broken triangle can be said to be centred on a pair of *values* in the domain of a variable whereas a broken triangle in the dual instance is centred on a pair of *tuples* in a constraint relation. One consequence of this is that eliminating tuples from constraint relations cannot introduce broken triangles in the dual instance, whereas the (directional) GABTP is only invariant under elimination of domain values. On the other hand, the (directional) GABTP is invariant under adding a complete constraint (i.e. whose relation is the direct product of the domains of the variables in its scope) whereas this operation can introduce broken triangles in the dual instance. Another important difference is that directional GABTP depends on an order on the variables whereas DBTP depends on an order on the constraints.

9.1. Directional general-arity BTP

Recall that we assume that a CSP instance I is given in the form of a set of incompatible tuples NoGoods(I), where a tuple is a set of variable-value assignments, and that the predicate Good(I, t) is true iff the tuple t does not contain any pair of distinct assignments to the same variable and $\nexists t' \subseteq t$ such that $t' \in$ NoGoods(I). We suppose given a total ordering $<$ of the variables of a CSP instance I. We write $t^{<x}$ to represent the subset of the tuple t consisting of assignments to variables occurring before x in the order $<$, and $Vars(t)$ to denote the set of all variables assigned by t.

Definition 21. A *directional general-arity (DGA) broken triangle* on assignments a, b to variable x in a CSP instance I is a pair of tuples t, u (containing no assignments to variable x) satisfying the following conditions:

1. $t^{<x}$ and $u^{<x}$ are non-empty
2. Good($I, t^{<x} \cup u^{<x}$) $\wedge$ Good($I, t^{<x} \cup \{\langle x, a\rangle\}$) $\wedge$ Good($I, u^{<x} \cup \{\langle x, b\rangle\}$)
3. $\exists t'$ s.t. $Vars(t') = Vars(t)$ $\wedge$ $(t')^{<x} = t^{<x}$ $\wedge$ $t' \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)
4. $\exists u'$ s.t. $Vars(u') = Vars(u)$ $\wedge$ $(u')^{<x} = u^{<x}$ $\wedge$ $u' \cup \{\langle x, b\rangle\} \notin$ NoGoods(I)
5. $t \cup \{\langle x, b\rangle\} \in$ NoGoods(I) $\wedge$ $u \cup \{\langle x, a\rangle\} \in$ NoGoods(I)

I satisfies the *directional general-arity broken-triangle property (DGABTP)* according to the variable ordering $<$ if no directional general-arity broken triangle occurs on any pair of values a, b for any variable x.

Points (1), (2) and (5) of Definition 21 are illustrated by Fig. 18. The two important differences compared to a general-arity broken triangle (Fig. 17) are that there is now a variable ordering $<$, with $y < x$ for all variables $y \in Y \cup Z$, and the two dashed lines now represent nogoods $u \cup \{\langle x, a\rangle\}$ and $t \cup \{\langle x, b\rangle\}$ which possibly involve assignments to variables $w > x$.

We will show that any instance I satisfying the DGABTP can be solved in polynomial time by repeatedly alternating the following two operations: (i) merge all values in the last remaining variable (according to the order $<$); (ii) eliminate this variable when its domain becomes a singleton. We will give the two operations (merging and variable-elimination) and show that both operations preserve satisfiability and that neither of them can introduce DGA broken triangles. Moreover, as for GABT-freeness, the DGABTP can be tested in polynomial time for a given order whether constraints are given as tables of satisfying assignments or as nogoods. Indeed, in the former case, using items (3) and (4) in Definition 21 we can restrict the search for a DGA broken triangle to pairs of tuples satisfying some constraint (there must be a constraint with scope

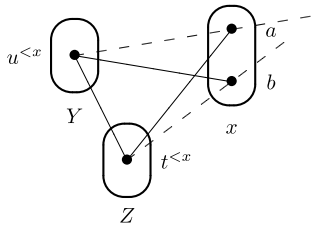


Fig. 18. Illustration of a directional general-arity broken triangle.

$Vars(t' \cup \{x\})$ since there is a nogood on these variables by item (5), and similarly for u'). This is sufficient to define a tractable class.

9.2. Merging

Let x be the last variable according to the variable order $<$. When values a, b in the domain of variable x do not belong to any DGA broken triangle, we can replace a, b by a new value c to produce an instance I' with the new set of nogoods given by Definition 12. Since x is the last variable in the ordering $<$, DGA broken triangles on $a, b \in \mathcal{D}(x)$ are GA broken triangles (and vice versa). Thus, from Proposition 13 we can deduce that satisfiability is preserved by this merging operation. What remains to be shown is that merging two values in the domain of the last variable cannot introduce the forbidden pattern.

Lemma 22. *Merging two values a, b into a value c in the domain of the last variable x (according to a DGABTP variable order $<$) in an instance I cannot introduce a directional general-arity broken triangle (DGABT) in the resulting instance I'.*

Proof. We first claim that this operation cannot introduce a DGABT on a variable $y < x$. Indeed, assume there is a DGABT on $d, e \in \mathcal{D}(y)$ in I', that is, that there are tuples v, w such that

1. $v^{<y}$ and $w^{<y}$ are non-empty
2. $\text{Good}(I', v^{<y} \cup w^{<y}) \wedge \text{Good}(I', v^{<y} \cup \{\langle y, d\rangle\}) \wedge \text{Good}(I', w^{<y} \cup \{\langle y, e\rangle\})$
3. $\exists v'\ Vars(v') = Vars(v) \ \wedge\ (v')^{<y} = v^{<y} \ \wedge\ v' \cup \{\langle y, d\rangle\} \notin \text{NoGoods}(I')$
4. $\exists w'\ Vars(w') = Vars(w) \ \wedge\ (w')^{<y} = w^{<y} \ \wedge\ w' \cup \{\langle y, e\rangle\} \notin \text{NoGoods}(I')$
5. $v \cup \{\langle y, e\rangle\} \in \text{NoGoods}(I') \wedge w \cup \{\langle y, d\rangle\} \in \text{NoGoods}(I')$

If v' contains the assignment $\langle x, c\rangle$ then, by construction of $\text{NoGoods}(I')$ (Definition 12), $\exists v'' \in \{(v' \setminus \langle x, c\rangle) \cup \{\langle x, a\rangle\}, (v' \setminus \langle x, c\rangle) \cup \{\langle x, b\rangle\}\}$ such that $v'' \cup \{\langle y, d\rangle\} \notin \text{NoGoods}(I)$. If v' does not contain $\langle x, c\rangle$ then let $v'' = v'$. Define w'' in a similar way. Now considering the last item, if v contains $\langle x, c\rangle$ then by construction of $\text{NoGoods}(I')$ there is v''' assigning a or b to x and otherwise equal to v, such that $v''' \cup \{\langle y, e\rangle\}$ was in $\text{NoGoods}(I)$, and if $v \not\ni \langle x, c\rangle$ we let $v''' = v$. We define w''' similarly. Then:

1. $(v''')^{<y} = v^{<y}$ and $(w''')^{<y} = w^{<y}$ are non-empty
2. $\text{Good}(I, (v''')^{<y} \cup (w''')^{<y}) \wedge \text{Good}(I, (v''')^{<y} \cup \{\langle y, d\rangle\}) \wedge \text{Good}(I, (w''')^{<y} \cup \{\langle y, e\rangle\})$ (since x is the last variable, $(v''')^{<y} = v^{<y}$ and $(w''')^{<y} = w^{<y}$)
3. $Vars(v'') = Vars(v''') \ \wedge\ (v'')^{<y} = (v''')^{<y} \ \wedge\ v'' \cup \{\langle y, d\rangle\} \notin \text{NoGoods}(I)$
4. $Vars(w'') = Vars(w''') \ \wedge\ (w'')^{<y} = (w''')^{<y} \ \wedge\ w'' \cup \{\langle y, e\rangle\} \notin \text{NoGoods}(I))$
5. $v''' \cup \{\langle y, e\rangle\} \in \text{NoGoods}(I) \ \wedge\ w''' \cup \{\langle y, d\rangle\} \in \text{NoGoods}(I)$

that is, there was a DGABT on d, e in I, contradicting our assumption.

We now show that a broken triangle cannot be introduced on x. Observe that since x is the last variable, for all tuples t not containing an assignment to x, $t^{<x} = t$ holds. We use this tacitly in the rest of the proof. Suppose for a contradiction that I contained no DGABT, but that after merging $a, b \in D(x)$ in I to produce the instance I', in which a, b have been replaced by a new value c, we have a DGABT on c, d. Then there is a pair of non-empty tuples t, u (containing no assignments to variable x) satisfying in particular the following conditions:

(1) $\text{Good}(I', t \cup u)$
(2) $\text{Good}(I', t \cup \{\langle x, c\rangle\})$
(3) $\text{Good}(I', u \cup \{\langle x, d\rangle\})$
(4) $t \cup \{\langle x, d\rangle\} \in \text{NoGoods}(I')$
(5) $u \cup \{\langle x, c\rangle\} \in \text{NoGoods}(I')$

We show that there was a DGABT in I either on a, d, on b, d or on a, b.

Since merging only affects tuples containing $\langle x, a\rangle$ or $\langle x, b\rangle$, (1) implies that Good$(I, t \cup u)$ and hence Good$(I, t \cup u')$ for all $u' \subseteq u$. Similarly, (3) implies that Good$(I, u \cup \{\langle x, d\rangle\})$ and hence Good$(I, u' \cup \{\langle x, d\rangle\})$ for all $u' \subseteq u$. Similarly, (4) implies that $t \cup \{\langle x, d\rangle\} \in$ NoGoods(I).

There are three possible cases to consider:

(a) Good$(I, t \cup \{\langle x, a\rangle\})$,
(b) Good$(I, t \cup \{\langle x, b\rangle\})$,
(c) $\exists t_1, t_2 \subseteq t$ such that $t_1 \cup \{\langle x, a\rangle\}$, $t_2 \cup \{\langle x, b\rangle\} \in$ NoGoods(I).

Case (a): By Definition 12 of the creation of nogoods during merging, (5) implies that $\exists u' \subseteq u$ such that $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I). We know that u' is non-empty since $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I) but Good$(I, t \cup \{\langle x, a\rangle\})$ (and hence Good$(I, \{\langle x, a\rangle\})$). We have Good$(I, t \cup u')$, Good$(I, t \cup \{\langle x, a\rangle\})$ (and hence $t \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)), Good$(I, u' \cup \{\langle x, d\rangle\})$ (and hence $u' \cup \{\langle x, d\rangle\} \notin$ NoGoods(I)), $t \cup \{\langle x, d\rangle\} \in$ NoGoods(I), $u' \cup \{\langle x, a\rangle\} \in$ NoGoods(I) and hence there was a DGABT on a, d in I.
Case (b): Symmetrically to case (a), there was a DGABT on b, d in I.
Case (c): We claim that Good$(I, t_1 \cup \{\langle x, b\rangle\})$. If not, then we would have $\exists t_3 \subseteq t_1$ such that $t_3 \cup \{\langle x, b\rangle\} \in$ NoGoods(I) which would imply $t_1 \cup \{\langle x, c\rangle\} \in$ NoGoods(I') which is impossible since, by (2) above, we have Good$(I', t \cup \{\langle x, c\rangle\})$. By a symmetrical argument, we can deduce Good$(I, t_2 \cup \{\langle x, a\rangle\})$. Since Good$(I, t \cup u)$ and $t_1, t_2 \subseteq t$, we have Good$(I, t_1 \cup t_2)$. Since $t_1 \cup \{\langle x, a\rangle\} \in$ NoGoods(I) and Good$(I, t_2 \cup \{\langle x, a\rangle\})$ (and hence Good$(I, \{\langle x, a\rangle\})$), we must have $t_1 \neq \emptyset$. By a symmetric argument, $t_2 \neq \emptyset$. We therefore have non-empty tuples t_1, t_2 such that Good$(I, t_1 \cup t_2)$, Good$(I, t_1 \cup \{\langle x, b\rangle\})$ (and hence $t_1 \cup \{\langle x, b\rangle\} \notin$ NoGoods(I)), Good$(I, t_2 \cup \{\langle x, a\rangle\})$ (and hence $t_2 \cup \{\langle x, a\rangle\} \notin$ NoGoods(I)), $t_1 \cup \{\langle x, a\rangle\} \in$ NoGoods(I), $t_2 \cup \{\langle x, b\rangle\} \in$ NoGoods(I) and hence we have a DGABT in I on a, b.

Since in each of the three possible cases, we produced a contradiction, this completes the proof. □

9.3. Tractability of DGABTP for a known variable ordering

We are now in a position to give a new tractable class of general-arity CSP instances based on the DGABTP.

Theorem 23. *A CSP instance I satisfying the DGABTP on a given variable ordering can be solved in polynomial time.*

Proof. Suppose that I satisfies the DGABTP for variable ordering $<$ and that x is the last variable according to this ordering. Lemma 22 tells us that DGA broken triangles cannot be introduced by merging all elements in $\mathcal{D}(x)$ to form a singleton domain $\{a\}$. At this point it may be that $\{\langle x, a\rangle\}$ is a nogood. In this case the instance is clearly unsatisfiable and the algorithm halts returning this result. If not then we simply delete $\langle x, a\rangle$ from all nogoods in which it occurs. This operation of variable elimination clearly preserves satisfiability. It is polynomial time to recursively apply this merging and variable elimination algorithm until a nogood corresponding to a singleton domain is discovered or until all variables have been eliminated (in which case I is satisfiable).

To complete the proof of correction of this algorithm, it only remains to show that elimination of the last variable x cannot introduce a DGA broken triangle on another variable y. For all tuples t, u and all values $c, d \in D(y)$, none of Good$(I, t^{<y} \cup u^{<y})$, Good$(I, t^{<y} \cup \{\langle y, c\rangle\})$ and Good$(I, u^{<y} \cup \{\langle y, d\rangle\})$ can become true due to the variable elimination operation described above. On the other hand it is possible that $t \cup \{\langle y, d\rangle\}$ or $u \cup \{\langle y, c\rangle\}$ becomes a nogood due to variable elimination. Without loss of generality, suppose that $t \cup \{\langle y, d\rangle\}$ becomes a nogood and that $t' \cup \{\langle y, d\rangle\}$ is not a nogood for some t' such that $Vars(t') = Vars(t)$ and $(t')^{<y} = t^{<y}$. Then by construction there was a nogood $t \cup \{\langle y, d\rangle\} \cup \{\langle x, a\rangle\}$ before the variable x (with singleton domain $\{a\}$) was eliminated, and $t' \cup \{\langle y, d\rangle\} \cup \{\langle x, a\rangle\}$ was not a nogood. But then there was a DGA broken triangle (given by tuples $t \cup \{\langle x, a\rangle\}$, u on values $c, d \in D(y)$) before elimination of x. This contradiction shows that variable elimination cannot introduce DGA broken triangles. □

9.4. Finding a DGABTP variable ordering is NP-hard

An important question is the tractability of the recognition problem of the class DGABTP when the variable order is not given, i.e. testing the existence of a variable ordering for which a given instance satisfies the DGABTP. In the case of binary CSP, this test can be performed in polynomial time [7]. Unfortunately, as the following theorem shows, the problem becomes NP-complete in the general-arity case.

When a DGABTP ordering exists, there is at least one variable x such that all pairs of values $a, b \in \mathcal{D}(x)$ are GABT-free. In fact there may be several such variables which are all candidates for being the last variable in the DGABTP ordering. For any such variable x, after merging all values in the domain $\mathcal{D}(x)$ so that it becomes a singleton $\{a\}$, we can eliminate x from the instance, by deleting $\langle x, a\rangle$ from all nogoods, without changing its satisfiability. It is at this moment that DGABTs may be introduced on other variables. In the binary case, we can eliminate all such variables without the risk of introducing broken triangles. This is because deleting $\langle x, a\rangle$ from a binary nogood, such as $\{\langle x, a\rangle, \langle y, b\rangle\}$, produces the unary nogood $\langle y, b\rangle$ corresponding to the elimination of b from $\mathcal{D}(y)$ and the DGABTP cannot be destroyed by such domain reductions. In

the general-arity case, on the other hand, we cannot use such a greedy algorithm since the elimination of such a variable x may destroy the DGABTP for the as-yet-unknown variable ordering $<$ if x is not the last variable according to $<$.

Theorem 24. *Testing the existence of a variable ordering for which a CSP instance satisfies the DGABTP is NP-complete (even if the arity of constraints is at most 5).*

Proof. The problem is in NP since verifying the DGABTP is polytime for a given order, so it suffices to give a polynomial-time reduction from the well-known NP-complete problem 3SAT. Let I_{3SAT} be an instance of 3SAT with variables $X_1, \ldots, X_N$ and clauses $C_1, \ldots, C_M$. We will create a CSP instance I_{CSP} which has a DGABTP variable-ordering if and only if I_{3SAT} is satisfiable. For each variable X_i of I_{3SAT}, we add two variables x_i, y_i to I_{CSP}. To complete the set of variables in I_{CSP}, we add three special variables v, w, z. We add constraints to I_{CSP} in such a way that each DGABTP ordering of its variables corresponds to a solution to I_{3SAT} (and vice versa). The role of the variable z is critical: a DGABTP ordering $>$ of the variables of I_{CSP} corresponds to a solution to I_{3SAT} in which $X_i = \texttt{true} \Leftrightarrow x_i > z$. The variables y_i are used to code $\overline{X_i}$: $y_i > z$ in a DGABTP ordering if and only if $X_i = \texttt{false}$ in the corresponding solution to I_{3SAT}. The variables v, w are necessary for our construction and will necessarily satisfy $v, w < z$ in a DGABTP ordering. Each clause $C = l_1 \vee l_2 \vee l_3$, where l_1, l_2, l_3 are literals in I_{3SAT}, is imposed in I_{CSP} by adding constraints which force one of $\overline{l_1}$, $\overline{l_2}$, $\overline{l_3}$ to be false. To give a concrete example, if $C = X_1 \vee X_2 \vee X_3$, then constraints are added to I_{CSP} to force $y_1 < z$ or $y_2 < z$ or $y_3 < z$ in a DGABTP ordering. If the clause C contains a negated variable $\overline{X_i}$ instead of X_i, it suffices to replace y_i by x_i.

We now give in detail the necessary gadgets in I_{CSP} to enforce each of the following properties in a DGABTP ordering:

1. $v, w < z$
2. $y_i < z \Leftrightarrow x_i > z$
3. $y_i < z$ or $y_j < z$ or $y_k < z$

We introduce broken triangles in order to impose these properties. However, it is important not to inadvertently introduce other broken triangles. This can be avoided by making all pairs of assignments $\langle x, a\rangle$, $\langle x', a'\rangle$ from two different gadgets incompatible (i.e. $\{\langle x, a\rangle, \langle x', a'\rangle\} \in \text{NoGoods}(I_{CSP})$). We also assume that two gadgets which use the same variable x use distinct domain values in $\mathcal{D}(x)$. To avoid creating a trivial instance in which the gadgets disappear after establishing arc consistency, we can also add extra values in each domain which are compatible with all variable-value assignments in the gadgets.

We give the details of the three types of gadget:

1. The gadget to force $v, w < z$ in a DGABTP ordering consists of values $a_0 \in \mathcal{D}(z)$, $b_0, b_1 \in \mathcal{D}(v)$, $c_0, c_1 \in \mathcal{D}(w)$ and three nogoods $\{\langle z, a_0\rangle, \langle v, b_0\rangle\}$, $\{\langle z, a_0\rangle, \langle w, c_0\rangle\}$, $\{\langle v, b_1\rangle, \langle w, c_1\rangle\}$. The only way to satisfy the DGABTP on this triple of variables is to have $v, w < z$ since there are broken triangles on variables v and w.
2. To force $y_i < z \Leftrightarrow x_i > z$ in a DGABTP ordering we use two gadgets, the first to force $y_i > z \vee x_i > z$ and the second to force $y_i < z \vee x_i < z$.
   The first gadget is a broken triangle consisting of values $a_1, a_2 \in \mathcal{D}(z)$, $d_0 \in \mathcal{D}(x_i)$, $e_0 \in \mathcal{D}(y_i)$ and two nogoods $\{\langle z, a_1\rangle, \langle x_i, d_0\rangle\}$, $\{\langle z, a_2\rangle, \langle y_i, e_0\rangle\}$. In a DGABTP ordering we must have $y_i > z \vee x_i > z$.
   The second gadget consists of values $a_3, a_4 \in \mathcal{D}(z)$, $b_2 \in \mathcal{D}(v)$, $c_2 \in \mathcal{D}(w)$, $d_1 \in \mathcal{D}(x_i)$, $e_1 \in \mathcal{D}(y_i)$ and four nogoods $\{\langle z, a_3\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $\{\langle z, a_4\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $\{\langle z, a_4\rangle, \langle w, c_2\rangle, \langle y_i, e_1\rangle\}$, $\{\langle z, a_3\rangle, \langle w, c_2\rangle, \langle y_i, e_1\rangle\}$. We assume that we have forced $v, w < z$ using the gadget described in point (1). The tuples $t = \{\langle v, b_2\rangle, \langle x_i, d_1\rangle\}$, $u = \{\langle w, c_2\rangle, \langle y_i, e_1\rangle\}$ then form a DGA broken triangle on assignments $a_3, a_4 \in \mathcal{D}(z)$ if $x_i, y_i > z$. If either $x_i < z$ or $y_i < z$ then there is no DGA broken triangle; for example, if $x_i < z$, then we no longer have $\text{Good}(I_{CSP},\ t^{<z} \cup \{\langle z, a_3\rangle\})$ since $t^{<z} \cup \{\langle z, a_3\rangle$ is precisely the nogood $\{\langle z, a_3\rangle, \langle v, b_2\rangle, \langle x_i, d_1\rangle\}$. Thus this gadget forces $y_i < z \vee x_i < z$ in a DGABTP ordering.
3. The gadget to force $y_i < z$ or $y_j < z$ or $y_k < z$ in a DGABTP ordering consists of values $a_5, a_6 \in \mathcal{D}(z)$, $b_3 \in \mathcal{D}(v)$, $c_3 \in \mathcal{D}(w)$, $e_2 \in \mathcal{D}(y_i)$, $e_3 \in \mathcal{D}(y_j)$, $e_4 \in \mathcal{D}(y_k)$ and five nogoods: $\{\langle z, a_6\rangle, \langle v, b_3\rangle, \langle y_i, e_2\rangle, \langle y_j, e_3\rangle, \langle y_k, e_4\rangle\}$, $\{\langle z, a_5\rangle, \langle w, c_3\rangle\}$, $\{\langle z, a_5\rangle, \langle y_i, e_2\rangle\}$, $\{\langle z, a_5\rangle, \langle y_j, e_3\rangle\}$, $\{\langle z, a_5\rangle, \langle y_k, e_4\rangle\}$. The tuples $t = \{\langle v, b_3\rangle, \langle y_i, e_2\rangle, \langle y_j, e_3\rangle, \langle y_k, e_4\rangle\}$, $u = \{\langle w, c_3\rangle\}$ form a DGA broken triangle on $a_5, a_6 \in \mathcal{D}(a)$ if $y_i, y_j, y_k > z$. If $y_i < z$ or $y_j < z$ or $y_k < z$, then there is no DGA broken triangle; for example, if $y_i < z$, then we no longer have $\text{Good}(I_{CSP},\ t^{<z} \cup \{\langle z, a_5\rangle\})$ since $\{\langle z, a_5\rangle, \langle y_i, e_2\rangle\}$ is a nogood. Thus this gadget forces $y_i < z$ or $y_j < z$ or $y_k < z$ in a DGABTP ordering.

The above gadgets allow us to code I_{3SAT} as the problem of testing the existence of a DGABTP ordering in the corresponding instance I_{CSP}. To complete the proof it suffices to observe that this reduction is clearly polynomial. □

Our proof of Theorem 24 used large domains. The question still remains whether it is possible to detect in polynomial time whether a DGABTP variable ordering exists in the case of domains of bounded size, and in particular in the important case of SAT.

10. Conclusion

This paper described a novel reduction operation for binary CSP, called BTP-merging, which is strictly stronger than neighbourhood substitution. Experimental trials have shown that in several benchmark-domains, applying BTP-merging until convergence can significantly reduce the total number of variable-value assignments. We gave a general-arity version of BTP-merging and demonstrated a theoretical link with resolution in SAT. From a theoretical point of view, we then went on to define a general-arity version of the tractable class defined by the broken-triangle property for a known variable ordering. Our investigation of the interaction of BTP-merging and AllDifferent constraints has shown that the semantics of binary difference constraints can allow us to speed up the search for BTP-merges. An interesting avenue of future research is to try to take advantage of the semantics of other types of constraints to speed up the search for BTP-merges.

Acknowledgements

This research was supported by ANR Project ANR-10-BLAN-0210. Martin Cooper was also supported by EPSRC grant EP/L021226/1.

References

[1] D.A. Cohen, P.G. Jeavons, The complexity of constraint languages, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006, pp. 245–280.
[2] N. Creignou, P.G. Kolaitis, H. Vollmer (Eds.), Complexity of Constraints – An Overview of Current Research Themes [Result of a Dagstuhl Seminar], Lect. Notes Comput. Sci., vol. 5250, Springer, 2008, http://dx.doi.org/10.1007/978-3-540-92800-3.
[3] M. Grohe, The complexity of homomorphism and constraint satisfaction problems seen from the other side, J. ACM 54 (1) (2007), http://doi.acm.org/10.1145/1206035.1206036.
[4] D. Marx, Tractable hypergraph properties for constraint satisfaction and conjunctive queries, J. ACM 60 (6) (2013) 42, http://doi.acm.org/10.1145/2535926.
[5] P. Jégou, Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems, in: R. Fikes, W.G. Lehnert (Eds.), Proceedings of the 11th National Conference on Artificial Intelligence, Washington, DC, USA, July 11–15, 1993, AAAI Press/The MIT Press, 1993, pp. 731–736, http://www.aaai.org/Library/AAAI/1993/aaai93-109.php.
[6] A.Z. Salamon, P.G. Jeavons, Perfect constraints are tractable, in: P.J. Stuckey (Ed.), Principles and Practice of Constraint Programming, Proceedings of the 14th International Conference, CP 2008, Sydney, Australia, September 14–18, 2008, in: Lect. Notes Comput. Sci., vol. 5202, Springer, 2008, pp. 524–528, http://dx.doi.org/10.1007/978-3-540-85958-1_35.
[7] M.C. Cooper, P.G. Jeavons, A.Z. Salamon, Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination, Artif. Intell. 174 (9–10) (2010) 570–584, http://dx.doi.org/10.1016/j.artint.2010.03.002.
[8] D.A. Cohen, M.C. Cooper, P. Creed, D. Marx, A.Z. Salamon, The tractability of CSP classes defined by forbidden patterns, J. Artif. Intell. Res. 45 (2012) 47–78, http://dx.doi.org/10.1613/jair.3651.
[9] M.C. Cooper, G. Escamocher, Characterising the complexity of constraint satisfaction problems defined by 2-constraint forbidden patterns, Discrete Appl. Math. 184 (2015) 89–113, http://dx.doi.org/10.1016/j.dam.2014.10.035.
[10] M.C. Cooper, S. Živný, Tractable triangles and cross-free convexity in discrete optimisation, J. Artif. Intell. Res. 44 (2012) 455–490, http://dx.doi.org/10.1613/jair.3598.
[11] D.A. Cohen, M.C. Cooper, G. Escamocher, S. Živný, Variable and value elimination in binary constraint satisfaction via forbidden patterns, J. Comput. Syst. Sci. 81 (7) (2015) 1127–1143, http://dx.doi.org/10.1016/j.jcss.2015.02.001.
[12] M.C. Cooper, Beyond consistency and substitutability, in: O'Sullivan [31], pp. 256–271, http://dx.doi.org/10.1007/978-3-319-10428-7_20.
[13] P. Jégou, C. Terrioux, The extendable-triple property: a new CSP tractable class beyond BTP, in: B. Bonet, S. Koenig (Eds.), Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, January 25–30, 2015, AAAI Press, 2015, pp. 3746–3754, http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9939.
[14] M.C. Cooper, P. Jégou, C. Terrioux, A microstructure-based family of tractable classes for CSPs, in: Pesant [30], pp. 74–88, http://dx.doi.org/10.1007/978-3-319-23219-5_6.
[15] W. Naanaa, Unifying and extending hybrid tractable classes of CSPs, J. Exp. Theor. Artif. Intell. 25 (4) (2013) 407–424, http://dx.doi.org/10.1080/0952813X.2012.721138.
[16] A.E. Mouelhi, P. Jégou, C. Terrioux, A hybrid tractable class for non-binary CSPs, Constraints 20 (4) (2015) 383–413, http://dx.doi.org/10.1007/s10601-015-9185-y.
[17] J. Gao, M. Yin, J. Zhou, Hybrid tractable classes of binary quantified constraint satisfaction problems, in: W. Burgard, D. Roth (Eds.), Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, CA, USA, August 7–11, 2011, AAAI Press, 2011, http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3507.
[18] M.C. Cooper, A. El Mouelhi, C. Terrioux, B. Zanuttini, On broken triangles, in: O'Sullivan [31], pp. 9–24, http://dx.doi.org/10.1007/978-3-319-10428-7_5.
[19] M.C. Cooper, A. Duchein, G. Escamocher, Broken triangles revisited, in: Pesant [30], pp. 58–73, http://dx.doi.org/10.1007/978-3-319-23219-5_5.
[20] C. Likitvivatanavong, R.H. Yap, Eliminating redundancy in CSPs through merging and subsumption of domain values, ACM SIGAPP Appl. Comput. Rev. 13 (2) (2013).
[21] E.C. Freuder, Eliminating interchangeable values in constraint satisfaction problems, in: T.L. Dean, K. McKeown (Eds.), Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14–19, 1991, vol. 1, AAAI Press/MIT Press, 1991, pp. 227–233, http://www.aaai.org/Library/AAAI/1991/aaai91-036.php.
[22] M.C. Cooper, Fundamental properties of neighbourhood substitution in constraint satisfaction problems, Artif. Intell. 90 (1–2) (1997) 1–24, http://dx.doi.org/10.1016/S0004-3702(96)00018-5.
[23] C. Bessière, J. Régin, Refining the basic constraint propagation algorithm, in: B. Nebel (Ed.), Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4–10, 2001, Morgan Kaufmann, 2001, pp. 309–315.
[24] D. Sabin, E.C. Freuder, Contradicting conventional wisdom in constraint satisfaction, in: A. Borning (Ed.), Principles and Practice of Constraint Programming, Proceedings of the Second International Workshop, PPCP'94, Rosario, Orcas Island, Washington, USA, May 2–4, 1994, in: Lect. Notes Comput. Sci., vol. 874, Springer, 1994, pp. 10–20, http://dx.doi.org/10.1007/3-540-58601-6_86.

[25] F. Boussemart, F. Hemery, C. Lecoutre, L. Saïs, Boosting systematic search by weighting constraints, in: R.L. de Mántaras, L. Saitta (Eds.), Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004, Including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22–27, 2004, IOS Press, 2004, pp. 146–150.

[26] A. El Mouelhi, P. Jégou, C. Terrioux, Hidden tractable classes: from theory to practice, in: 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10–12, 2014, IEEE Computer Society, 2014, pp. 437–445, http://dx.doi.org/10.1109/ICTAI.2014.73.

[27] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: F. Bacchus, T. Walsh (Eds.), Theory and Applications of Satisfiability Testing, Proceedings of the 8th International Conference, SAT 2005, St. Andrews, UK, June 19–23, 2005, in: Lect. Notes Comput. Sci., vol. 3569, Springer, 2005, pp. 61–75, http://dx.doi.org/10.1007/11499107_5.

[28] W.-J. van Hoeve, I. Katriel, Global constraints, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006, pp. 169–208.

[29] M. Kutz, K.M. Elbassioni, I. Katriel, M. Mahajan, Simultaneous matchings: hardness and approximation, J. Comput. Syst. Sci. 74 (5) (2008) 884–897, http://dx.doi.org/10.1016/j.jcss.2008.02.001.

[30] G. Pesant (Ed.), Principles and Practice of Constraint Programming – Proceedings of the 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Lect. Notes Comput. Sci., vol. 9255, Springer, 2015, http://dx.doi.org/10.1007/978-3-319-23219-5.

[31] B. O'Sullivan (Ed.), Principles and Practice of Constraint Programming – Proceedings of the 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014, Lect. Notes Comput. Sci., vol. 8656, Springer, 2014, http://dx.doi.org/10.1007/978-3-319-10428-7.

ELSEVIER

Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Artificial Intelligence 146 (2003) 43–75

Artificial Intelligence

www.elsevier.com/locate/artint

Hybrid backtracking bounded by tree-decomposition of constraint networks

Philippe Jégou, Cyril Terrioux *

Laboratoire des Sciences de l'Information et des Systèmes, LSIS-CNRS, Université d'Aix-Marseille 3, Av. Escadrille Normandie-Niemen, 13397 Marseille Cedex 20, France

Received 23 January 2002

Abstract

We propose a framework for solving CSPs based both on backtracking techniques and on the notion of tree-decomposition of the constraint networks. This mixed approach permits us to define a new framework for the enumeration, which we expect that it will benefit from the advantages of two approaches: a practical efficiency of enumerative algorithms and a warranty of a limited time complexity by an approximation of the tree-width of the constraint networks. Finally, experimental results allow us to show the advantages of this approach.

Keywords: Constraint networks; Time-space; Hybrid algorithms; Tree-decomposition; Empirical evaluation

1. Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. Formulating a problem as a CSP consists in defining a set X of variables $x_1, x_2, \ldots, x_n$, which must be assigned in their respective finite domain D_i, by satisfying a set C of constraints which express restrictions between the different possible assignments. A solution is an assignment of every variable which satisfies all constraints. Many academic or real problems can be formulated in this framework. This formal framework allows the expression of NP-complete problems.

* Corresponding author.
E-mail addresses: philippe.jegou@univ.u-3mrs.fr (P. Jégou), cyril.terrioux@univ.u-3mrs.fr (C. Terrioux).

0004-3702/03/$ – see front matter © 2003 Published by Elsevier Science B.V.
doi:10.1016/S0004-3702(02)00400-9

The usual method for solving CSPs is based on backtracking search, which, in order to be efficient, must use both filtering techniques and heuristics for choosing the next variable or value. This approach, often efficient in practice, has an exponential theoretical complexity in $O(m.d^n)$ where n and m are respectively the number of variables and the number of constraints of the treated instance, while d is the maximum size of domains.

Several works have been developed, in order to provide bounds of the theoretical complexity according to particular features of the instance, like for example the acyclicity of a constraint network [14,16]. The best known bounds of complexity are given by the "tree-width" of a CSP, i.e., a parameter associated with the graph which represents the interactions between variables via the constraints. Different methods are proposed like the *Tree-Clustering* [15] (see [19] for a survey about these methods and their theoretical comparison). Tree-Clustering is based on the notion of tree-decomposition of the graph. It aims to represent any constraint network by covering the constraints by cliques, whose arrangement is a tree. The new structure must be equivalent in terms of set of solutions. The best decomposition leads to a time complexity in $O(n.d^{w+1})$, where w is the tree-width of the network [29]. Depending on the instances, the effective gain may be significant with respect to enumerative approaches. Yet, the space complexity, which is not considered for the backtracking because it is generally linear, may make such an approach absolutely ineffective in practice. It can be reduced to $O(n.s.d^s)$ where s is the maximum size of minimal separators of the network [13].

The purpose of this contribution is to propose an alternative way which aims to benefit from backtracking for its practical efficiency while giving bounds of complexity which will be ones provided by structural approaches. The main idea of our approach is that backtracking search will be guided, for the choice of variables, by the structure of the network's tree-decomposition. The order imposed to enumeration will allow to exploit two notions. The first one is the notion of "structural nogood". It is a nogood in the classical sense of the term (i.e., a partial assignment of the set of variables which cannot be extended to a solution [34]), but we only find it thanks to structural properties. It will be used for pruning the tree search by cuts which permit not to explore inconsistent subtrees. The second notion is one of "structural good". A good is a partial assignment which has at least a consistent extension on a well-identified part of the problem. A good will be detected by structural criteria. The pruning induced by goods is used to cut branches of the search tree in order to avoid exploring consistent subtrees. In some respects, exploiting goods leads to realize a "forward-jump" in the search tree, by analogy with the classical and reverse terminology of backjumping. Note that related notions of goods and nogoods based on structural properties have been introduced in [5] but these notions are formally different.

The exploitation of the structure through the notions of structural goods and nogoods is at the root of our scheme of enumerative resolution. We will explain how this approach can guarantee a theoretical time bound, which is at most $O(n.d^{w+1})$ if we get an optimal tree-decomposition of the network, while limiting the space complexity to $O(n.s.d^s)$. The given bounds are in the worst case; so we will show that our approach is always more efficient than Tree-Clustering because our method requires less time and less space. Experimental results will confirm these features.

In Section 2, we remember the main notations and results about CSPs as well as the notions of graph theory exploited in tree-decomposition methods. Section 3 presents the method and justifies its validity. In Section 4, we then provide comparative theoretical results and time and space complexities. Section 5 presents some experimental results, Section 6 recalls some related works, and we finally give some perspectives which are offered by our approach in Section 7.

2. Preliminaries

2.1. Notations

Formally, a *Constraint Satisfaction Problem* is defined by a quadruplet $\mathcal{P} = (X, D, C, R)$ with $X = \{x_1, x_2, \ldots, x_n\}$ a finite set of variables and $D = \{D_1, D_2, \ldots, D_n\}$ a finite set of domains such that D_i is the finite set of values which the variable x_i can take. $C = \{C_1, C_2, \ldots, C_m\}$ is a finite set of constraints such that a constraint C_i is defined by a set of variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_{j_i}}\}$ and $R = \{R_1, R_2, \ldots, R_m\}$ is a finite set of relations over the domains of variables of each constraint, i.e., a relation is associated with each C_i such that $R_i \subseteq D_{i_1} \times \cdots \times D_{i_{j_i}}$. The relation R_i defines the allowed assignments of variables, i.e., the assignments which satisfy the constraint C_i.

Given such a quadruplet, different queries can be formulated, like the decision problem which concerns the existence of an assignment of variables satisfying all the constraints, i.e., does a function $f : X \to \bigcup_{i=1}^{n} D_i$ exist such that $\forall i,\ 1 \leqslant i \leqslant m$, $(f(x_{i_1}), f(x_{i_2}), \ldots, f(x_{i_{j_i}})) \in R_i$. If such a function exists, then f is a solution of $\mathcal{P}$. The CSP problem is NP-complete.

Afterwards, we call *binary CSP* every instance of CSP whose arity of constraints is two. For binary CSPs (every constraint involves a pair of variables), the mathematical object corresponding to the constraint network is a graph (X, C), whose vertices and edges are labeled respectively by the domains and the relations; it is called the *constraint graph*. For n-ary CSPs (the constraints have any arity), the mathematical object is an hypergraph, the *constraint hypergraph*. In this paper, we restrict the study to binary CSPs, without loss of generality, in order to simplify the explanations.

2.2. Tree-decomposition of CSPs

The significant works about CSPs can be divided in three trends, which are not incompatible: the techniques of simplification by filtering, the optimization of backtracking algorithms, and the decomposition methods based on the exploitation of polynomial classes.

The basic method of resolution, generally called *Chronological Backtracking*, assigns to each variable a value of its domain, by checking the consistency of the current instantiation—compatibility of the new assignment with the previous ones—and by going back as far as possible in the search tree if an inconsistency occurs, or by extending it otherwise. This approach leads to a combinatorial explosion. Its worst-case time complexity is $O(m.d^n)$ while its space complexity can be bounded to $O(n)$. In order

to lessen the impact of the theoretical and practical inefficiency of such an approach, many different techniques were developed. For example, simplify the treated instance by filtering, before or during the resolution. Either, analyze the reasons of failures in order to prevent these failures reproducing (constraint learning [12], nogood recording [34]) as well as jumping back as far as possible in the search tree (backjumping [17], dependency directed backtracking [33], conflict-directed backjumping [28], Dynamic Backtracking [18]). Jointly, many heuristics were proposed with a view to guide the algorithms for the choices of variables and values to assign first. To date, there is neither algorithm, nor heuristic which are always better than other ones, because the particular features of instances can favour one method or another one. Note that if we consider static variables (and/or values) ordering, a formal comparison between backtracking algorithms can be partially established (see [23]). [11] partly extends these results to dynamic orderings.

The only guarantee which can exist in terms of theoretical complexity before solving a problem are offered by decomposition methods. They proceed by isolating the parts intrinsically exponential—that is to say untractable in polynomial theoretical time—to induce a second step which guarantees a polynomial time of resolution. These methods generally exploit topological properties of the constraint graph and are based on the notion of tree-decomposition of graphs [29], as defined below.

Definition 1 (*tree-decomposition* [29]). Let $G = (X, E)$ be a graph.

A tree-decomposition of G is a pair $(\mathcal{C}, \mathcal{T})$ with $\mathcal{T} = (I, F)$ a tree and $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ a family of subsets of X, such that each $\mathcal{C}_i$ is a node of $\mathcal{T}$ and verifies:

(1) $\bigcup_{i \in I} \mathcal{C}_i = X$,
(2) for all edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subset \mathcal{C}_i$, and
(3) for all $i, j, k \in I$, if k is in a path from i to j in $\mathcal{T}$, then $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$.

The width of a tree-decomposition $(\mathcal{C}, \mathcal{T})$ is equal to $\max_{i \in I} |\mathcal{C}_i| - 1$. The tree-width of the graph G is the minimal width over all the tree-decompositions of G.

Note that for the reader who is not familiar with these notions, the definition of a tree $\mathcal{T} = (I, F)$ refers to a set of edges F which is required to satisfy the part (3) of Definition 1. Even if the complexity of the problem of finding tree-decomposition is NP-Hard [1], many works have been developed in this direction [3], which often exploit equivalent definitions of this notion, including one based on an algorithmic approach related to *triangulated* graphs (see [20] for an introduction to triangulated graphs). The link between triangulated graphs and tree-decomposition is obvious. Indeed, given a triangulated graph, the set of maximal cliques $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ of (X, E) corresponds to the family of subsets associated with a tree-decomposition. As any graph $G = (X, E)$ is not necessarily triangulated, a tree-decomposition can be approximated by triangulating G. We call *triangulation* the addition to G of a set E' of edges such that $G' = (X, E \cup E')$ has no cycle of length at least 4 without a chord (i.e., an edge joining two non-consecutive vertices in the cycle). The width of a triangulation G' of graph G is equal to the maximal size of cliques minus one in the resulting graph G'. The tree-width of G is then equal to the minimal width over all triangulations.

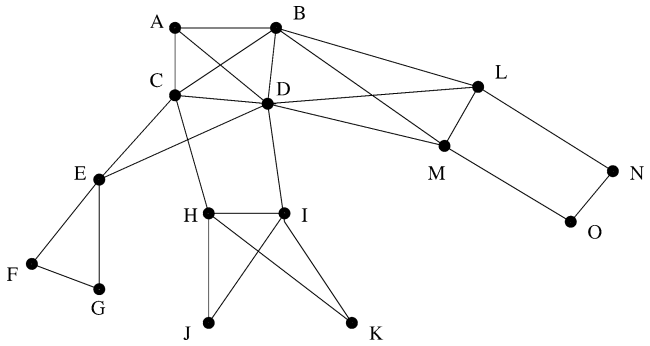


Fig. 1. A constraint graph on 15 variables.

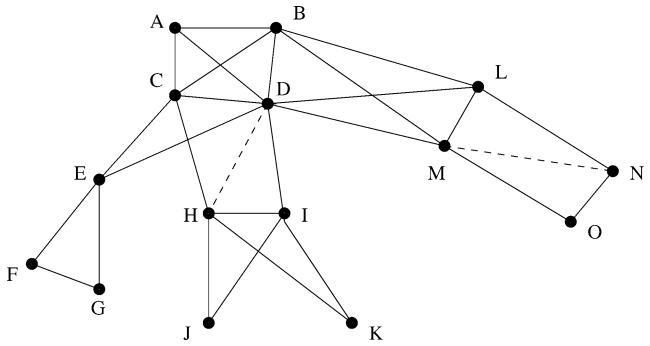


Fig. 2. The constraint graph given in Fig. 1 after its triangulation (dashed lines).

The graph in Fig. 1 is not triangulated. In Fig. 2, a possible triangulation of this graph is provided where the maximum size of cliques is four (see Fig. 3). This is an optimal triangulation, so, the tree-width of this graph is three. In Fig. 4, a tree whose nodes correspond to maximal cliques of the triangulated graph is a possible tree-decomposition for the graph of Fig. 1. So, we get $\mathcal{C}_1 = \{A, B, C, D\}$, $\mathcal{C}_2 = \{C, D, E\}$, $\mathcal{C}_3 = \{E, F, G\}$, $\mathcal{C}_4 = \{C, D, H\}$, $\mathcal{C}_5 = \{D, H, I\}$, $\mathcal{C}_6 = \{H, I, J\}$, $\mathcal{C}_7 = \{H, J, K\}$, $\mathcal{C}_8 = \{B, D, L, M\}$, $\mathcal{C}_9 = \{L, M, N\}$ and $\mathcal{C}_{10} = \{M, N, O\}$.

The CSP decomposition method called *Tree-Clustering*, proposed by Dechter and Pearl [15] is based on these notions (see also [13] for a more recent description); it proceeds by four steps:

1. Triangulate the constraint graph.
2. Find maximal cliques (each clique corresponds to a subproblem).
3. Solve every subproblem induced by the maximal cliques.
4. Solve the new acyclic n-ary CSP.

The guiding idea of this method is to provide a systematic scheme, which, from any CSP, produces an equivalent n-ary CSP by a covering of the set of constraints in order to

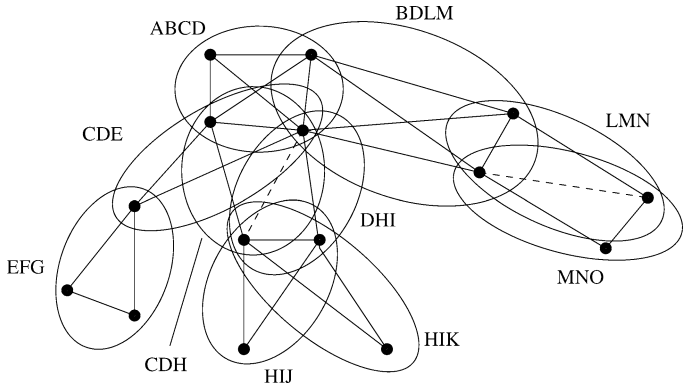


Fig. 3. The acyclic hypergraph induced by maximal cliques of the triangulated graph given in Fig. 2.

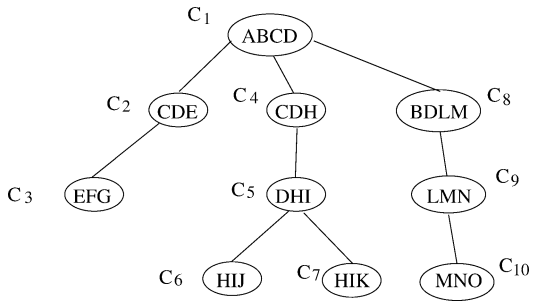


Fig. 4. The tree-decomposition of the triangulated constraint graph given in Fig. 2.

build an acyclic constraint hypergraph. Such a CSP can be solved in polynomial time with respect to the size of the induced n-ary CSP.

This method is generally presented [15] using an approximation of the optimal triangulation (some comments about triangulations are given in Section 5). Phases 1 and 2 are feasible in polynomial time, more precisely, in $O(n + m')$ with m' the number of edges of the graph after the triangulation ($m \leqslant m' < n^2$). Moreover, note that the tree associated to the acyclic hypergraph can be computed in linear time, given the maximal cliques. Step 3 is feasible in $O(m.d^{w^+ + 1})$ with w^+ the size minus one of the biggest produced clique ($w^+ + 1 \leqslant n$). The last step has the same complexity. The space complexity, which is bound to the storage of solutions of subproblems, can be reduced to $O(n.s.d^s)$ with s the maximal size of minimal separators, which equals the size of the biggest intersection between subproblems ($s \leqslant w^+$). Finally, note that for every decomposition which induces a value w^+, we have $w \leqslant w^+$ with w the tree-width of the initial constraint graph.

Figs. 1–3 can be considered as an illustration of this method. In Fig. 1, we see a constraint graph. After step 1, the triangulation adds two edges (the dashed lines). A covering of this graph by maximal cliques defines an acyclic hypergraph. Each maximal clique defines a subproblem.

Although theoretically interesting, all the practical interest of this method is not proved yet, even if it is clear that, for some classes of CSP, it can provide an useful approach [13]. One reason of the lack of efficiency of Tree-Clustering is due to the heaviness of the approach, and specially the required space. In the case where all the solutions are searched, it may be useful. In the other hand, if we check the consistency or if we search only one solution, we will prefer to use an enumerative algorithm such as Forward Checking (denoted FC [21], Real Full Look-Ahead (denoted RFLA [27]) or Maintaining Arc-Consistency (denoted MAC [31]), due to the space costs of Tree-Clustering, and to its practical efficiency.

In the next section, we show how the reference to such a structural decomposition allows to establish a search procedure based on enumeration while keeping the complexity bounds given above.

3. The BTD method

3.1. Presentation

The BTD method (for Backtracking with Tree-Decomposition) proceeds by an enumerative search guided by a static pre-established partial order induced by a tree-decomposition of the constraint-network. So, the first step of BTD consists in computing a tree-decomposition or an approximation of a tree-decomposition. The obtained partial order allows to exploit some structural properties of the graph, during the search, in order to prune some branches of the search tree. Hence, what distinguishes BTD from other techniques concerns the following points:

- the variable instantiation order is induced by a tree-decomposition of the constraint graph,
- some parts of the search space would not be visited again as soon as their consistency is known (notion of *structural good*),
- some parts of the search space would not be visited again if it is known that the current instantiation leads to a failure (notion of *structural nogood*).

Note that this method is called BTD for Backtracking with Tree-Decomposition, but we will see latter that the enumerative search can be implemented with the basic Backtracking, or FC, or MAC (and more sophisticated algorithms).

3.2. Theoretical foundations

Let $\mathcal{P} = (X, D, C, R)$ be an instance where (X, C) is a graph, with $\mathcal{A} = (\mathcal{C}, \mathcal{T})$ a tree-decomposition (or an approximation) where $\mathcal{T} = (I, F)$ is a tree. We suppose that the elements of $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ are indexed with respect to the notion of *compatible numeration*:

Definition 2. A numeration on $\mathcal{C}$ compatible with a prefix numeration of $\mathcal{T} = (I, F)$ with $\mathcal{C}_1$ the root is called *compatible numeration* $N_{\mathcal{C}}$.

Note that the example of tree-decomposition given in Fig. 4 is a compatible numeration on $\mathcal{C}$. We note $Desc(\mathcal{C}_j)$ the set of variables belonging to the union of the descendants $\mathcal{C}_k$ of $\mathcal{C}_j$ in the tree rooted in $\mathcal{C}_j$, $\mathcal{C}_j$ included. For example, $Desc(\mathcal{C}_4) = \mathcal{C}_4 \cup \mathcal{C}_5 \cup \mathcal{C}_6 \cup \mathcal{C}_7 = \{C, D, H, I, J, K\}$. Note that the numeration $N_{\mathcal{C}}$ defines a partial variable ordering that permits to get an enumeration order of the variables of $\mathcal{P}$:

Definition 3. An order $\preceq_X$ of variables of X such that $\forall x \in \mathcal{C}_i,\ \forall y \in \mathcal{C}_j$, with $i < j$, $x \preceq_X y$ is a compatible enumeration order.

For example, the alphabetical order $A, B, \ldots, N, O$ is a compatible enumeration order. The tree-decomposition with the numeration N_C permits to clarify some relations in the constraint graph.

Theorem 1. *Let $\mathcal{C}_j$ be a son of $\mathcal{C}_i$ (so $i < j$). There does not exist an edge $\{x, y\}$ in the graph (X, C) where $x \in (\bigcup_{k=1}^{j-1} \mathcal{C}_k) \backslash (\mathcal{C}_i \cap \mathcal{C}_j)$ and $y \in Desc(\mathcal{C}_j) \backslash (\mathcal{C}_i \cap \mathcal{C}_j)$.*

Proof. By construction, $\mathcal{C}_i \cap \mathcal{C}_j$ is clearly a separator of the graph which disconnects $(\bigcup_{k=1}^{j-1} \mathcal{C}_k) \backslash (\mathcal{C}_i \cap \mathcal{C}_j)$ and $Desc(\mathcal{C}_j) \backslash (\mathcal{C}_i \cap \mathcal{C}_j)$. □

For example, let $i = 1$, $j = 4$, and $\mathcal{C}_4$ be a son of $\mathcal{C}_1$. There is no edge in G between $(\mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3) \backslash (\mathcal{C}_1 \cap \mathcal{C}_4) = \{A, B, C, D, E, F, G\} \backslash \{C, D\} = \{A, B, E, F, G\}$ and $Desc(\mathcal{C}_4) \backslash (\mathcal{C}_1 \cap \mathcal{C}_4) = \{C, D, H, I, J, K\} \backslash \{C, D\} = \{H, I, J, K\}$.

In terms of CSP, there is no constraint joining these two subsets of variables and therefore these two subproblems. Consequently, the compatibility relations between instantiations pass only through the separator $\mathcal{C}_i \cap \mathcal{C}_j$.

The BTD method is based on compatible enumeration order and this first theorem. Let us consider a consistent instantiation $\mathcal{A}$ of variables of $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_i \cup \mathcal{C}_{i+1} \cup \cdots \cup \mathcal{C}_{j-1}$, with $\mathcal{C}_j$ a son of $\mathcal{C}_i$. Due to the definition of compatible orders, the enumeration continues with the variables of the lineage $Desc(\mathcal{C}_j)$ except ones which belong to $\mathcal{C}_i \cap \mathcal{C}_j$. Then two cases arise depending on whether a consistent extension of the current instantiation on $Desc(\mathcal{C}_j)$ exists or not:

- *There is no consistent extension*. In such a case, the reason of the inconsistency can only be the unsatisfaction of some constraints which join two variables of $Desc(\mathcal{C}_j)$ or (not exclusive or) a variable of this set and a variable which precedes it in the order, so which belongs to $\mathcal{C}_i \cap \mathcal{C}_j$ (see Theorem 1). In both case, if a new consistent assignment $\mathcal{A}'$ such that $\mathcal{A}'$ and $\mathcal{A}$ are equal on $\mathcal{C}_i \cap \mathcal{C}_j$ is tried, its extension on $Desc(\mathcal{C}_j)$ will lead to the same failure, independently of what precedes. In fact, the instantiation restricted to $\mathcal{C}_i \cap \mathcal{C}_j$ may be considered as a *nogood* in the usual sense of the term, although, here, it is found by structural criteria. This nogood can be recorded and exploited during next searches.

- *There exists a consistent extension*. By a similar reasoning to previous one, we can prove that every instantiation which is the same on $\mathcal{C}_i \cap \mathcal{C}_j$ will lead to a success on $Desc(\mathcal{C}_j)$, because it is independent of what precedes. This assignment can be now considered as a *good* in the sense that on a part of the problem, $Desc(\mathcal{C}_j)$, this assignment has a consistent extension. Like nogoods, goods may be recorded and used during further searches, allowing to jump in the search tree (*forward-jumping*), what leads to continue the enumeration with the variables located after ones of $Desc(\mathcal{C}_j)$ in the compatible enumeration order.

The closest works of our approach are ones of Bayardo and Miranker in [4] whose study is limited to the resolution of binary CSPs whose constraint graph is a tree. Our approach can be considered as a generalization of their work since their goods and nogoods instantiate variables while our goods and nogoods instantiate sets of variables (separators). In [5], Bayardo and Miranker propose another generalization of goods and nogoods which is not based on separators but on sets of ancestors in an ordered constraint graph. Formally, their work is different though their use of goods and nogoods during search is similar to ours (see Section 6 for more details).

Now, we formally introduce goods and nogoods based on separators.

Definition 4. Given $\mathcal{C}_i$ and $\mathcal{C}_j$ one of its sons, a **good** (respectively **nogood**) of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$, noted $g(\mathcal{C}_i/\mathcal{C}_j)$ (respectively $ng(\mathcal{C}_i/\mathcal{C}_j)$), is a consistent assignment $\mathcal{A}$ of $\mathcal{C}_i \cap \mathcal{C}_j$ such that there exists (respectively does not exist) a consistent extension of $\mathcal{A}$ on $Desc(\mathcal{C}_j)$.

The following Lemma 1 and its corollary show that the interactions between a subproblem rooted in $\mathcal{C}_j$ and the remaining of the CSP pass through the intersection between $\mathcal{C}_j$ and its father $\mathcal{C}_i$. These properties are at the origin of the cuttings (for the nogoods) and the jumps (for the goods) which will be realized in the tree search.

Lemma 1. *Given $\mathcal{C}_i$ and $\mathcal{C}_j$ one of its sons, given $Y \subset X$ such that $Desc(\mathcal{C}_j) \cap Y = \mathcal{C}_i \cap \mathcal{C}_j$, every consistent instantiation $\mathcal{B}$ of $Desc(\mathcal{C}_j)$ is compatible with every consistent instantiation $\mathcal{A}$ of Y iff $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j]$.*

Proof. According to Theorem 1 and by construction, the only constraints joining the variables of Y to the variables of $Desc(\mathcal{C}_j)$ are the constraints which involve the variables common to $Desc(\mathcal{C}_j)$ and to Y, i.e., $\mathcal{C}_i \cap \mathcal{C}_j$. It results that $\mathcal{A}$ and $\mathcal{B}$ are compatible iff each common variable has the same value in $\mathcal{A}$ and $\mathcal{B}$ (i.e., $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j]$). □

It ensues the following corollary:

Corollary 1. *Given $\mathcal{C}_i$ and $\mathcal{C}_j$ one of its sons, every consistent instantiation $\mathcal{B}$ of $Desc(\mathcal{C}_j)$ is compatible with every consistent instantiation $\mathcal{A}$ of $(X \setminus Desc(\mathcal{C}_j)) \cup \mathcal{C}_i$ iff $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j]$.*

We then formalize the exploitation of goods:

Lemma 2 (jump by the goods). *Given $\mathcal{C}_i$ and $\mathcal{C}_j$ one of its sons, given $Y \subset X$ such that $Desc(\mathcal{C}_j) \cap Y = \mathcal{C}_i \cap \mathcal{C}_j$, for all $g(\mathcal{C}_i/\mathcal{C}_j)$, every consistent instantiation $\mathcal{A}$ of Y such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = g(\mathcal{C}_i/\mathcal{C}_j)$ has a consistent extension on $Desc(\mathcal{C}_j)$.*

Proof. Let $\mathcal{A}$ be a consistent instantiation such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = g(\mathcal{C}_i/\mathcal{C}_j)$. According to the definition of goods, there exists an instantiation $\mathcal{B}$ on $Desc(\mathcal{C}_j)$ such that $\mathcal{B}$ is consistent and $\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j] = g(\mathcal{C}_i/\mathcal{C}_j)$. As $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = g(\mathcal{C}_i/\mathcal{C}_j) = \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j]$, $\mathcal{A}$ and $\mathcal{B}$ are compatible (according to Lemma 1). Therefore, $\mathcal{B}$ is a consistent extension of $\mathcal{A}$ on $Desc(\mathcal{C}_j)$. □

Thus, if a partial instantiation $\mathcal{A}$ is such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is a good of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$, then it is not necessary to extend the search on $Desc(\mathcal{C}_j)$. So the enumeration goes on with the variables of the first $\mathcal{C}_k$ located out of $Desc(\mathcal{C}_j)$, for instance the next brother of $\mathcal{C}_j$, if there exists one.

Lemma 3 (cutting by the nogoods). *Given $\mathcal{C}_i$ and $\mathcal{C}_j$ one of its sons, given $Y \subset X$ such that $Desc(\mathcal{C}_j) \cap Y = \mathcal{C}_i \cap \mathcal{C}_j$, for all $ng(\mathcal{C}_i/\mathcal{C}_j)$, there is no assignment $\mathcal{A}$ of Y such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = ng(\mathcal{C}_i/\mathcal{C}_j)$ and such that $\mathcal{A}$ has a consistent extension on $Desc(\mathcal{C}_j)$.*

Proof. According to the definition of a nogood, there is no extension of $ng(\mathcal{C}_i/\mathcal{C}_j)$ on $Desc(\mathcal{C}_j)$. As $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = ng(\mathcal{C}_i/\mathcal{C}_j)$, $\mathcal{A}$ cannot be extended on $Desc(\mathcal{C}_j)$. □

3.3. The basic algorithm

The method obtained from these notions can be implemented in several ways according to whether a filtering is associated or not with the enumeration. However, the mechanisms will be similar. The BTD method explores the search space by using a compatible order $\preceq_X$, which begins with the variables of $\mathcal{C}_1$. Inside $\mathcal{C}_i$, the enumeration works in classical way. On the other hand, when all the variables are assigned by satisfying all the involved constraints, we then get a consistent instantiation $\mathcal{A}$ of variables of $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_i$. The search must go on with the variables of the first son $\mathcal{C}_{i+1}$ of $\mathcal{C}_i$ if there exists one. More generally, let us consider the case of one son $\mathcal{C}_j$ of $\mathcal{C}_i$. We check if $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is a good or a nogood and we take appropriate action:

- In the case of a nogood, we change the current instantiation on $\mathcal{C}_i$.
- In the case of a good, a "forward-jump" happens in order to continue the enumeration with the first variable located after those of $Desc(\mathcal{C}_j)$. Fig. 5 illustrates the case of a forward-jump, assuming that $\mathcal{A}[\mathcal{C}_4 \cap \mathcal{C}_5] = \mathcal{A}[\{D, H\}]$ is a good. We show in part (a) the jump in a compatible enumeration order, and in part (b), where the search goes on in the structure of the instance.
- In the other cases, i.e., $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is neither a good nor a nogood, $\mathcal{A}$ must be extended in consistent way on the variables of $Desc(\mathcal{C}_j)$. If so, $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is recorded as a good; on the contrary, if $\mathcal{A}$ cannot be extended in consistent way, the nogood $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is recorded.

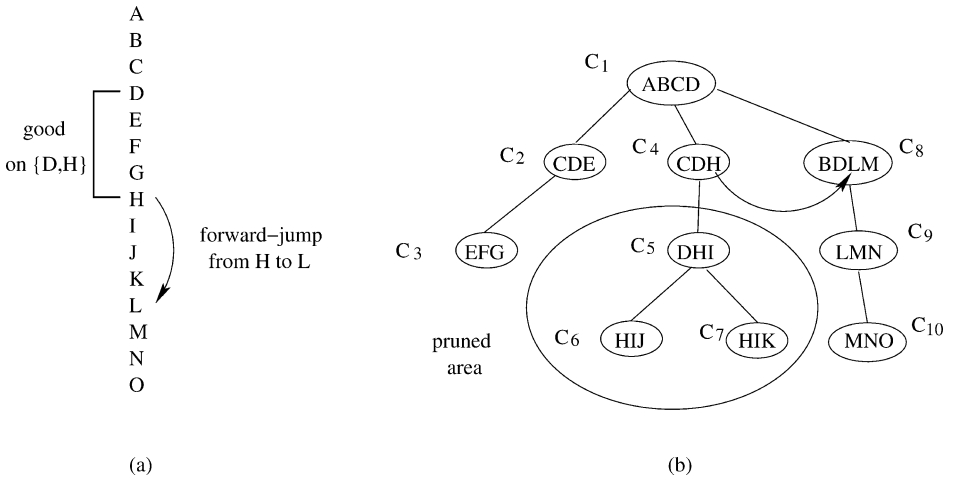


Fig. 5. Example of a forward-jump with a good $\mathcal{A}[\mathcal{C}_4 \cap \mathcal{C}_5]$ on $\{D, H\}$. In (a), we show the jump along the enumeration order, while in (b) we see the jump in the structure of the problem.

Fig. 6 describes the BTD algorithm restricted to the consistency check: it returns **True** if the consistent instantiation $\mathcal{A}$ can be extended to a consistent instantiation on $V_{\mathcal{C}_i}$ and on all the descents of $\mathcal{C}_i$; **False** otherwise. $V_{\mathcal{C}_i}$ represents the set of unassigned variables of $\mathcal{C}_i$ and G and N respectively the set of recorded goods and of nogoods. This algorithm is run after having computed a tree-decomposition (or an approximation) of the constraint graph.

Theorem 2. *BTD is sound, complete and terminates.*

Proof. This algorithm is proved by induction, exploiting properties of structural goods and nogoods. The induction is made on the number of variables appearing in the lineage of $\mathcal{C}_i$ except the already assigned variables of $\mathcal{C}_i$. This set of variables is denoted

$$VAR(\mathcal{C}_i, V_{\mathcal{C}_i}) = V_{\mathcal{C}_i} \cup \left(\bigcup_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \big(Desc(\mathcal{C}_j) \backslash (\mathcal{C}_i \cap \mathcal{C}_j)\big) \right).$$

$VAR(\mathcal{C}_i, V_{\mathcal{C}_i})$ is then the set of variables to assign to know whether $\mathcal{A}$ can be extended to a consistent assignment on $V_{\mathcal{C}_i}$ and its lineage.

To prove BTD, we must prove the property $P(\mathcal{A}, VAR(\mathcal{C}_i, V_{\mathcal{C}_i}))$ defined as:

"BTD$(\mathcal{A}, \mathcal{C}_i, V_{\mathcal{C}_i})$ returns true if the consistent assignment $\mathcal{A}$ can be extended to a consistent assignment on $V_{\mathcal{C}_i}$ and the lineage of $\mathcal{C}_i$; otherwise, BTD returns false".

Consider $P(\mathcal{A}, \emptyset)$:

If $VAR(\mathcal{C}_i, V_{\mathcal{C}_i}) = \emptyset$, then $V_{\mathcal{C}_i} = \emptyset$ and $Sons(\mathcal{C}_i) = \emptyset$. Since $\mathcal{A}$ is a consistent assignment, $\mathcal{A}$ can be extended to a consistent assignment on $V_{\mathcal{C}_i}$ and on the lineage of $\mathcal{C}_i$. Therefore $P(\mathcal{C}_i, VAR(\mathcal{C}_i, V_{\mathcal{C}_i}))$ is true.

Induction step: $P(\mathcal{A}, S)$ with $S \neq \emptyset$. Suppose that $\forall S' \subset S$, $P(\mathcal{A}, S')$ holds.

```
1.  BTD(A, C_i, V_{C_i})
2.  If V_{C_i} = ∅
3.  Then
4.     If Sons(C_i) = ∅ Then Return True
5.     Else
6.        Consistency ← True
7.        F ← Sons(C_i)
8.        While F ≠ ∅ and Consistency Do
9.           Choose C_j in F
10          F ← F\{C_j}
11.         If A[C_j ∩ C_i] is a good of C_i/C_j in G Then Consistency ← True
12.         Else
13.            If A[C_j ∩ C_i] is a nogood of C_i/C_j in N Then Consistency ← False
14.            Else
15.               Consistency ← BTD(A, C_j, C_j\(C_j ∩ C_i))
16.               If Consistency
17.               Then Record the good A[C_j ∩ C_i] of C_i/C_j in G
18.               Else Record the nogood A[C_j ∩ C_i] of C_i/C_j in N
19.            EndIf
20.         EndIf
21.      EndWhile
22.      Return Consistency
23.   EndIf
24. Else
25.   Choose x ∈ V_{C_i}
26.   d_x ← D_x
27.   Consistency ← False
28.   While d_x ≠ ∅ and ¬Consistency Do
29.      Choose v in d_x
30.      d_x ← d_x\{v}
31.      If ∄c ∈ C such that c is not satisfied by A ∪ {x ← v}
32.      Then Consistency ← BTD(A ∪ {x ← v}, C_i, V_{C_i}\{x})
33.      EndIf
34.   EndWhile
35.   Return Consistency
36. EndIf
```

Fig. 6. The BTD algorithm.

– If $V_{\mathcal{C}_i} \neq \emptyset$:
During the **While** loop (lines 28–34) the assertion: "there is no value v of x already checked such that $\mathcal{A}$ extended by that value leads to a consistent assignment for $V_{\mathcal{C}_i}$ and the lineage of $\mathcal{C}_i$" is true.
If BTD is called (line 32), $\mathcal{A} \cup \{x \leftarrow v\}$ is then consistent (since no constraint is violated) and $VAR(\mathcal{C}_i, V_{\mathcal{C}_i \setminus \{x\}}) \subset VAR(\mathcal{C}_i, V_{\mathcal{C}_i})$. According to the induction hypothesis, the assignment $\mathcal{A}$ has been extended if $BTD(\mathcal{A} \cup \{x \leftarrow v\}, \mathcal{C}_i, V_{\mathcal{C}_i} \setminus \{x\})$ is true. In that case, $BTD(\mathcal{A}, \mathcal{C}_i, V_{\mathcal{C}_i})$ returns true and $P(\mathcal{A}, VAR(\mathcal{C}_i, V_{\mathcal{C}_i}))$ is satisfied.

After the loop (line 35), all the possible values have been tried without consistent extension of $\mathcal{A}$. Therefore, $BTD(\mathcal{A}, \mathcal{C}_i, V_{\mathcal{C}_i})$ returns false and $P(\mathcal{A}, VAR(\mathcal{C}_i, V_{\mathcal{C}_i}))$ is satisfied.

– If $V_{\mathcal{C}_i} = \emptyset$:
During the **While** loop (lines 8–21) the assertion: "for each son $\mathcal{C}_f$ already checked, $\mathcal{A}$ can be extended to a consistent assignment on $Desc(\mathcal{C}_f)$" holds.
We show that this assertion is true at the end of the loop.
Let $\mathcal{C}_j$ be a son of $\mathcal{C}_i$ to be examined.
  + If $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$ is a good of $\mathcal{C}_i/\mathcal{C}_j$, by Lemma 2, we know that $\mathcal{A}$ can be extended on $Desc(\mathcal{C}_j)$. Therefore, the assertion is true at the end of the loop.
  + If $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$ is a nogood of $\mathcal{C}_i/\mathcal{C}_j$, by Lemma 3, we know that $\mathcal{A}$ cannot be extended on $Desc(\mathcal{C}_j)$. The loop is then finished.
  + If $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$ is neither a good, nor a nogood, then, BTD is called with $\mathcal{A}$ which is a consistent assignment and $VAR(\mathcal{C}_j, \mathcal{C}_j \setminus (\mathcal{C}_j \cap \mathcal{C}_i)) \subset VAR(\mathcal{C}_i, \emptyset)$. So, according to the induction hypothesis, $\text{BTD}(\mathcal{A}, \mathcal{C}_j, \mathcal{C}_j \setminus (\mathcal{C}_j \cap \mathcal{C}_i))$ returns true if $\mathcal{A}$ admits a consistent assignment on $Desc(\mathcal{C}_j)$, and then the assertion is verified. Otherwise, the loop is stopped.
  After the loop (line 22), $\text{BTD}(\mathcal{A}, \mathcal{C}_i, \emptyset)$ returns true if $\mathcal{A}$ has been consistently extended on every son, and returns false otherwise.
  Therefore, $P(\mathcal{A}, VAR(\mathcal{C}_i, V_{\mathcal{C}_i}))$ is satisfied. Note that the memorization of goods and nogoods is justified by their definition.

To summarize, since BTD satisfies $P(\mathcal{A}, VAR(\mathcal{C}_i, V_{\mathcal{C}_i}))$, in particular BTD satisfies the property $P(\emptyset, VAR(\mathcal{C}_1, \mathcal{C}_1))$ for the first call, and then BTD is sound, complete and terminates. □

3.4. Extensions of BTD

We now discuss extensions of the BTD algorithm presented in the previous section. It is based on Chronological Backtracking. It is well known that this algorithm is not efficient in practice. So, its natural extensions which generally exploit lookahead techniques like arc-consistency or forward-checking must be integrated to the BTD approach.

Thus, we introduce two extensions based on filterings:

- **FC-BTD** which is BTD using the classical filtering used in Forward-Checking [21].
- **MAC-BTD** which is BTD using an arc-consistency filtering [31].

These extensions are straightforward if the used filtering does not modify the structure of the constraint network. Indeed, a more powerful filtering like path-consistency [26] [25] applied during search is not possible because new edges can be added to the constraint network, modifying its structural properties with consequences on the properties of BTD. So that for FC-BTD, the correctness of the extension is trivial, for MAC-BTD this extension is straightforward but we consider it must be established by the next property:

Theorem 3. *Let $\mathcal{C}_j$ be a son of $\mathcal{C}_i$ and let $\mathcal{A}$ be a consistent assignment on $\bigcup_{k=1}^{j-1} \mathcal{C}_k$. Assume that the arc-consistent closure of the CSP $\mathcal{P}$ after the assignment $\mathcal{A}$ (denoted $AC(\mathcal{P}, \mathcal{A})$) has no empty domains. If g is a good of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$ in $\mathcal{P}$ such that $g = \mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$, then g is a good in $AC(\mathcal{P}, \mathcal{A})$.*

Proof. Let $\mathcal{B}$ be a consistent assignment on $Desc(\mathcal{C}_i)$ associated to the good g. That is $\mathcal{B}$ is a solution of the subproblem of $\mathcal{P}$ induced by the variables occurring in $Desc(\mathcal{C}_i)$. Therefore, we get $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j]$. By definition, $\mathcal{B}$ satisfies all the constraints belonging to $Desc(\mathcal{C}_j)$. Moreover, all the values in $\mathcal{A}$ are compatible with all the values in $\mathcal{B}$. Indeed, the constraints between $\mathcal{A}$ and $\mathcal{B}$ associate pairs of variables $\{x_i, x_j\}$ such that $x_i \in \mathcal{C}_i$ and $x_j \in \mathcal{C}_j$. Then, three cases exist:

(1) $x_j \in \mathcal{C}_i$. Therefore, since $\mathcal{A}$ is a consistent assignment, $\mathcal{A}$ satisfies the constraints occurring in $\mathcal{C}_i$ especially $\{x_i, x_j\}$.
(2) $x_i \in \mathcal{C}_j$. Therefore, since $\mathcal{B}$ is a consistent assignment, $\mathcal{B}$ satisfies the constraints occurring in $\mathcal{C}_j$ especially $\{x_i, x_j\}$.
(3) $x_i, x_j \in \mathcal{C}_i \cap \mathcal{C}_j$ which is a particular case of the upper cases.

Therefore, the assignment defined by the assignment $\mathcal{A}$ extended by $\mathcal{B}$, that is $\mathcal{A} \cup \mathcal{B}$, is a solution of the subproblem defined by the variables appearing in $\mathcal{A}$ or in $Desc(\mathcal{C}_j)$ since all the constraints are satisfied. Thus, $\mathcal{A} \cup \mathcal{B}$ is a consistent assignment, and then the values in $\mathcal{B}$ necessarily appear in $AC(\mathcal{P}, \mathcal{A})$. □

Another way to improve backtracking search consists in using a non-chronological backtracking like *Backjumping* classically denoted **BJ** [17]. Backjumping allows us to define three immediate extensions of BTD:

- **BTD-BJ** which is BTD using Backjumping.
- **FC-BTD-BJ** which is BTD using the classical filtering used in Forward-Checking and Backjumping.
- **MAC-BTD-BJ** which is BTD using an arc-consistency filtering and Backjumping.

BTD-BJ (respectively FC-BTD-BJ and MAC-BTD-BJ) is similar to BTD (respectively FC-BTD and MAC-BTD) with an additional phase of backjump. This phase of backjump is achieved when BTD comes back to the cluster $\mathcal{C}_i$ after a failure during the search for an extension of the current instantiation over the descent of $\mathcal{C}_i$ rooted in a son $\mathcal{C}_j$ of $\mathcal{C}_i$. It consists in coming back to the deepest variable which belongs both to $\mathcal{C}_i$ and $\mathcal{C}_j$.

Finally, note that the BTD algorithm only builds a consistent instantiation which can be extended to a solution of the treated instance, if one exists. Indeed, some variables are unassigned due to the jumps realized thanks to goods. Nonetheless, it is easy to extend the produced assignment to a solution of the problem by using a backtracking search and by checking the recorded goods and nogoods as new constraints. Note that this extension does not change anything to the complexity bounds provided in the next section.

4. Time and space complexities

In this section, we first assess the time and space complexities of the BTD algorithm. Then, we compare BTD with the Chronological Backtracking and the Tree-Clustering. Note that these results also hold if we consider more sophisticated backtracking search as FC or MAC. Let us assume that a tree-decomposition or its approximation has been computed.

We begin by evaluating the space complexity of BTD:

Theorem 4. *BTD has a space complexity in* $O(n.s.d^s)$ *where* s *is the size of the largest intersection* $\mathcal{C}_i \cap \mathcal{C}_j$ *with* $\mathcal{C}_j$ *son of* $\mathcal{C}_i$.

Proof. BTD only records the goods and the nogoods. Goods and nogoods are instantiations on the intersections $\mathcal{C}_i \cap \mathcal{C}_j$ with $\mathcal{C}_j$ son of $\mathcal{C}_i$. Therefore, if s is the size of the largest of these intersections, BTD has a space complexity in $O(n.s.d^s)$ because the number of intersections $\mathcal{C}_i \cap \mathcal{C}_j$ is bounded by n while the number of goods and nogoods associated to one intersection is bounded by d^s and the size of a good or a nogood is at most s. □

Next, we calculate the time complexity of BTD.

Theorem 5. *BTD has a time complexity in* $O(n.s^2.m.\log(d^s).d^{w^+ +1})$ *with* $w^+ + 1$ *the size of the largest* $\mathcal{C}_i$ *and* s *the size of the largest intersection* $\mathcal{C}_i \cap \mathcal{C}_j$ *with* $\mathcal{C}_j$ *son of* $\mathcal{C}_i$.

Proof. Assume that we want to extend an instantiation on $\mathcal{C}_j$. There exist two cases:

- Either $\mathcal{C}_j = \mathcal{C}_1$, and then find the consistent instantiations on $\mathcal{C}_j$ has a worst-case time complexity in $O(m.d^{|\mathcal{C}_j|})$. Note that m is due to the number of constraints to check to ensure consistency.
- Or $\mathcal{C}_j$ is a son of $\mathcal{C}_i$. Let $\mathcal{A}$ be a consistent assignment on Y ($Y \subset X$ such that $Desc(\mathcal{C}_j) \cap Y = \mathcal{C}_i \cap \mathcal{C}_j$).
  Find the consistent extensions of $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$ on $\mathcal{C}_j$ has a worst-case time complexity in $O(m.d^{|\mathcal{C}_j \setminus \mathcal{C}_j \cap \mathcal{C}_i|})$.
  BTD searches the extension of $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$ once and only once (thanks to recorded goods and nogoods). As there exist at most $d^{|\mathcal{C}_i \cap \mathcal{C}_j|}$ assignments $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$, the worst-case time complexity of finding the extension on $\mathcal{C}_j$ is in $O(d^{|\mathcal{C}_j|})$.

Therefore, if $w^+ + 1$ is the size of the largest $\mathcal{C}_i$, the search of an extension by BTD has a complexity in $O(n.m.d^{w^+ +1})$, to which must be added the cost of managing and exploiting goods and nogoods. As this cost is zero for $\mathcal{C}_1$, we focus on the case where $\mathcal{C}_j$ is a son of $\mathcal{C}_i$. The comparison between $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ and a recorded good (or nogood) requires $O(|\mathcal{C}_i \cap \mathcal{C}_j|)$ steps. The addition or the search of a good (or a nogood) is in $O(|\mathcal{C}_i \cap \mathcal{C}_j| \log(d^{|\mathcal{C}_i \cap \mathcal{C}_j|}))$. So the management and the exploitation of goods and nogoods have a complexity in

$O(d^{|\mathcal{C}_i|}|\mathcal{C}_i \cap \mathcal{C}_j| \log(d^{|\mathcal{C}_i \cap \mathcal{C}_j|}))$, given $\mathcal{C}_i$ and one of its sons $\mathcal{C}_j$. Therefore, on the overall search, it has a cost in $O(n.s.m.d^{w^+ +1} \log(d^s))$.

Thus, the time complexity of BTD is $O(n.m.d^{w^+ +1} + n.s.m.d^{w^+ +1} \log(d^s))$, i.e., a complexity in $O(n.s^2.m. \log(d^s).d^{w^+ +1})$. □

The time and space complexities of BTD are comparable to ones of Tree-Clustering. We now show that BTD develops fewer nodes (or as many nodes in the worst case) than Chronological Backtracking (denoted BT) and than Tree-Clustering (denoted TC). In order to do these comparisons, we consider that BT and TC use the same variables/values order as BTD and TC must exploit the same tree-decomposition as BTD. Using compatible orders allows to compare easily BT with BTD. Nevertheless, it is clear that a compatible order is not necessarily a good variable order for BT. A more general comparison between BTD and BT (FC and MAC too), requires to study different orders. So, this analysis should be extended in the future to consider different orders. We first compare BTD and BT:

Theorem 6. *Given a compatible order, BTD develops at most as many nodes as BT.*

Proof. Using goods and nogoods permits BTD to avoid some redundancies in the tree search. So BTD develops at most as many nodes as BT. □

Like BT, BTD stops as soon as the problem's consistency is found. In the other hand, TC builds every consistent assignment on $\mathcal{C}_i$, for each $\mathcal{C}_i$. Furthermore, when BTD does not develop a consistent instantiation on $\mathcal{C}_i$, it ensues a saving in number of nodes on all the descent of $\mathcal{C}_i$.

And so, the next theorem shows the gain in nodes of BTD with respect to TC:

Theorem 7. *Given a compatible order, BTD develops at most as many nodes as TC, which uses BT for solving each $\mathcal{C}_i$.*

Proof. BTD and TC develop in the worst case the same number of nodes for $\mathcal{C}_1$. For all other $\mathcal{C}_j$ ($j \neq 1$), TC searches systematically all consistent assignments on $\mathcal{C}_j$, whereas BTD only builds the consistent instantiations on $\mathcal{C}_j$ which are compatible with the current instantiation on $\mathcal{C}_i$, the father of $\mathcal{C}_j$. Thus, BTD develops at most as many nodes as TC. □

Finally, to conclude this section, note that if we put FC or MAC instead of BT, Theorem 6 still holds. Moreover, for time complexity, we get the theoretical complexity time by multiplying the cost by a factor due to the cost of one filtering, in the same spirit as the complexity analysis proposed in [24].

5. Experimental results

The following experiments are carried out with a view to assessing the interest of a method like BTD. The first experiments concern networks whose tree-width is not necessarily small. For them, we hope that BTD is as efficient as any classical enumerative

algorithms. The second experiments work on structured CSPs: we hope that BTD will exploit efficiently topological properties of the network when these properties are related to tree-decomposition, that is CSP with small tree-width. Finally, we assess the behaviour of our method on some real-world instances.

5.1. About implementation

5.1.1. The implemented algorithms

We implement different versions of BTD. The first version, noted FC-BTD, corresponds to a simple implementation of the BTD algorithm based on the Forward-Checking algorithm. The second version, noted FC-BTD-BJ, is FC-BTD with the additional phase of backjump (see Section 3.4 for more details). The last two versions, noted respectively FC-BTD$^-$ and FC-BTD-BJ$^-$, respectively correspond to FC-BTD and FC-BTD-BJ without the recording of the goods and nogoods. We need these versions to assess the contribution of goods and nogoods. In other words, these versions correspond to Forward Checking where the choice of the next variable to instantiate is partly guided by a compatible enumeration order of BTD. Likewise, we define the MAC based versions of BTD.

We implement several algorithms in order to compare them with the different versions of BTD. We use FC [21], Forward-Checking with Conflict-directed BackJumping (denoted FC-CBJ [28]), and MAC [31]. For MAC, arc-consistency is achieved thanks to the AC-2001 algorithm [8], which has an optimal worst-case time complexity.

For the purpose of comparing the number of developed nodes and the space requirements of BTD and Tree-Clustering, we implement a partial version of Tree-Clustering. By partial version, we mean that we only compute all solutions of each cluster. We do not solve the acyclic CSP obtained from the previous computation because this step presents no interest for our comparisons. We note TC-FC our partial implementation of Tree-Clustering based on the Forward-Checking algorithm. Of course, BTD and TC-FC exploit the same tree-decomposition (or the same approximation). Finally, note that we only assess the required memory for TC-FC without recording any partial instantiation because we would need too much space.

5.1.2. Heuristic for choosing the next variable to instantiate

For choosing the next variable to instantiate, all the algorithms in this study use the heuristic *dom*/*deg* [7]. This heuristic is one of the best heuristics for ordering variables. According to this heuristic, the next variable to instantiate is the variable x_i which minimizes the ratio $|D_i|/|\Gamma_i|$ with D_i the current domain of x_i and Γ_i the set of the neighbours of x_i. We select the next variable:

- among all the unassigned variables of the problem for FC, FC-CBJ or MAC,
- among all the unassigned variables of the current cluster for the different versions of BTD.

Note that the different versions of FC-BTD (respectively MAC-BTD) use exactly the same variable ordering.

5.1.3. Approximation of a tree-decomposition by triangulation

As the problem of finding a tree-decomposition is NP-Hard, we only use an approximation of a tree-decomposition by triangulating the constraint graph.

We try several algorithms for triangulating the constraint graph among the *LEX-M* algorithm [30], the *LB-TRIANG* algorithm [2] and the *Fill-in Computation* algorithm [35]. The first two algorithms produce a minimal triangulation (a triangulation E' of a graph $G = (V, E)$ is minimal if there is no triangulation E'' such that $E'' \subset E'$). They have a time complexity in $O(nm)$ with n the number of vertices and m one of edges of the graph, whereas the time complexity of the *Fill-in Computation* algorithm is linear in $O(n + m')$ (m' is the number of edges of the triangulated graph). The experimentations on classical random problems show that the *LEX-M* algorithm provides the best results for BTD. So, for all the following results, we use the *LEX-M* algorithm to compute a triangulation.

From this triangulation, if we compute an approximation of a tree-decomposition, we obtain that cliques and separators have on average a reasonable size, that is to say, the time and the memory needed by BTD are feasible in practice. On the contrary, the largest separator size may be too important, that is to say BTD may request too much memory. So, to prevent this problem, we propose to limit the size of separators by a given parameter $s_{\max}$, like in [13]. This trade-off is made to the detriment of the size of clusters and so of the time. First we compute normally the clique-tree. Then, we traverse the tree in breadth first search. If the son $\mathcal{C}_j$ has an intersection with its parent $\mathcal{C}_i$ whose size is less than $s_{\max}$, the son and its parent remain unchanged. Else, we merge the parent $\mathcal{C}_i$ and its son $\mathcal{C}_j$. The obtained cluster replaces $\mathcal{C}_i$ in the tree (so we call this cluster $\mathcal{C}_i$). Furthermore, the sons of $\mathcal{C}_j$ become the sons of $\mathcal{C}_i$. Finally, note that these modifications do not change the size of the intersection between $\mathcal{C}_i$ and the brothers of $\mathcal{C}_j$.

For the provided results, we limit the separator size to 5. For this size, the separator size is neither too small, nor too large.

5.2. The experimental protocol

The following experimentations are realized on a Linux-based PC with an Intel Pentium III 550 MHz processor and 256 Mb of memory. We set a one hour time limit for determining whether a problem is consistent or not. Beyond one hour, the search is stopped and the problem's consistency is said unknown. The given run-time includes the time of the preliminary treatments (like computing an approximation of a tree-decomposition).

We work on random binary CSPs generated according to two models and on real-world instances.

5.2.1. Classical random CSPs

In order to produce classical random instances, we use the random generator written by D. Frost, C. Bessière, R. Dechter and J.-C. Régin. This generator[1] takes 4 parameters n, d, m and T. It builds a CSP of class (n, d, m, T) with n variables, each having a domain of size d, and m binary constraints ($0 \leqslant m \leqslant n(n-1)/2$) in which T tuples are forbidden

[1] Downloadable at http://www.lirmm.fr/~bessiere/generator.html.

($0 \leqslant T \leqslant d^2$). Among the CSPs produced by this generator, we keep only those whose constraint graph is connected.

The listed results are the averages of results obtained on 100 problems per class. We experiment on random instances with 50 variables and domains of size 15 and whose constraint graph has a density between 10% and 30%. We also test some problems with a larger domain from the class $(50, 25, 123, 439)$. Considered classes are close to the satisfiability's threshold.

5.2.2. Structured random CSPs

We define a new binary CSPs random generator, which produces instances with a structured constraint graph. The constraint graph is triangulated. This property allows us to exactly know the tree-width of the constraint network, and then to know the theoretical complexity bound. This generator takes 5 parameters n, d, $r_{\max}$, T and $s_{\max}$. It builds a binary CSP of the class $(n, d, r_{\max}, T, s_{\max})$ with n variables which have domains of size d and whose constraint graph has the following properties:

- each vertex v belongs at least to a maximal clique with a size greater than 1,
- the cliques have a size at most $r_{\max}$,
- the intersection between two cliques has a size at most $s_{\max}$,
- the cliques form a clique-tree and then the graph is triangulated.

To build such a problem, we first choose a set of $r_{\max}$ variables to form the root clique. Then, while there are remaining variables, we proceed like this:

(1) choose randomly a parent clique $\mathcal{C}_i$,
(2) choose randomly a size of the intersection between $\mathcal{C}_i$ and its son $\mathcal{C}_j$ (the size is bounded by 1 and $s_{\max}$),
(3) choose randomly a size of the clique $\mathcal{C}_j$ (the size is at least 3 and bounded by the size of the intersection plus 1 and $r_{\max}$),
(4) choose randomly the variables of $\mathcal{C}_i$ which belong to the separator.

We associate to each constraint a relation in which T tuples are forbidden ($0 \leqslant T \leqslant d^2$). An important drawback of this generator is that the number of constraints depends on the produced problem. For each class $(n, d, r_{\max}, T, s_{\max})$, we solve 100 problems and present the average of obtained results. The given results correspond to problems of the classes $(50, 25, 15, T, 5)$ with T between 265 and 281. Theses classes are near the satisfiability's threshold.

5.2.3. Real-world instances

We experiment our algorithm on some real-world instances of the CELAR from the FullRLFAP archive.[2] These instances correspond to radio link frequency assignment problems. For more details, they are described in [10]. Note that solving the problems

[2] We thank the Centre d'Electronique de l'Armement (France).

SCEN#01 and SCEN#08 requires a special adaptation of our implementation of BTD because these problems have a constraint graph with several connected components.

5.3. Experimental results for classical random CSPs

5.3.1. Comparisons of the different versions of BTD

Before comparing BTD to some classical algorithms like FC or MAC, we study the behaviour of our algorithm. First, we assess the contribution of backjumping by counting the number of nodes developed by FC-BTD which are not visited by FC-BTD-BJ. We observe there is no gain for most classes and a slight one for classes $(50, 15, 123, 141)$ or $(50, 25, 123, 439)$ (the classes we use are given in Table 1). But, even if there is a gain, it is insignificant. As a good or a nogood is recorded each time BTD comes back from a cluster to its parent, we can say, according to the little number of recorded goods and nogoods, that FC-BTD and FC-BTD-BJ rarely visit the descendants of the root cluster. Therefore, the phase of backjumping is seldom used, which explains that FC-BTD and FC-BTD-BJ obtain similar or equal results for classical random problems.

Then, we measure the contribution of goods and nogoods by counting the number of nodes developed by FC-BTD-BJ$^-$ which are not visited by FC-BTD-BJ. Like the previous comparison, there is no gain or a slight one. Indeed only a few goods or nogoods are used by FC-BTD-BJ to prune the search because of the little number of recorded goods and nogoods. And so FC-BTD-BJ$^-$ and FC-BTD-BJ present similar results. For information, we obtain similar results with MAC-BTD. As the various versions of BTD based on FC (respectively on MAC) obtain similar results, for the following comparisons on classical random problems, we only present the results of FC-BTD-BJ (respectively MAC-BTD-BJ).

5.3.2. Comparisons between FC-BTD-BJ and FC and between MAC-BTD-BJ and MAC

Table 1 presents the number of nodes and of constraint checks and the run-time for FC and FC-BTD-BJ. We observe that FC-BTD-BJ and FC are comparable. And even, for some classes, FC-BTD-BJ improves the results of FC, by developing fewer nodes and realizing fewer constraint checks than FC.

Similar results are obtained with MAC and MAC-BTD-BJ, as shown in Table 2.

Table 1
(Classical random CSPs.) Number of nodes, and number of constraint checks and run-time (in milliseconds) for FC and FC-BTD-BJ

Class	FC			FC-BTD-BJ		
	# nodes	# checks	time	# nodes	#checks	time
(50, 15, 123, 141)	15,884	458,342	250	19,417	541,178	263
(50, 15, 184, 112)	223,588	7,346,620	3,775	229,901	7,521,911	3,490
(50, 15, 245, 93)	1,742,077	64,695,274	31,613	1,690,389	62,741,411	28,045
(50, 15, 306, 78)	6,695,576	275,447,261	130,334	6,516,523	268,222,843	122,202
(50, 15, 368, 68)	19,899,917	865,863,076	410,365	20,202,681	880,491,613	374,439
(50, 25, 123, 439)	148,793	5,968,598	3,164	183,304	7,106,934	3,416

Table 2
(Classical random CSPs.) Number of nodes, and number of constraint checks and run-time (in milliseconds) for MAC and MAC-BTD-BJ

Class	MAC			MAC-BTD-BJ		
	# nodes	# checks	time	# nodes	# checks	time
(50, 15, 123, 141)	433	211,854	158	426	212,751	160
(50, 15, 184, 112)	10,570	4,749,549	4,366	10,589	4,767,163	4,468
(50, 15, 245, 93)	115,272	53,354,043	55,693	111,641	51,618,005	52,203
(50, 15, 306, 78)	577,928	263,294,873	293,339	560,541	255,317,033	279,650
(50, 15, 368, 68)	2,024,325	936,053,949	1,082,427	2,053,352	948,798,297	1,101,599
(50, 25, 123, 439)	2,912	2,600,557	1,767	2,703	2,476,033	1,674

Table 3
(Classical random CSPs.) Number of nodes, and number of constraint checks and run-time (in milliseconds) for FC-CBJ

Class	FC-CBJ			FC-BTD-BJ		
	# nodes	# checks	time	# nodes	# checks	time
(50, 15, 123, 141)	13,820	407,967	285	19,417	541,178	263
(50, 15, 184, 112)	214,314	7,089,277	4,657	229,901	7,521,911	3,490
(50, 15, 245, 93)	1,707,839	63,628,692	39,310	1,690,389	62,741,411	28,045
(50, 15, 306, 78)	6,612,237	272,582,414	160,745	6,516,523	268,222,843	122,202
(50, 15, 368, 68)	19,722,533	859,100,282	504,513	20,202,681	880,491,613	374,439
(50, 25, 123, 439)	127,093	5,208,464	3,613	183,304	7,106,934	3,416

5.3.3. Comparisons between FC-BTD-BJ and FC-CBJ

As FC-BTD-BJ exploits backjumping and "forwardjumping", we compare our algorithm with a classical backjumping algorithm, namely FC-CBJ. Table 3 provides the number of nodes, of constraint checks and the run-time for FC-CBJ. We observe that FC-CBJ often develops fewer nodes than FC-BTD-BJ. However, if we consider the run-time, we note that FC-BTD-BJ is faster than FC-CBJ for all the classes. A partial explanation of such a result is the cost of the computation of the conflicts which is too expensive compared to the number of saved nodes.

5.3.4. Comparisons between BTD and Tree-Clustering

We compare the space requirements for FC-BTD-BJ and our partial version of Tree-Clustering. In order to measure the memory requirement, we count one unit per assigned value contained in the recorded partial instantiation. For example, recording a good about five variables requires five units. Table 4 presents the memory required by FC-BTD-BJ (for recording goods and nogoods), the memory required by TC-FC (for recording consistent instantiations respectively on separators and on clusters), the number of developed nodes and the run-time (in milliseconds) for TC-FC. We observe that TC-FC requires significantly more memory than FC-BTD-BJ, because FC-BTD-BJ records only a part of the goods which TC-FC memorizes. Note that for some classes like $(50, 25, 123, 439)$, TC-FC requires too much memory in practice. Furthermore, TC-FC develops significantly

Table 4
(Classical random CSPs.) Comparison between FC-BTD-BJ and Tree-Clustering based on FC

Class	FC-BTD-BJ	TC-FC			
	memory	memory		# nodes	time
		separator	cluster		
(50, 15, 123, 141)	24.7	219,402	406,212,164	155,668,480	62,994
(50, 15, 184, 112)	9.9	163,523	1,840,482	942,758	8,752
(50, 15, 245, 93)	1.3	33,217	401,269	2,438,672	38,894
(50, 15, 306, 78)	0.5	11,620	199,244	12,932,108	226,546
(50, 15, 368, 68)	0.1	7,052	53,470	25,859,906	492,491
(50, 25, 123, 439)	19.2	1,560,479	375,943,617	89,379,304	106,367

more nodes and is slower than FC-BTD-BJ. So it seems difficult to use Tree-Clustering in practice.

5.3.5. Summary

FC-BTD and MAC-BTD obtain results which are comparable with ones of FC (or FC-CBJ) and MAC respectively. It seems difficult to use Tree-Clustering in practice, due to the required space.

5.4. Experimental results with structured random CSPs

5.4.1. Comparisons of the different versions of BTD

Like for classical problems, before making a comparison between BTD and classical algorithms like FC, FC-CBJ or MAC, we study the behaviour of our algorithm. First, with a view to comparing FC-BTD and FC-BTD-BJ, we assess the contribution of the backjumping by counting the number of nodes developed by FC-BTD which are not visited by FC-BTD-BJ. Fig. 7 presents the number of nodes developed by FC-BTD and FC-BTD-BJ. We note on this figure that FC-BTD-BJ develops significantly fewer nodes than FC-BTD. The economy in term of number of nodes varies between 8% and 26%. However, using the backjumping has a cost. Indeed, according to Fig. 8 (which reports the run-time for FC-BTD and FC-BTD-BJ), we observe that the gain in time is slightly less important than one in nodes. It is bounded by 5% and 19%.

In order to assess the contribution of goods and nogoods, we count the number of nodes developed by FC-BTD-BJ$^-$ which are not visited by FC-BTD-BJ. According to Fig. 9, it turns out that FC-BTD-BJ always develops fewer nodes than FC-BTD-BJ$^-$ and the gain is very important in some cases, namely near the satisfiability's threshold. The two algorithms differ only in recording and using goods and nogoods. It ensues that the gain in nodes is obtained thanks to the use of goods and nogoods. This gain leads to an economy in time, as shown in Fig. 10 (which presents the run-time for FC-BTD-BJ and FC-BTD-BJ$^-$).

Similar experimentations are realized with FC-BTD and FC-BTD$^-$. First, it results from these experimentations that FC-BTD$^-$ is unable to solve some instances in one hour. Table 5 gives their number. Therefore, in order to compare FC-BTD and FC-BTD$^-$, we take into account the problems solved by FC-BTD$^-$. Fig. 11 shows the number of nodes developed by FC-BTD and FC-BTD$^-$.

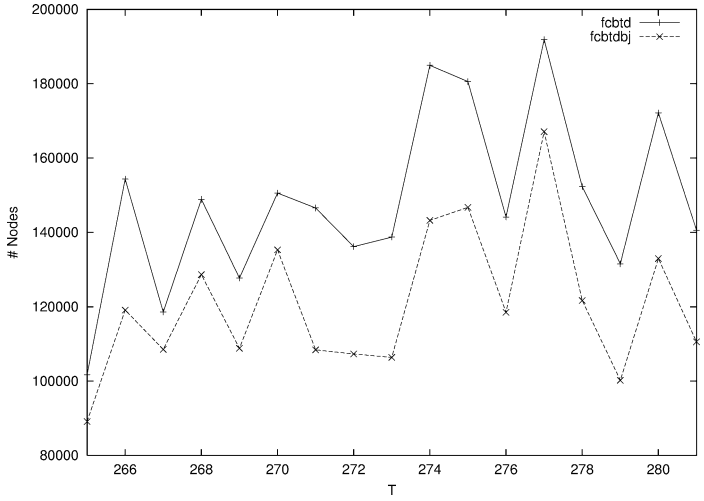


Fig. 7. (Structured random CSPs $(50, 25, 15, T, 5)$.) Number of nodes developed by FC-BTD and FC-BTD-BJ.

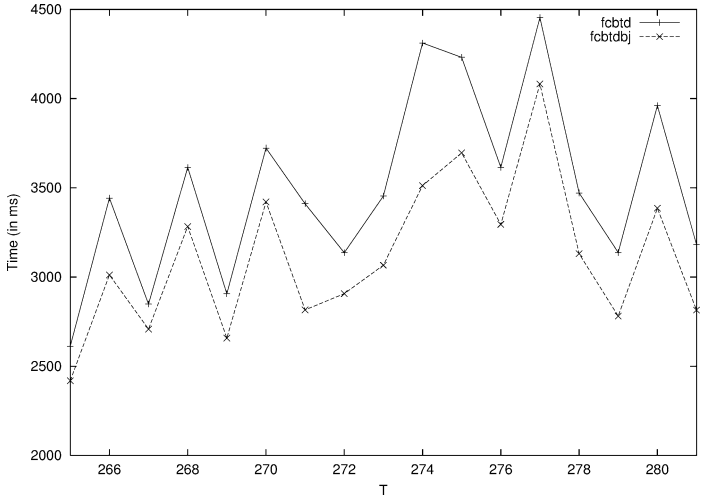


Fig. 8. (Structured random CSPs $(50, 25, 15, T, 5)$.) Run-time (in milliseconds) for FC-BTD and FC-BTD-BJ.

Then, we observe that FC-BTD develops fewer nodes than FC-BTD$^-$, thanks to the use of goods and nogoods. Furthermore, the difference between FC-BTD and FC-BTD$^-$ is more important than one between FC-BTD-BJ and FC-BTD-BJ$^-$. This gap highlights a lot of redundancies in the search tree developed by FC-BTD$^-$, which underlines all the more the contribution of goods and nogoods and/or of the phase of backjumping (because FC-BTD-BJ$^-$ is not so penalized as FC-BTD$^-$).

According to the previous results, we focus our study on FC-BTD-BJ for the next comparisons.

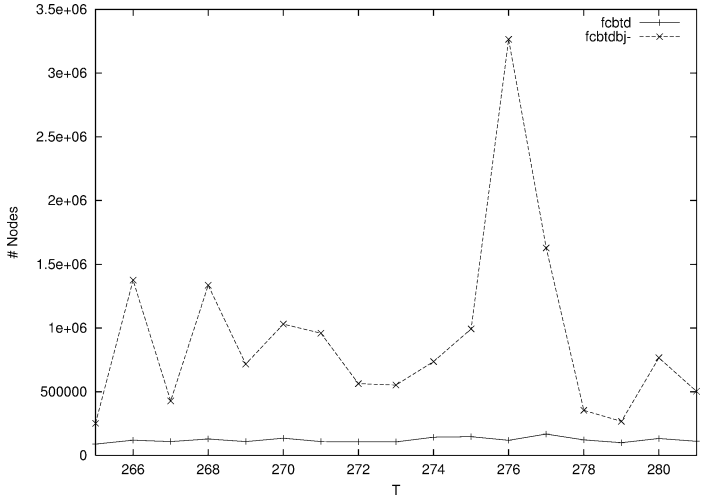


Fig. 9. (Structured random CSPs $(50, 25, 15, T, 5)$.) Number of nodes developed by FC-BTD-BJ and FC-BTD-BJ$^-$.

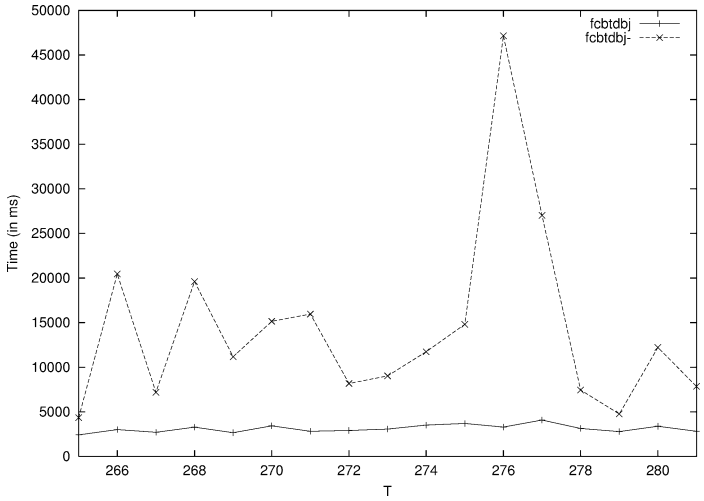


Fig. 10. (Structured random CSPs $(50, 25, 15, T, 5)$.) Run-time (in milliseconds) for FC-BTD-BJ and FC-BTD-BJ$^-$.

5.4.2. Comparisons between FC-BTD-BJ and FC and between MAC-BTD-BJ and MAC

FC and MAC are unable to solve some problems in one hour. Hence, in order to compare FC (respectively MAC) and FC-BTD-BJ (respectively MAC-BTD-BJ) we consider only the instances which FC (respectively MAC) can solve in one hour. Table 5 gives the number of problems solved by FC (respectively MAC). Note that FC-BTD-BJ and MAC-BTD-BJ solve all the considered instances.

Table 5
(Structured random CSPs (50, 25, 15, T, 5).) Number of consistent (C), inconsistent (I) and unknown (U) problems

T	FC-BTD		FC-BTD$^-$			FC			MAC		
	C	I	C	I	U	C	I	U	C	I	U
265	70	30	70	30	0	67	30	3	67	30	3
266	61	39	60	38	2	56	39	5	55	35	10
267	63	37	62	36	2	61	36	3	60	36	4
268	57	43	56	42	2	55	42	3	54	42	4
269	63	37	62	37	1	58	35	7	54	35	11
270	60	40	60	39	1	53	40	7	53	39	8
271	53	47	51	46	3	46	47	7	43	47	10
272	51	49	49	48	3	47	49	4	44	49	7
273	51	49	50	46	4	45	45	10	44	45	11
274	39	61	38	60	2	34	61	5	32	60	8
275	37	63	35	62	3	32	60	8	31	58	11
276	29	71	26	70	4	27	71	2	24	71	5
277	39	61	36	57	7	34	61	5	33	59	8
278	35	65	33	65	2	26	64	10	25	64	11
279	41	59	39	57	4	35	57	8	33	56	11
280	24	76	24	72	4	21	75	4	20	75	5
281	27	73	26	72	2	25	72	3	24	72	4

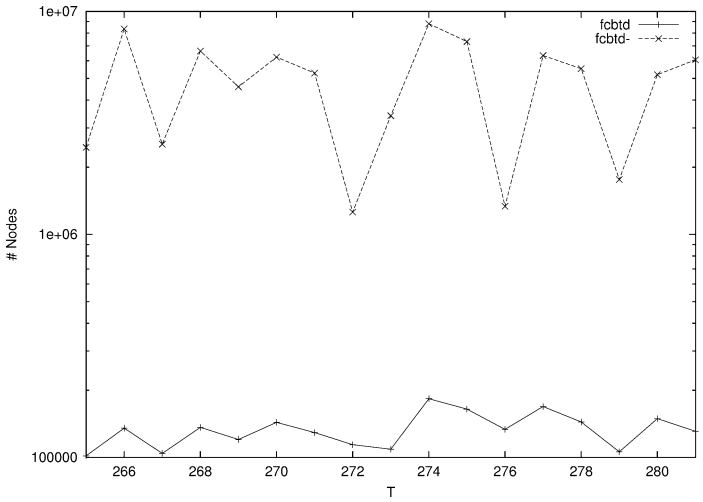


Fig. 11. (Structured random CSPs (50, 25, 15, T, 5).) Number of nodes developed by FC-BTD and FC-BTD$^-$ (with a log scale).

Fig. 12 presents the run-time for FC, FC-BTD-BJ and FC-BTD-BJ$^-$. We note that FC-BTD-BJ is significantly faster than FC. Indeed, the ratio of the run-time for FC over one for FC-BTD-BJ is between 7 and 24. We save time not only thanks to the goods and nogoods, but also thanks to the backjumping. Indeed, the contribution of the backjumping is proved by the run-time for FC-BTD-BJ$^-$, which is better than one of FC in most cases.

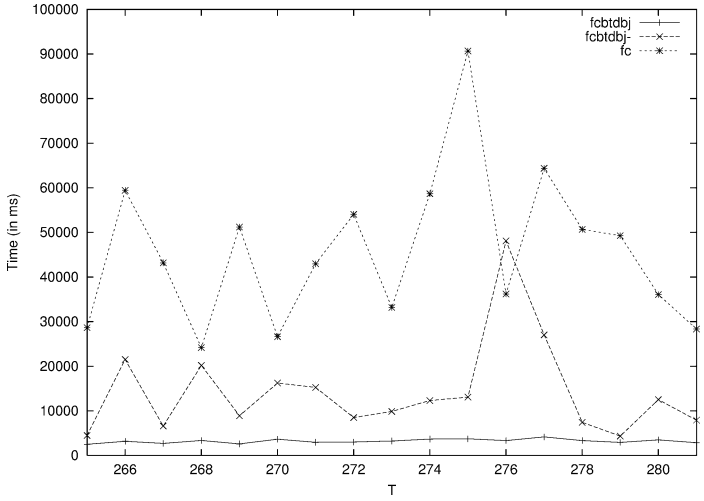


Fig. 12. (Structured random CSPs $(50, 25, 15, T, 5)$.) Run-time (in milliseconds) for FC-BTD-BJ, FC-BTD-BJ$^-$ and FC.

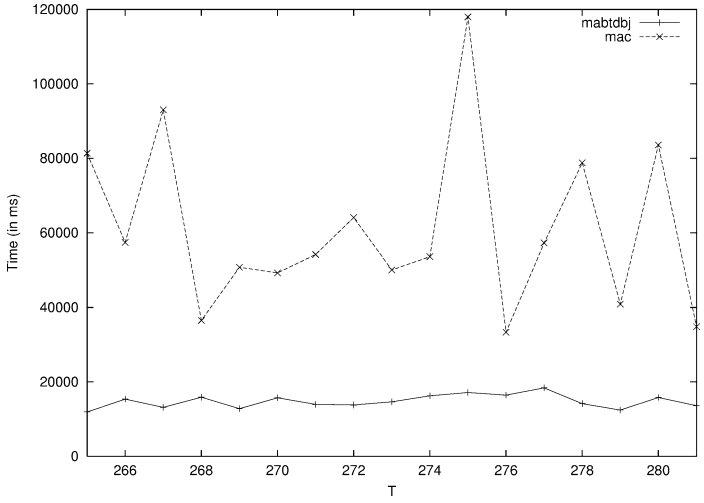


Fig. 13. (Structured random CSPs $(50, 25, 15, T, 5)$.) Run-time (in milliseconds) for MAC and MAC-BTD-BJ.

We obtain similar results when we compare MAC and MAC-BTD-BJ, as is shown by Fig. 13. MAC-BTD-BJ is between 2 and 7 times as fast as MAC.

5.4.3. Comparisons between FC-BTD-BJ and FC-CBJ

As FC-BTD-BJ uses backjumping and "forwardjumping", we have to compare FC-BTD-BJ with an algorithm which exploits backjumping like FC-CBJ. Figs. 14 and 15 present the number of nodes and the run-time for FC-CBJ and FC-BTD-BJ. About the number of nodes, neither FC-CBJ nor FC-BTD-BJ is always better than the other one.

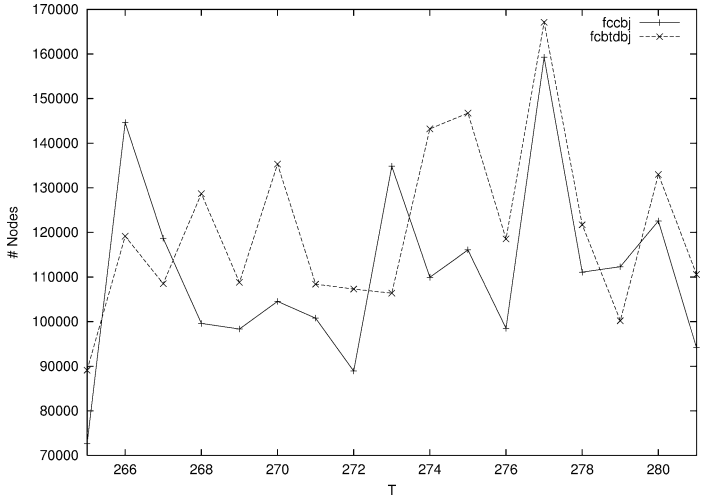


Fig. 14. (Structured random CSPs $(50, 25, 15, T, 5)$.) Number of nodes developed by FC-CBJ and FC-BTD-BJ.

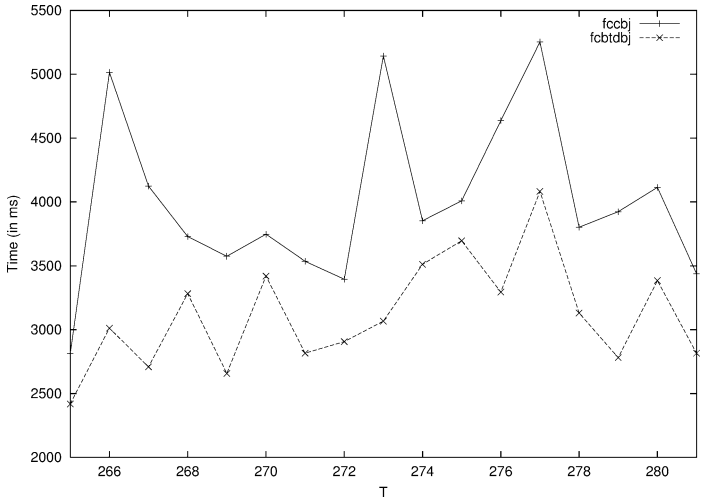


Fig. 15. (Structured random CSPs $(50, 25, 15, T, 5)$.) Run-time (in milliseconds) for FC-CBJ and FC-BTD-BJ.

Nonetheless, FC-BTD-BJ is always faster than FC-CBJ. This difference in time is mostly explained by the cost of the computation of conflicts in FC-CBJ which is too important in comparison with the gain obtained thanks to backjumping.

5.4.4. Comparisons between BTD and Tree-Clustering

Like for classical random problems, we compare the space requirements for FC-BTD-BJ and our partial version of Tree-Clustering. Table 4 shows the memory requirement of FC-BTD-BJ (for recording good ands nogoods), the memory requirement of TC-FC (for recording consistent instantiations respectively on separators and on clusters), the

Table 6
(Structured random CSPs $(50, 25, 15, T, 5)$.) Memory requirements for FC-BTD-BJ and Tree-Clustering based on FC

T	FC-BTD-BJ	TC-FC			
	memory	memory		# nodes	time
		separator	cluster		
265	3,599	563,376	16,269,923	4,329,007	14,718
266	4,054	544,683	15,741,988	4,081,041	13,270
267	3,471	391,140	13,536,010	3,630,053	12,486
268	4,174	500,454	14,331,723	3,779,950	12,439
269	3,218	426,517	12,862,473	3,477,518	11,743
270	5,567	457,120	13,071,460	3,505,728	11,652
271	5,005	413,560	13,010,768	3,451,125	11,170
272	4,273	453,395	11,745,237	3,192,403	10,770
273	5,476	401,098	11,476,495	3,083,502	10,286
274	9,008	444,808	10,237,218	2,805,207	9,750
275	5,289	393,569	9,107,353	2,575,782	9,324
276	5,134	342,977	8,301,716	2,385,563	8,921
277	8,408	379,848	9,794,940	2,712,032	9,246
278	5,910	350,243	8,589,484	2,384,354	8,370
279	6,734	416,477	8,265,270	2,307,709	7,917
280	8,304	319,735	7,267,237	2,066,851	7,398
281	5,637	247,736	6,232,299	1,790,943	6,554

number of developed nodes and the run-time (in milliseconds) for TC-FC. We observe that FC-BTD-BJ outperforms TC-FC by requiring significantly less memory. Furthermore, it develops fewer nodes and is faster than TC-FC. So, the use of Tree-Clustering seems difficult in practice.

5.4.5. Summary

Among the different versions of FC-BTD (respectively MAC-BTD), the best one is FC-BTD-BJ (respectively MAC-BTD-BJ). FC-BTD-BJ and MAC-BTD-BJ are significantly faster than FC and MAC respectively. Note that FC and MAC are unable to solve some instances. FC-BTD-BJ is faster than FC-CBJ although they develop comparable number of nodes. FC-BTD-BJ requires fewer memory is faster than TC-FC.

5.5. Real-world instances

Table 7 presents the results obtained for some instances of the CELAR from the FullRLFAP archive. In several cases, MAC-BTD-BJ realizes either fewer constraint checks than MAC or as many as MAC, except for the SCEN#02 instance for which MAC-BTD-BJ does a few additional checks. About the run-time, MAC-BTD-BJ and MAC are comparable, except for the SCEN#05 instance. For this instance, MAC-BTD-BJ is significantly faster than MAC thanks to its reduced number of constraint checks.

We do not give any results about TC-FC because TC-FC is unable to find all solutions of the root cluster for all problems except the obviously inconsistent ones.

Table 7
(Real-world Instances.) Number of constraint checks and run-time (in milliseconds) of MAC and MAC-BTD-BJ for some instances of the FullRLFAP archive

Instance	MAC		MAC-BTD-BJ	
	# checks	time	# checks	time
SCEN#01	1,857,660	610	1,855,040	790
SCEN#02	427,104	120	427,306	150
SCEN#03	947,199	300	930,909	400
SCEN#04	246,034	90	246,013	120
SCEN#05	9,220,866	15,380	1,190,682	210
SCEN#06	691,367	90	691,367	80
SCEN#07	1,123,856	110	1,123,856	110
SCEN#08	2,346,455	240	2,346,455	230
SCEN#09	84	10	84	10
SCEN#10	84	10	84	10
SCEN#11	22,520,823	25,520	22,513,770	25,230

5.6. Summary about experimental results

In this section, we have presented experiments on three kinds of CSPs benchmarks:

- Classical random CSPs.
- Structural random CSPs.
- Real-world instances.

For the first class, BTD, that is FC-BTD or MAC-BTD, obtains similar results than FC or MAC. So, the exploitation of the structure does not slow down the efficiency of search. For structured random CSPs, we have observed a significant improvement of the search in using FC-BTD (respectively MAC-BTD) with respect to FC (respectively MAC). We also have observed that FC-CBJ develops as many nodes as FC-BTD, but FC-BTD is faster. Finally, on real-world instances, BTD obtains either better results than classical algorithms, or comparable ones.

For these different kinds of benchmarks, we have observed that Tree-Clustering cannot be run for two reasons. On the one hand, its practical time complexity is too high. On the other hand, the required space is really prohibitive, making this method untractable while this criterion does not constitute a problem for BTD.

To conclude, BTD seems to be an approach which can exploit structural features of CSPs, without the drawbacks of other structural decomposition methods related to space complexity.

6. Related works

We can classify related works in three principal trends:

- Backtracking exploiting structural goods and nogoods as in Bayardo and Miranker [4, 5].
- Tree-Clustering [15] and its theoretical improvements [19].
- Hybrid approaches trying compromise between Tree-Clustering (or adaptive consistency [15]) and Backtracking [13,24].

As indicated in the presentation of BTD (see Section 3.2), the closest works are ones of Bayardo and Miranker in [4] and in [5]. Note that our approach can be considered as a natural generalization of [4] since their study is limited to acyclic binary CSPs (trees). With respect to [5], while the exploitation of goods and nogoods is similar to ours, our notions of goods and nogoods are formally different. In [5], a good (or a nogood) is defined with respect to a variable x_i and to an ordering on vertices. A good (or a nogood) is an assignment of a set of variables which precede x_i in the ordering and are connected to at least one variable belonging to the descendants of x_i in the tree-decomposition. This definition is thus formally different from ours. For example, if we consider a triangulated constraint graph, and $x_i \in \mathcal{C}_j$, the last variable in $\mathcal{C}_j$, then a good (or a nogood) will be an assignment of $\mathcal{C}_j \backslash \{x_i\}$. Then, the space requirement of Learning-Tree-Solve (the algorithm of [5]) will be $O(n.d^{w^+ + 1})$ ($w^+ + 1$ is the size of the largest $\mathcal{C}_j$) while the space requirement of BTD is limited to $O(n.d^s)$ with s the size of the largest separator. The time complexity of Learning-Tree-Solve is $O(\exp(w^+ + 1))$ like BTD. Note that these comments do not constitute an analysis but present some elements for a comparison that indicate the formal difference between these methods.

Finally, the practical interest of Learning-Tree-Solve is not presented in [5]. Moreover, in [6], Bayardo and Pehoushek recall the practical advantages on exploiting nogoods for consistency checking. Nevertheless they have also evoked the difficulty to implement efficiently this notion of goods which is not realized neither in [5] nor in [6].

The work of Baget and Tognetti [9] can be considered as a similar approach. Indeed, in their method, clusters are defined by biconnected components, and then goods and nogoods (they do not use these expressions) are limited to the assignment of one variable, the one which separates biconnected components. The time complexity of their method is then $O(n.d^k)$ with k the maximum size of biconnected components. In this case, $w^+ + 1 \leqslant k$. If we consider the constraint graph in Fig. 1, we get two biconnected components, $\{E, F, G\}$ and $\{A, B, C, D, H, I, J, K, L, M, N, O\}$, and then, $k = 12$ while $w^+ = 3$. Nevertheless, Baget and Tognetti indicated a few ways to improve their approach exploiting a generalization to k-connected components. Note that no experimental result is presented in [9].

BTD is principally based on tree-decomposition. So, works which have been developed like Tree-Clustering and its improvements are interesting for our purpose. In [19], an improvement of Tree-Clustering is presented while a theoretical comparison between decomposition methods is given. These results may indicate ways for (theoretical) improvements of BTD but we are not sure of their practical effects.

BTD can be considered as an hybrid approach realizing a tradeoff between practical time and space complexity. In [13], Dechter and El Fattah present a time-space tradeoff scheme. This scheme allows them to propose a spectrum of algorithms such that tree-clustering and cycle-cutset conditioning (linear for space complexity) are two extremes in

this spectrum. Another interesting idea in their work is the possibility to modify the size of separators to minimize space. We have exploited this idea in Section 5 to minimize the size of separators. Finally, note that their experimental results are limited to the valuation of structural parameters (w^+ and s) on real-world structured instances (combinatorial circuits), and then no result on the efficiency in solving these instances is presented.

In [24], Larrosa proposes an hybrid method based on Adaptive Consistency [15] and on Backtracking (or FC or MAC). Adaptive Consistency (AdCons) relies on the general scheme of variable elimination which replace sets of variables by new constraints which summarize the effects of eliminated variables. AdCons has the same bounds as Tree-Clustering for time and space complexities. So, exponential space complexity limits severely the algorithm usefulness. The idea of Larrosa consists in limiting the size of the new constraints produced by AdCons to a parameter k. If larger arity constraints should be produced, then it switches to search (BT, FC, MAC, ...). This hybrid approach allows to bound the required space to $O(d^k)$ but the time complexity is now $O(\exp(z(k) + k + 1))$. Here $z(k)$ is a structural parameter induced by k and the width of the constraint graph such that $z(k) + k < n$. Note that for sparse constraint graphs (6 per cent), and limited values of k ($k = 2$), the author obtains interesting results on random CSPs.

7. Summary and conclusion

The CSP formalism offers a powerful framework for representing and solving efficiently many problems. Generally, CSPs are solved applying tree search algorithms which use optimizations of backtracking and then obtain good experimental results. However, since CSP is a NP-complete problem, there are no better bound for theoretical time complexity than the size of the search space, which is exponential. On the contrary, methods which offer better bounds for time complexity—which are generally based on tree-decomposition of CSPs—have not proved yet their practical efficiency. This paper presents a framework—BTD—for solving CSPs. BTD is based both on backtracking techniques and on the notion of tree-decomposition of the constraint network.

We have shown that BTD inherits the advantages of the two other approaches: the practical efficiency of backtracking algorithms, and a warranty of limited time/space complexity. In Section 4, we have proved that the theoretical time and space complexities of BTD are similar to Tree-Clustering's ones, namely a time complexity in $O(n.s^2.m.\log(d^s).d^{w^+ + 1})$ and a space complexity in $O(n.s.d^s)$. Moreover, experiments allow us to show that:

- BTD is as efficient as classical algorithms on classical random problems, in some cases, it is even better,
- on structured random problems, BTD presents a significant gain thanks to the exploitation of goods and nogoods,
- on real-world instances, BTD obtains either better results than classical algorithms, or comparable ones,
- about required space, BTD can be used in practice, unlike Tree-Clustering which is too expensive in memory.

Among the potential extensions of this method, the first one concerns the generalization to n-ary CSPs, which should not raise much difficulty, because it is immediately obtained by construction. A more promising extension is related to optimization tasks. In fact, if we consider, for instance, the valued CSP framework [32], methods like Russian Dolls Search [36] or the dynamic programming approach [22] are among the most efficient ones. These methods record and exploit some informations they explicit during the search. Now, if we exploit a method like BTD which limits the number of recorded informations, we can expect significant gains in practice. Finally, the theoretical comparison between BTD and BT (respectively FC-BTD vs FC and MAC-BTD vs MAC) should be extended in the future to consider different orders.

References

[1] S. Arnborg, D. Corneil, A. Proskuroswki, Complexity of finding embedding in a k-tree, SIAM J. Discrete Math. 8 (1987) 277–284.
[2] A. Berry, A wide-range efficient algorithm for minimal triangulation, in: Proceedings of SODA'99 SIAM Conference, 1999.
[3] A. Becker, D. Geiger, A sufficiently fast algorithm for finding close to optimal clique trees, Artificial Intelligence 125 (2001) 3–17.
[4] R.J. Bayardo, D.P. Miranker, An optimal backtrack algorithm for tree-structured constraint satisfaction problems, Artificial Intelligence 71 (1994) 159–181.
[5] R.J. Bayardo, D.P. Miranker, A complexity analysis of space-bounded learning algorithms for the constraints satisfaction problem, in: Proceedings of 13th National Conference on Artificial Intelligence, Portland, OR, 1996, pp. 298–304.
[6] R. Bayardo, J. Pehoushek, Counting models using connected components, in: Proceedings of AAAI 2000, Austin, TX, 2000, pp. 157–162.
[7] C. Bessière, J.-C. Régin, MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems, in: Proceedings of CP'96, 1996, pp. 61–75.
[8] C. Bessière, J.-C. Régin, Refining the basic constraint propagation algorithm, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence, Seattle, WA, 2001, pp. 309–315.
[9] J.-F. Baget, Y. Tognetti, Backtracking through biconnected components of a constraint graph, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence, Seattle, WA, 2001, pp. 291–296.
[10] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, J.P. Warners, Radio link frequency assignment, Constraints 4 (1999) 79–89.
[11] X. Chen, P. van Beek, Conflict-directed backjumping revisited, J. Artificial Intelligence Res. 14 (2001) 53–81.
[12] R. Dechter, Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition, Artificial Intelligence 41 (1990) 273–312.
[13] R. Dechter, Y. El Fattah, Topological parameters for time-space tradeoff, Artificial Intelligence 125 (2001) 93–118.
[14] R. Dechter, J. Pearl, The cycle-cutset method for improving search performance in AI applications, in: Proceedings of the Third IEEE on Artificial Intelligence Applications, 1987, pp. 224–230.
[15] R. Dechter, J. Pearl, Tree-clustering for constraint networks, Artificial Intelligence 38 (1989) 353–366.
[16] E. Freuder, A sufficient condition for backtrack-free search, J. ACM 29 (1982) 24–32.
[17] J. Gaschnig, Performance Measurement and Analysis of Certain Search Algorithms, Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.
[18] M. Ginsberg, Dynamic backtracking, J. Artificial Intelligence Res. 1 (1993) 25–46.
[19] G. Gottlob, N. Leone, F. Scarcello, A comparison of structural CSP decomposition methods, Artificial Intelligence 124 (2000) 282–343.
[20] M. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[21] R. Haralick, G. Elliot, Increasing tree search efficiency for constraint satisfaction problems, Artificial Intelligence 14 (1980) 263–313.
[22] A. Koster, Frequency assignment—models and algorithms, PhD Thesis, University of Maastricht, November 1999.
[23] G. Kondrak, P. van Beek, A theorical evaluation of selected backtracking algorithms, Artificial Intelligence 89 (1997) 365–387.
[24] J. Larrosa, Boosting search with variable elimination, in: Proceedings of the 6th CP, 2000.
[25] A. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1977) 99–118.
[26] U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, Artificial Intelligence 7 (1974) 95–132.
[27] B. Nadel, Tree search and arc consistency in constraint-satisfaction algorithms, in: Search in Artificial Intelligence, Springer, Berlin, 1988, pp. 287–342.
[28] P. Prosser, Hybrid algorithms for the constraint satisfaction problem, Comput. Intelligence 9 (1993) 268–299.
[29] N. Robertson, P.D. Seymour, Graph minors II: Algorithmic aspects of tree-width, Algorithms 7 (1986) 309–322.
[30] D. Rose, R. Tarjan, G. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (1976) 266–283.
[31] D. Sabin, E. Freuder, Contradicting conventional wisdom in constraint satisfaction, in: Proceedings of 11th ECAI, 1994, pp. 125–129.
[32] T. Schiex, H. Fargier, G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Quebec, 1995, pp. 631–637.
[33] R. Stallman, G. Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, Artificial Intelligence 9 (1977) 135–196.
[34] T. Schiex, G. Verfaillie, Nogood recording for static and dynamic constraint satisfaction problems, Internat. J. Artificial Intelligence Tools 3 (2) (1994) 187–207.
[35] R. Tarjan, M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM J. Comput. 13 (3) (1984) 566–579.
[36] G. Verfaillie, M. Lemaître, T. Schiex, Russian doll search for solving constraint optimization problems, in: Proceedings of the 13th AAAI, Portland, OR, 1996, pp. 181–187.

Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs

Philippe Jégou and **Samba Ndojh Ndiaye** and **Cyril Terrioux**
LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, samba-ndojh.ndiaye, cyril.terrioux}@univ-cezanne.fr

Abstract

This paper deals with methods exploiting tree-decomposition approaches for solving constraint networks. We consider here the practical efficiency of these approaches by defining five classes of variable orders more and more dynamic which preserve the time complexity bound. For that, we define extensions of this theoretical time complexity bound to increase the dynamic aspect of these orders. We define a constant k allowing us to extend the classical bound from $O(exp(w+1))$ firstly to $O(exp(w+k+1))$, and finally to $O(exp(2(w+k+1)-s^-))$, with w the "tree-width" of a CSP and s^- the minimum size of its separators. Finally, we assess the defined theoretical extension of the time complexity bound from a practical viewpoint.

1 Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. Modeling a problem as a CSP consists in defining a set X of variables $x_1, x_2, \ldots x_n$, which must be assigned in their respective finite domain, by satisfying a set C of constraints which express restrictions between the different possible assignments. A solution is an assignment of every variable which satisfies all the constraints. Determining if a solution exists is a NP-complete problem.

The usual method for solving CSPs is based on backtracking search, which, in order to be efficient, must use both filtering techniques and heuristics for choosing the next variable or value. This approach, often efficient in practice, has an exponential theoretical time complexity in $O(e.d^n)$ for an instance having n variables and e constraints and whose largest domain has d values. Several works have been developed, in order to provide better theoretical complexity bounds according to particular features of the instance. The best known complexity bounds are given by the "tree-width" of a CSP (often denoted w). This parameter is related to some topological properties of the constraint graph which represents the interactions between variables via the constraints. It leads to a time complexity in $O(n.d^{w+1})$ (denoted $O(exp(w+1))$). Different methods have been proposed to reach this bound like *Tree-Clustering* [Dechter and Pearl, 1989] (see [Gottlob *et al.*, 2000] for more details). They rely on the notion of tree-decomposition of the constraint graph. They aim to cluster variables such that the cluster arrangement is a tree. Depending on the instances, we can expect a significant gain w.r.t. enumerative approaches. Most of works based on this approach only present theoretical results. Except [Gottlob *et al.*, 2002; Jégou and Terrioux, 2003], no practical results have been provided. So, we study these approaches by concentrating us on the BTD method [Jégou and Terrioux, 2003] which seems to be the most effective method proposed until now within the framework of these structural methods.

While the problem of finding the best decomposition has been studied in the literature firstly from a theoretical point of view, recently, some studies have been realized in the field of CSP, integrating as quality parameter for a decomposition, its efficiency for solving the considered CSP [Jégou *et al.*, 2005]. Yet, these studies do not consider the questions related to an efficient use of the considered decompositions.

This paper deals with this question. Given a tree-decomposition, we study the problem of finding good orders on variables for exploiting this decomposition. As presented in [Jégou and Terrioux, 2003], the order on the variables is static and compatible with a depth first traversal of the associated cluster tree. Since enumerative methods highlight the efficiency of dynamic variable orders, we give conditions which allow to exploit in a more dynamic way the tree-decomposition and guarantee the time complexity bound. We propose five classes of orders respecting these conditions, two of them giving more freedom to order variables dynamically. Consequently, their time complexity possess larger bounds: $O(exp(w+k+1))$ and $O(exp(2(w+k+1)-s^-))$, where k is a constant to parameterize and s^- the minimum size of separators. Based on the properties of these classes, we exploit several heuristics which aim to compute a good order on clusters and more generally on variables. They rely on topological and semantic properties of CSP instance. Heuristics based on the expected number of solutions enhance significantly the performances of BTD. Meanwhile, those based on the cluster size or on the dynamic variable ordering heuristic provide often similar improvements and may outperform the first ones on real-world instances. Finally, we report here experiments to assess the interest of the extensions based on the time complexity bound.

This paper is organized as follows. The next section pro-

vides basic notions about CSPs and methods based on tree-decompositions. Then, in section 3, we define several classes of variable orders which preserve the classical bounds for time complexity. Section 4 introduces two extensions giving new time complexity bounds. Section 5 is devoted to experimental results to assess the practical interest of our propositions. Finally, in section 6, we summarize this work and we outline some future works.

2 Preliminaries

A *constraint satisfaction problem* (CSP) is defined by a tuple (X, D, C). X is a set $\{x_1, \ldots, x_n\}$ of n variables. Each variable x_i takes its values in the finite domain d_{x_i} from D. The variables are subject to the constraints from C. Given an instance (X, D, C), the CSP problem consists in determining if there is an assignment of each variable which satisfies each constraint. This problem is NP-complete. In this paper, without loss of generality, we only consider binary constraints (i.e. constraints which involve two variables). So, the structure of a CSP can be represented by the graph (X, C), called the *constraint graph*. The vertices of this graph are the variables of X and an edge joins two vertices if the corresponding variables share a constraint.

Tree-Clustering [Dechter and Pearl, 1989] is the reference method for solving CSPs thanks to the structure of its constraint graph. It is based on the notion of tree-decomposition of graphs [Robertson and Seymour, 1986]. Let $G = (X, C)$ be a graph, a *tree-decomposition* of G is a pair (E, T) where $T = (I, F)$ is a tree with nodes I and edges F and $E = \{E_i : i \in I\}$ a family of subsets of X, such that each subset (called cluster) E_i is a node of T and verifies: (i) $\cup_{i \in I} E_i = X$, (ii) for each edge $\{x, y\} \in C$, there exists $i \in I$ with $\{x, y\} \subseteq E_i$, and (iii) for all $i, j, k \in I$, if k is in a path from i to j in T, then $E_i \cap E_j \subseteq E_k$.

The width of a tree-decomposition (E, T) is equal to $max_{i \in I}|E_i| - 1$. The *tree-width* w of G is the minimal width over all the tree-decompositions of G.

Assume that we have a tree-decomposition of minimal width (w), the time complexity of Tree-Clustering is $O(exp(w + 1))$ while its space complexity can be reduced to $O(n.s.d^s)$ with s the size of the largest minimal separators of the graph [Dechter and Fattah, 2001]. Note that Tree-Clustering does not provide interesting results in practical cases. So, an alternative approach, also based on tree-decomposition of graphs was proposed in [Jégou and Terrioux, 2003]. This method is called BTD (for Backtracking with Tree-Decomposition) and seems to provide among the best empirical results obtained by structural methods.

The BTD method proceeds by an enumerative search guided by a static pre-established partial order induced by a tree-decomposition of the constraint-network. So, the first step of BTD consists in computing a tree-decomposition. The obtained tree-decomposition allows to exploit some structural properties of the graph, during the search, in order to prune some branches of the search tree, what distinguishes BTD from other classical techniques. Firstly, the order for the assignment of the variables is induced by the considered tree-decomposition of the constraint graph. Secondly, some parts of the search space will not be visited again as soon as their consistency is known. This is possible by using the notion of *structural good*. A good is a consistent partial assignment on a set of variables, namely a separator (i.e. an intersection between two clusters), such that the part of the CSP located after the separator is consistent and admits a solution compatible with the good. So, it is not necessary to explore this part because we know its consistency. Thirdly, some parts of the search space will not be visited again if we know that the current instantiation leads to a failure. This is possible in applying the notion of *structural nogood*. A structural nogood is a particular kind of nogood justified by structural properties of the constraints network: the part of the CSP located after the nogood is not consistent (a nogood is a consistent assignment of a separator of the graph).

To satisfy the complexity bounds, the variable ordering exploited in BTD is related to the cluster ordering. Formally, let us consider a tree-decomposition (E, T) of the CSP with $T = (I, F)$ a tree and assume that the elements of $E = \{E_i : i \in I\}$ are indexed w.r.t. a *compatible numeration*. A numeration on E compatible with a prefix numeration of $T = (I, F)$ with E_1 the root is called compatible numeration. An order $\preceq_X$ of variables of X such that $\forall x \in E_i$, $\forall y \in E_j$, with $i < j$, $x \preceq_X y$ is a compatible enumeration order. The numeration on the clusters gives a partial order on the variables since the variables in the E_i are assigned before those in E_j if $i < j$, except variables in the descent of a good, namely those located in the subproblem rooted on the cluster containing the good. In fact, using goods and nogoods allows not to explore twice subproblems if their consistency (inconsistency) with the current assignment is known. If we use a good to avoid visiting again a subtree, we known that the variables in it can be assigned consistently with the current assignment. So BTD does not assign them effectively, but they are considered done. For consistent problems, an additional work must be performed to assign these variables if we want to provide a solution [Jégou and Terrioux, 2004]. They are named *consistently assignable variables* thanks to a good. Thus the variables in E_j are assigned if the variables in E_i are either already assigned or consistently assignable thanks to a good. To complete this order, we have to choose variable ordering heuristics inside a cluster. Finally, a compatible enumeration order on the variables is given by a compatible numeration on clusters and a variable order in each cluster.

In [Jégou and Terrioux, 2003; 2004], the presented results were obtained without heuristics for the choice of the clusters and thus the choice of the variables. Only a traditional dynamic order was used inside the clusters. Obviously, the variable ordering have a great impact on the efficiency of enumerative methods. Thus, we study here how the benefits of variable orderings can be fully exploited in BTD. Nevertheless, to guarantee the time complexity bounds, it is necessary to respect some conditions. So, in the next section, we define classes of orders guaranteeing complexity bounds.

3 Dynamic orders in $O(exp(w + 1))$

The first version of BTD was defined with a compatible static variable ordering. We prove here that it is possible to consider

more dynamic orders without loosing the complexity bounds. The defined classes contain orders more and more dynamic. These orders are in fact provided by the cluster order and the variable ordering inside each cluster.
Let (X, D, C) be a CSP and (E, T) a tree-decomposition of the graph (X, C), we exploit an order σ_i on the subproblems $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}$ rooted on the sons E_{i_j} of E_i and an order γ_i on the variables in E_i. We define recursively the following classes of orders. In the three next classes, we choose the first cluster to assign (the root): E_1 among all the clusters and we consider $\mathcal{P}_1$ the subproblem rooted on E_1 (i.e. the whole problem).

Definition 1 *We begin the search in E_1 and we try recursively to extend the current assignment on the subproblem rooted on E_i by assigning first the variables in E_i according to γ_i and then on $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}$ according to σ_i.*

- ***Class 1.*** *σ_i* ***and*** *γ_i* ***are static****. We compute σ_i and γ_i statically (before starting the search).*
- ***Class 2.*** *σ_i* ***is static and*** *γ_i* ***is dynamic****. We compute statically σ_i, while γ_i is computed during the search.*
- ***Class 3.*** *σ_i* ***and*** *γ_i* ***are dynamic****. Both, σ_i and γ_i are computed during the search. σ_i is computed w.r.t. the current assignment as soon as all the variables in E_i are assigned.*
- ***Class ++. Enumerative dynamic order****. The variable ordering is completely dynamic. So, the assignment order is not necessarily a compatible enumeration order. There is no restriction due to the cluster tree.*

The defined classes form a hierarchy since we have: *Class 1* $\subset$ *Class 2* $\subset$ *Class 3* $\subset$ *Class ++*.
Property of the *Class 3*. Let Y be an assignment, $x \in E_j - (E_i \cap E_j)$ with E_i the parent of E_j: $x \in Y$ iff: i)$\forall v \in E_i$, $v \in Y$ ii) Let $E_{i_p} = E_j$, $\forall \mathcal{P}_{i_u}$ s.t. $\sigma_i(\mathcal{P}_{i_u}) \leq \sigma_i(\mathcal{P}_{i_p})$, $\forall v \in \mathcal{P}_{i_u}$, $v \in Y$ iii) $\forall v \in E_j$ s.t. $\gamma_j(v) \leq \gamma_j(x)$, $v \in Y$.

In [Jégou and Terrioux, 2003], the experiments use *Class 2* orders. Formally, only the orders of the *Class 1* are compatible. Nevertheless, for an order o_3 in the *Class 3* and a given assignment $\mathcal{A}$, one can find an order o_1 in the *Class 1* that instantiates the variables in $\mathcal{A}$ in the same way and the same order o_3 does. This property gives to the *Class 3* (thus *Class 2*) orders the ability of recording goods and nogoods and using them to prune branches in the same way *Class 1* orders do. The *Class ++* gives a complete freedom. Yet, it does not guarantee the time complexity bound because sometimes it is impossible to record nogoods. Indeed, let the cluster E_j be a son of the cluster E_i, we suppose that the enumeration order assigns the variables in E_i except those in $E_i \cap E_j$, as well as the variables in the clusters which are on the path from the root cluster to E_i. Let x, the next variable to assign, be in E_j and not in $E_i \cap E_j$. If the solving of the subtree rooted on E_j leads to a failure, it is impossible to record a nogood on $E_i \cap E_j$ (if it is consistently assigned) because we do not try the other values of x to prove that the assignment on $E_i \cap E_j$ cannot be consistently extended on this subtree. If the subproblem has a solution, we can record a good. Actually, this solution is a consistent extension of the assignment on $E_i \cap E_j$ which is a good. A nogood not recorded could be computed again. Thus the time complexity bound is not guaranteed anymore. Meanwhile, the *Class 3* orders guarantee this bound.

Theorem 1 *Let the enumeration order be in the Class 3, the time complexity of BTD is $O(exp(w+1))$.*

Proof We consider a cluster E_j in the cluster tree, and we must prove that any assignment on E_j is computed only once. Let E_i be the cluster parent of E_j and suppose that all the variables in E_i are assigned and those in $E_j - (E_i \cap E_j)$ are not. Since the order belongs to the *Class 3*, the variables of the clusters on the path from the root to E_i are already assigned and those in the subtree rooted on E_j not yet. A consistent assignment $\mathcal{A}$ on $E_i \cap E_j$ is computed at the latest when the variables in E_i are assigned (before those in the subproblem rooted in E_j). Solving this subproblem leads to a failure or a solution. In each case, $\mathcal{A}$ is recorded as a good or nogood. Let $\mathcal{A}'$ be an assignment on E_j compatible with $\mathcal{A}$. The next assignment of variables in E_i leading to $\mathcal{A}$ on $E_i \cap E_j$ will not be pursued on the subproblem rooted on E_j. $\mathcal{A}'$ is not computed twice, only the variables in $E_i \cap E_j$ are assigned again. So the time complexity is $O(exp(w+1))$. □
The properties of the *Class 3* offer more possibilities in the variable ordering. So it is possible to choose any cluster to visit next among the sons of the current cluster. And in each cluster, the variable ordering is totally free. In section 4, we propose two natural extensions of the complexity bound.

4 Bounded extensions of dynamic orders

We propose two extensions based on the ability given to the heuristics to choose the next variable to assign not only in one cluster, but also among k variables in a path rooted on the cluster that verifies some properties. So, we define two new classes of orders extending *Class 3*. First, we propose a generalization of the tree-decomposition definition.

Definition 2 *Let $G = (X, C)$ be a graph and k a positive integer, the set of directed k-covering tree-decompositions of a tree-decomposition (E, T) of G with E_1 its root cluster, is defined by the set of tree-decompositions (E', T') of G that verify:*

- *$E_1 \subseteq E'_1$, E'_1 the root cluster of (E', T')*
- *$\forall E'_i \in E'$, $E'_i \subseteq E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$ and $E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R - 1} \subset E'_i$, with $E_{i_1} \ldots E_{i_R}$ a path in T*
- *$|E'_i| \leq w + k + 1$, where $w = max_{E_i \in E}|E_i| - 1$*

Now, we give a definition of the *Class 4*.

Definition 3 *Let (X, D, C) be a CSP, (E, T) a tree-decomposition of the graph (X,C) and k a positive integer. A variable order is in the* ***Class 4****, if this order is in the Class 3 for one directed k-covering tree-decomposition of (E, T).*

We derive a natural theorem:

Theorem 2 *Let the enumeration order be in the Class 4, the time complexity of BTD is $O(exp(w+k+1))$.*

Proof This proof is similar to one given for Class 3 since we can consider that BTD runs on a tree-decomposition (E', T') of width at most $w + k + 1$. □

A second extension is possible in exploiting during the search, a dynamic computing of the tree-decomposition (we

can use several directed k-covering tree-decompositions during the search). Then, the time complexity bound changes because sometimes it would be impossible to record nogoods.

Definition 4 *Let (X, D, C) be a CSP, (E, T) a tree-decomposition of the graph (X,C) and k a positive integer. A variable order o_5 is in the **Class 5**, if for a given assignment $\mathcal{A}$, one can find one directed k-covering tree-decomposition (E', T') of (E, T) such that $\forall E'_i \in E'$, $E'_i = E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$, with $E_{i_1} \ldots E_{i_R}$ a path in T and find an order o_3 on (E', T'), in the Class 3 that instantiates the variables in $\mathcal{A}$ in the same way and the same order o_5 does.*

This definition enforces to use directed k-covering tree-decompositions (E', T') of (E, T) that verify the additional condition: $\forall E'_i \in E'$, $E'_i = E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$. Hence, a separator in (E', T') is also a separator in (E, T). We denote by s^- the minimum size of separators in (E, T).

Theorem 3 *Let the enumeration order be in the Class 5, the time complexity of BTD is $O(exp(2(w + k + 1) - s^-))$.*

Proof Let (X, D, C) be a CSP, (E, T) a tree-decomposition of the graph (X, C) and E_1 its root cluster. We have to prove that any assignment on a set V of $2(w+k+1)-s^-$ variables on a path of the tree T is computed only once. Let $\mathcal{A}$ be an assignment containing V. The order in which the variables of $\mathcal{A}$ were assigned is in the *Class 3* for a directed k-covering tree-decomposition (E', T') of (E, T) that verifies $\forall E'_i \in E'$, $E'_i = E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$, with $E_{i_1} \ldots E_{i_R}$ a path in T. The size of the clusters in (E', T') is bound by $w+k+1$, so the set V is covered by at least two clusters since s^- is the minimum size of the separators. Let $E'_{i_1} \ldots E'_{i_q}$ be a path on (E', T') covering V. The solving of the subproblem rooted on E'_{i_1} with the assignment $\mathcal{A}$ leads to the recording of (no)goods on the separators of these clusters. If E'_{i_1} is the root cluster of (E', T'), then V contains E_1. Thus $\mathcal{A}$ will not be computed again because it contains the first variables in the search. We suppose that E'_{i_1} is not the root cluster of (E', T'). Since $q \geq 2$, we record a (no)good on the separator of E'_{i_1} and its parent and at least an other on the separator of E'_{i_1} and E'_{i_2}. Let $\mathcal{B}$ be a new assignment that we try to extend on V with the same values in $\mathcal{A}$. One of the (no)goods will be computed first. Thus before all the variables in V are assigned, the search is stopped thanks to this (no)good. So $\mathcal{A}$ is not computed again. We prove that any assignment on V is computed only once.□

Note that the new defined classes are included in the hierarchy presented in section 3: *Class i* ⊂ *Class j*, if $i < j$ and for $1 \leq i < j \leq 5$, with also *Class 5* ⊂ *Class ++*.

To define the value of k, we have several approaches to choose variables to group. A good one consists in trying to reduce the value of the parameter s and, by this way, to enhance the space complexity bound. Then, we can observe that grouping clusters with large separators permits to achieve a significant reduction of s.

5 Experimental study

Applying a structural method on an instance generally assumes that this instance presents some particular topological features. So, our study is first performed on instances having a structure which can be exploited by structural methods. In practice, we assess here the proposed strategies on random partial structured CSPs in order to point up the best ones w.r.t. CSP solving. For building a random partial structured instance of a class (n, d, w, t, s, n_c, p), the first step consists in producing randomly a structured CSP according to the model described in [Jégou and Terrioux, 2003]. This structured instance consists of n variables having d values in their domain. Its constraint graph is a clique tree with n_c cliques whose size is at most w and whose separator size does not exceed s. Each constraint forbids t tuples. Then, the second step removes randomly $p\%$ edges from the structured instance. Secondly, we experiment the proposed heuristics on benchmark instances of the CP'2005 solver competition[1]. All these experimentations are performed on a Linux-based PC with a Pentium IV 3.2GHz and 1GB of memory. For each considered random partial structured instance class, the presented results are the average on instances solved over 50. We limit the runtime to 30 minutes for random instances and to 10 minutes for CP'2005 benchmark instances. Above, the solver is stopped and the involved instance is considered as unsolved (what is denoted by the letter T in tables). In the following tables, the letter M means that at least one instance cannot be solved because it requires more than 1GB of memory.

In [Jégou *et al.*, 2005], a study was performed on triangulation algorithms to find out the best way to compute a good tree-decomposition w.r.t. CSP solving. As MCS [Tarjan and Yannakakis, 1984] obtains the best results and is very easy to implement, we use it to compute tree-decompositions in this study. We do not provide the results obtained by classical enumerative algorithms like FC or MAC since they are often unable to solve several instances of each class within 30 minutes.

Here, for lack of place, we only present the more efficient heuristics:

- $minexp(A)$: this heuristic is based on the expected number of partial solutions of clusters [Smith, 1994] and on their size. It chooses as root cluster one which minimizes the ratio between the expected number of solutions and the size of the cluster.
- $size(B)$: we choose the cluster of maximum size as root cluster
- $minexp_s(C)$: this heuristic is similar to $minexp$ and orders the son clusters according to the increasing value of their ratio.
- $minsep_s(D)$: we order the son clusters according to the increasing size of their separator with their parent.
- $nv(E)$: we choose a dynamic variable ordering heuristic and we visit first the son cluster where appears the next variable in the heuristic order among the variables of the unvisited son clusters.
- $minexp_{sdyn}(F)$: the next cluster to visit minimizes the ratio between the current expected number of solutions and the size of the cluster. The current expected number of solutions of a cluster is modified by filtering the domains of unassigned variables.

[1] This competition held during the Second International Workshop on Constraint Propagation and Implementation of CP'2005.

CSP (n, d, w, t, s, n_c, p)	w^+	s	Class 1		Class 2		Class 3		Class 4			
			B D	A C	B D	A C	A F	B G	B D	A C	A F	B G
$(150, 25, 15, 215, 5, 15, 10)$	13.00	12.22	9.31	28.12	3.41	2.52	2.45	5.34	2.75	2.17	2.08	2.65
$(150, 25, 15, 237, 5, 15, 20)$	12.54	11.90	9.99	5.27	5.10	2.47	1.99	5.47	2.58	1.76	1.63	2.97
$(150, 25, 15, 257, 5, 15, 30)$	12.16	11.40	13.36	27.82	3.38	5.06	4.97	3.55	1.41	1.05	1.13	1.30
$(150, 25, 15, 285, 5, 15, 40)$	11.52	10.64	3.07	8.77	1.13	0.87	1.27	1.17	1.67	0.39	0.63	1.75
$(250, 20, 20, 107, 5, 20, 10)$	17.82	16.92	54.59	57.75	15.92	12.39	12.14	14.93	10.18	7.75	7.34	10.26
$(250, 20, 20, 117, 5, 20, 20)$	17.24	16.56	55.39	79.80	23.38	14.26	13.25	24.14	10.05	8.81	8.39	10.34
$(250, 20, 20, 129, 5, 20, 30)$	16.80	15.80	26.21	21.14	7.23	5.51	6.19	7.84	33.93	4.61	4.41	34.20
$(250, 20, 20, 146, 5, 20, 40)$	15.92	15.24	44.60	30.17	26.24	3.91	4.51	17.99	11.38	3.17	3.17	10.63
$(250, 25, 15, 211, 5, 25, 10)$	13.04	12.34	28.86	38.75	15.33	11.67	13.37	18.12	5.86	7.71	6.65	6.44
$(250, 25, 15, 230, 5, 25, 20)$	12.86	11.98	20.21	34.47	8.60	7.12	14.84	19.47	4.19	3.94	3.36	6.81
$(250, 25, 15, 253, 5, 25, 30)$	12.38	11.82	11.36	16.91	5.18	11.13	5.14	5.26	2.80	3.71	3.52	3.06
$(250, 25, 15, 280, 5, 25, 40)$	11.80	11.16	7.56	32.74	3.67	16.32	17.49	4.91	4.03	1.40	1.26	3.55
$(250, 20, 20, 99, 10, 25, 10)$	17.92	17.02	M	M	M	M	M	M	66.94	63.15	62.99	66.33
$(500, 20, 15, 123, 5, 50, 10)$	13.04	12.58	12.60	13.63	7.01	8.08	7.31	7.54	5.48	4.50	4.41	5.86
$(500, 20, 15, 136, 5, 50, 20)$	12.94	12.10	47.16	19.22	25.54	23.49	27.01	15.11	4.86	4.92	3.94	5.24

Table 1: Parameters w^+ and s of the tree-decomposition and runtime (in s) on random partial structured CSPs with mdd for class 1 and mdd_{dyn} for classes 2, 3 and 4.

CSP Instance	n	e	w^+	s	Class 1		Class 2		Class 3		
					B D	A D	B C	A C	A F	E G	B G
qa_5	26	309	25	9	32.01	132.82	2.67	77.82	84.19	80.86	2.62
qcp-10-64-45-QWH-10	100	900	91	82	T	M	69.83	T	T	70.10	69.82
qcp-15-120-278-QWH-15	225	3,150	211	197	T	T	120.69	10.47	10.92	122.26	122.47
qwh-15-106-392-QWH-15	225	3,150	211	197	T	M	0.89	169.49	181.18	0.86	0.89
qwh-15-106-974-QWH-15	225	3,150	211	197	T	94.58	2.55	75.77	79.46	2.67	2.54

Table 2: Parameters w^+ and s of the tree-decomposition and runtime (in s) on some instances from the CP'2005 solver competition with mdd for class 1 and mdd_{dyn} for classes 2 and 3.

- $nv_{sdyn}(G)$: it is similar to nv. We visit first the son cluster where appears the next variable in the heuristic order among the variables of the unvisited son clusters.

Inside a cluster, the heuristic used for choosing the next variable is min domain/degree (static version mdd_s for class 1 and dynamic mdd_{dyn} for the other classes).

Table 1 shows the runtime of BTD with several heuristics of Classes 1, 2, 3 and 4. For Class 5, we cannot get good results and then, the results are not presented. Also it presents the width of the computed tree-decomposition and the maximum size of the separators. Clearly, we observe that Class 1 orders obtain poor results. This behaviour is not surprising since static variable orders are well known to be inefficient compared to dynamic ones. A dynamic strategy allows to make good choices by taking in account the modifications of the problem during search. Thus these choices are more justified than in a static case. That explains the good results of Classes 2 and 3 orders. The results show as well the crucial importance of the root cluster choice since each heuristic of the Classes 2 and 3 fails to solve an average of 4 instances over all instances of all classes because of a bad choice of root cluster. We note that the unsolved instances are not the same for $size$ and $minexp$ heuristics. The memory problems marked by M can be solved by using a *Class 4* order with the sep heuristic for grouping variables (we merge cluster whose intersection is greater than a value s_{max}). Table 1 gives the runtime of BTD for this class with $s_{max} = 5$.

When we analyze the value of the parameter k, we observe that its value is generally limited (between 1 to 6). Nevertheless, for the CSPs $(250, 20, 20, 99, 10, 25, 10)$, the value of k is near 40, but this high value allows to solve these instances.

The heuristics for the Classes 2 and 3 improve very significantly the results obtained. The impact of the dynamicity is obvious. $minexp$ and nv heuristics solve all the instances except one due to a bad root cluster choice, $size$ solve all the instances. Except the unsolved instance, $minexp$ obtains very promising results. The son cluster ordering has a limited effect because the instances considered have a few son clusters reducing the possible choices and so their impact. We can expect a more important improvement for instances with more son clusters. The best results are obtained by $minexp + minexp_{sdyn}$, but $size + minsep_s$ obtains often similar results and succeed in solving all instances in the *Class 4*. The calculus of the expected number of solution assumes that the problem constraints are independent, what is the case for the problems considered here. Thus, $size + minsep$ may outperform $minexp + minexp_{sdyn}$ on real-world problems which have dependent constraints.

When we observe the results in table 1, we see the relevance of extending the dynamic order. Merging clusters with

k less than 6 decreases the maximal size of separators and allows a more dynamic ordering of variables. That leads to an important reduction of the runtime. These experiments highlight the importance of dynamic orders and make us conclude that the Class 4 gives the best variable orders w.r.t CSP solving with a good value of k. Of course, this behaviour has been observed on random instances. The next step of our study consists in assessing the proposed heuristics on benchmark instances of the CP'2005 solver competition. At the beginning of the section, we reminded that structural methods like BTD assume the CSP has interesting topological features. This is not the case for a great part of instances of this competition. Since the space complexity of BTD is in $O(n.s.d^s)$, if the size of the cluster separators of the computed tree-decomposition of CSP is too large, the method needs more than 1GB of memory. Many instances have tree-decompositions with very large separators and clusters. Reducing the required space memory leads to group all the variables in one cluster and so to perform like the FC algorithm. In table 2, which provides results on these instances, we do not present the results for instances with very bad topological features for which BTD has a behavior close to FC one.

On these instances, the $size$ heuristic outperforms $minexp$ except on qcp instance. To resume, we can say that the dynamicity improves significantly the method and the expected number of solutions provides an important improvement on random CSPs, while the $size + minsep$ heuristic outperforms the others on real-world instances.

6 Discussion and Conclusion

In this article, we have studied the CSP solving methods based on tree-decompositions in order to improve their practical interest. This study was done both theoretically and empirically. The analysis of the variable orders allows us to define more dynamic heuristics without loosing the time complexity bound. So, we have defined classes of variable orders which allow a more and more dynamic ordering of variables and preserve the theoretical time complexity bound. This bound has been extended to enforce the dynamic aspect of orders that has an important impact on the efficiency of enumerative methods. Even though these new bounds are theoretically less interesting that the initial, it allows us to define more efficient heuristics which improve significantly the runtime of BTD. This study, which could not be achieved previously, takes now on importance for solving hard instances with suitable structural properties. For example, the structured instances used here are seldom solved by enumerative methods like FC or MAC.

We have compared the classes of variable orders with relevant heuristics w.r.t. CSP solving. This comparison points up the importance of a dynamic variable ordering. Indeed the best results are obtained by *Class 4* orders because they give more freedom to the variable ordering heuristic while their time complexity is $O(exp(w+k+1))$ where k is a constant to parameterize. Note that for the most dynamic class (the Class 5), we get a time complexity in $O(exp(2(w+k+1)-s^-))$. It seems that this bound should be too large to expect a significant practical improvement.

Concerning heuristics, we exploit the notion of expected number of partial solutions in order to guide the traversal of the cluster tree during the solving. Even though the other heuristics presented ($size + minsep$) are less efficient, often they obtain similar results. They are also more general what induces a stabler behaviour. So, on real-world problems with dependent constraints, they may outperform the expected number of solutions based heuristics. Then, for *Class 4*, we aim to improve the criteria used to compute the value of k and to define more general ones by exploiting better the problem features.

This study will be pursued on the Valued CSPs [Schiex *et al.*, 1995] which are well known to be more difficult.

Acknowledgments

This work is supported by a "programme blanc" ANR grant (STAL-DEC-OPT project).

References

[Dechter and Fattah, 2001] R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.

[Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.

[Gottlob *et al.*, 2000] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.

[Gottlob *et al.*, 2002] G. Gottlob, M. Hutle, and F. Wotawa. Combining hypertree, bicomp and hinge decomposition. In *Proceedings of ECAI*, pages 161–165, 2002.

[Jégou and Terrioux, 2003] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.

[Jégou and Terrioux, 2004] P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proceedings of ECAI*, pages 196–200, 2004.

[Jégou *et al.*, 2005] P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of CP*, pages 777–781, 2005.

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

[Schiex *et al.*, 1995] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proceedings of IJCAI*, pages 631–637, 1995.

[Smith, 1994] B. Smith. The Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings of ECAI*, pages 100–104, 1994.

[Tarjan and Yannakakis, 1984] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3):566–579, 1984.

Combining Restarts, Nogoods and Decompositions for Solving CSPs

Philippe Jégou and **Cyril Terrioux** [1]

Abstract. From a theoretical viewpoint, the (tree-)decomposition methods offer a good approach when the (tree)-width of constraint networks (CSPs) is small. In this case, they have often shown their practical interest. However, sometimes, a bad choice for the root cluster (a tree-decomposition is a tree of clusters) may drastically degrade the performance of the solving.

In this paper, we highlight an explanation of this degradation and we propose a solution based on restart techniques. Then, we present a new version of the BTD algorithm (for Backtracking with Tree-Decomposition [8]) integrating restart techniques. From a theoretical viewpoint, we prove that reduced nld-nogood can be safely recorded during the search and that their size is smaller than ones recorded by MAC+RST+NG [9]. We also show how structural (no)goods may be exploited when the search restarts from a new root cluster. Finally, from a practical viewpoint, we show experimentally the benefits of using restart techniques for solving CSPs by decomposition methods.

1 INTRODUCTION

Constraint Satisfaction Problems (CSPs, see [14] for a state of the art) provide an efficient way of formulating problems in computer science, especially in Artificial Intelligence.

Formally, a *constraint satisfaction problem* is a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of n variables, $D = (d_{x_1}, \ldots, d_{x_n})$ is a list of finite domains of values, one per variable, and $C = \{C_1, \ldots, C_e\}$ is a finite set of e constraints. Each constraint C_i is a pair $(S(C_i), R(C_i))$, where $S(C_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ is the *scope* of C_i, and $R(C_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ is its *compatibility relation*. The *arity* of C_i is $|S(C_i)|$. A CSP is called *binary* if all constraints are of arity 2. The structure of a constraint network is represented by a hypergraph (which is a graph in the binary case), called the constraint (hyper)graph, whose vertices correspond to variables and edges to the constraint scopes. In this paper, for sake of simplicity, we only deal with the case of binary CSPs but this work can easily be extended to non-binary CSP by exploiting the 2-section [1] of the constraint hypergraph (also called primal graph), as it will be done for our experiments since we will consider binary and non-binary CSPs. Moreover, without loss of generality, we assume that the network is connected. To simplify the notations, in the sequel, we denote the graph $(X, \{S(C_1), \ldots S(C_e)\})$ by (X, C). An assignment on a subset of X is said to be *consistent* if it does not violate any constraint. Testing whether a CSP has a *solution* (i.e. a consistent assignment on all the variables) is known to be NP-complete. So the time complexity of backtracking algorithms which are usually exploited to solve CSPs, is naturally exponential, at least in $O(e.d^n)$.

[1] Aix-Marseille Université, LSIS UMR 7296, France {philippe.jegou, cyril.terrioux}@lsis.org

Many works have been realized to make the solving more efficient in practice, by using optimized backtracking algorithms, heuristics, constraint learning, non-chronological backtracking, filtering techniques, etc. In order to ensure an efficient solving, most solvers commonly exploit jointly several of these techniques. Moreover, often, they also derive benefit from the use of restart techniques. In particular, restart techniques generally allow to reduce the impact of bad choices performed thanks to heuristics (like the variable ordering heuristic) or of the occurrence of heavy-tailed phenomena. They have been recently introduced in the CSP framework (e.g. in [9]). For efficiency reasons, they are usually exploited with some learning techniques (like recording of nld-nogoods in [9]).

In this paper, we introduce for the first time the restart techniques in the context of decomposition methods for solving CSPs. Decomposition methods (e.g. [4, 8]) solve CSPs by taking into account some particular features of the constraint networks. Often, they rely on the notion of tree-decomposition of graphs [12]. In such a case, their advantage is related to their theoretical complexity, i.e. $d^{w^+ + 1}$ where w^+ is the width of the considered tree-decomposition. Since computing an optimal tree-decomposition is NP-Hard, the used tree-decompositions are generally computed by heuristic methods and so approximate optimal tree-decompositions. When this graph has nice topological properties and thus when w^+ is small, these methods allow to solve large instances, e.g. radio link frequency assignment problems [3]. From a practical viewpoint, they have obtained promising results on such instances. However, their efficiency may drastically be degraded by some bad choices performed by heuristics. To present this issue, we consider here the BTD method [8] which is a reference in the state of the art for this type of approach [11].

For BTD, the considered tree-decomposition and the choice of the root cluster (i.e. the first studied cluster) induce a particular variable ordering. Hence, as it is well known that the variable ordering has a significant impact on the efficiency of the solving, the choice of the root cluster is crucial. In [7], an approach has been proposed to choose a variable ordering with more freedom but its efficiency still depends on the choice of the root cluster. In the next section, we explain why it is difficult to propose a suitable choice for the root cluster. As a consequence, in order to reduce the impact of the root cluster on the practical efficiency, we propose an alternative based on restart techniques. Then, we present a new version of BTD integrating restart techniques. From a theoretical viewpoint, we prove that reduced nld-nogood can be safely recorded during the search and that their size is smaller than ones recorded by MAC+RST+NG [9]. We also show how structural (no)goods can be exploited when the search restarts from a new root cluster. Finally, from a practical viewpoint, we show experimentally the benefits of the use of restart techniques for solving CSPs by decomposition methods.

Section 2 recalls the frame of BTD and describes the BTD-MAC algorithm[2]. Then, section 3 presents the algorithm BTD-MAC+RST. In section 4, we assess the benefits of restarts when solving CSPs thanks to a decomposition-based method and conclude in section 5.

2 THE BTD METHOD

BTD [8] relies on the notion of tree-decomposition of graphs [12].

Definition 1 *A* tree-decomposition *of a graph $G = (X, C)$ is a pair (E, T) with $T = (I, F)$ a tree and $E = \{E_i : i \in I\}$ a family of subsets of X, such that each subset (called cluster) E_i is a node of T and satisfies: (i) $\cup_{i \in I} E_i = X$, (ii) for each edge $\{x, y\} \in C$, there exists $i \in I$ with $\{x, y\} \subseteq E_i$, and (iii) for all $i, j, k \in I$, if k is in a path from i to j in T, then $E_i \cap E_j \subseteq E_k$. The width of a tree-decomposition (E, T) is equal to $max_{i \in I}|E_i| - 1$. The* tree-width w *of G is the minimal width over all the tree-decompositions of G.*

Given a tree-decomposition (E, T) and a root cluster E_r, we denote $Desc(E_j)$ the set of vertices (variables) belonging to the union of the descendants E_k of E_j in the tree rooted in E_j, E_j included. Figure 1(b) presents a tree whose nodes correspond to the maximal cliques of the graph depicted in Figure 1(a). It is a possible tree-decomposition for this graph. So, we get $E_1 = \{x_1, x_2, x_3\}$, $E_2 = \{x_2, x_3, x_4, x_5\}$, $E_3 = \{x_4, x_5, x_6\}$, and $E_4 = \{x_3, x_7, x_8\}$. As the maximum size of clusters is 4, the tree-width of this graph is 3. We have $Desc(E_1) = X$ and $Desc(E_2) = \{x_2, x_3, x_4, x_5, x_6\}$.

Given a compatible cluster ordering $<$ (i.e. an ordering which can be produced by a depth-first traversal of T from the root cluster E_r), BTD achieves a backtrack search by using a variable ordering $\preceq$ (said *compatible*) s.t. $\forall x \in E_i$, $\forall y \in E_j$, with $E_i < E_j$, $x \preceq y$. In other words, the cluster ordering induces a partial ordering on the variables since the variables in E_i are assigned before those in E_j if $E_i < E_j$. For the example of Figure 1, $E_1 < E_2 < E_3 < E_4$ (resp. $x_1 \preceq x_2 \preceq x_3 \preceq \ldots \preceq x_8$) is a possible compatible ordering on E (resp. X). In practice, BTD starts its backtrack search by assigning consistently the variables of the root cluster E_r before exploring a child cluster. When exploring a new cluster E_i, it only assigns the variables which appears in the cluster E_i but not in its parent cluster $E_{p(i)}$, that is all the variables of the cluster E_i except the variables of the separator $E_i \cap E_{p(i)}$[3].

In order to solve each cluster, BTD can exploit any solving algorithm which does not alter the structure. For instance, BTD can rely on the algorithm MAC (for Maintaining Arc-Consistency [15]). During the solving, MAC can make two kinds of decisions: *positive decisions* $x_i = v_i$ which assign the value v_i to the variable x_i (we denote $Pos(\Sigma)$ the set of positive decisions in a sequence of decisions Σ) and *negative decisions* $x_i \neq v_i$ which ensure that x_i cannot be assigned with v_i. Let us consider $\Sigma = \langle \delta_1, \ldots, \delta_i \rangle$ (where each δ_j may be a positive or negative decision) as the current decision sequence. A new positive decision $x_{i+1} = v_{i+1}$ is chosen and an AC filtering is achieved. If no dead-end occurs, the search goes on by choosing a new positive decision. Otherwise, the value v_{i+1} is deleted from the domain $d_{x_{i+1}}$, and an AC filtering is realized. If a dead-end occurs again, we backtrack and change the last positive decision $x_\ell = v_\ell$ to $x_\ell \neq v_\ell$. Regarding BTD-MAC (i.e. BTD relying on MAC for solving each cluster), we can note that the next positive decision necessarily involves a variable of the current cluster E_i and that only the domains of the future variables in $Desc(E_i)$ can be impacted by the AC filtering (since $E_i \cap E_{p(i)}$ is a separator of the constraint graph and all its variables have already been assigned).

When BTD has consistently assigned the variables of a cluster E_i, it then tries to solve each subproblem rooted in each child cluster E_j. More precisely, for a child E_j and a current decision sequence Σ, it attempts to solve the subproblem induced by the variables of $Desc(E_j)$ and the decision set $Pos(\Sigma)[E_i \cap E_j]$ (i.e. the set of positive decisions involving the variables of $E_i \cap E_j$). Once this subproblem solved (by showing that there is a solution or showing that there is none), it records a structural good or nogood. Formally, given a cluster E_i and E_j one of its children, a *structural good* (resp. *nogood*) of E_i with respect to E_j is a consistent assignment A of $E_i \cap E_j$ such that A can (resp. cannot) be consistently extended on $Desc(E_j)$ [8]. In the particular case of BTD-MAC, the consistent assignment of A will be represented by the restriction of the set of positive decisions of Σ on $E_i \cap E_j$, namely $Pos(\Sigma)[E_i \cap E_j]$. These structural (no)goods can be used later in the search in order to avoid exploring a redundant part of the search tree. Indeed, once the current decision sequence Σ contains a good (resp. nogood) of E_i w.r.t. E_j, BTD has already proved previously that the corresponding subproblem induced by $Desc(E_j)$ and $Pos(\Sigma)[E_i \cap E_j]$ has a solution (resp. none) and so does not need to solve it again. In the case of a good, BTD keeps on the search with the next child cluster. In the case of a nogood, it backtracks. For example, let us consider a CSP on 8 variables $x_1, \ldots, x_8$ for which each domain is $\{a, b, c\}$ and whose constraint graph and a possible tree-decomposition are given in Figure 1. Assume that the current consistent decision sequence $\Sigma = \langle x_1 = a, x_2 \neq b, x_2 = c, x_3 = b \rangle$ has been built according to a variable order compatible with the cluster order $E_1 < E_2 < E_3 < E_4$. BTD tries to solve the subproblem rooted in E_2 and once solved, records $\{x_2 = c, x_3 = b\}$ as a structural good or nogood of E_1 w.r.t. E_2. If, later, BTD studies the consistent decision sequence $\langle x_1 \neq a, x_3 = b, x_1 = b, x_2 \neq a, x_2 = c \rangle$, it will keep on its search with the next child cluster of E_1, namely E_4, if $\{x_2 = c, x_3 = b\}$ has been recorded as a good, or backtrack to the last decision in E_1 if $\{x_2 = c, x_3 = b\}$ corresponds to as a nogood.

Algorithm 1 without the lines 21-24 corresponds to the algorithm BTD-MAC. Initially, the current decision sequence Σ and the sets G and N of recorded structural goods and nogoods are empty and the search starts with the variables of the root cluster E_r. Given a current cluster E_i and the current decision sequence Σ, lines 16-27 consist in exploring the cluster E_i by assigning the variables of V_{E_i} (with V_{E_i} the set of unassigned variables of the cluster E_i) like MAC would do while lines 1-14 allow to manage the children of E_i and so to use and record structural (no)goods. BTD-MAC(P,Σ,E_i,V_{E_i},G,N) returns $true$ if it succeeds in extending consistently Σ on $Desc(E_i) \backslash (E_i \backslash V_{E_i})$, $false$ otherwise. It has a time complexity in $O(n.s^2.e.\log(d).d^{w^+ + 2})$ while its space complexity is $O(n.s.d^s)$ with w^+ the width of the used tree-decomposition and s the size of the largest intersection between two clusters.

From a practical viewpoint, generally, BTD efficiently solves CSPs having a small tree-width [6, 7, 8]. However, sometimes, a bad choice for the root cluster may drastically degrade the performance of the solving. The choice of the root cluster is crucial since it impacts on the variable ordering, in particular on the choice of the first variables. Hence, in order to make a smarter choice, we have selected some instances of the CSP 2008 Competition[4] and, for each instance, we run BTD from each cluster of its considered tree-decomposition.

[2] BTD-MAC has never been described before in the literature. The algorithm MAC-BTD evoked in [8] is in fact RFL-BTD, i.e. BTD based on Real Full Look-ahead [10] (see [16] for a comparison between MAC and RFL).

[3] We assume that $E_i \cap E_{p(i)} = \emptyset$ if E_i is the root cluster.

[4] See http://www.cril.univ-artois.fr/CPAI08 for more details.

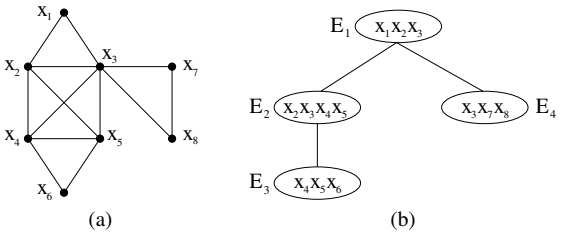


Figure 1. A constraint graph for 8 variables (a) and an optimal tree-decomposition (b).

We have first observed that for a same instance, the runtimes may differ from several orders of magnitude according the chosen root cluster. For instance, for the scen11-f12 instance (which is the easiest instance of the scen11 family), BTD succeeds in proving the inconsistency for only 75 choices of root cluster among the 301 possible choices. Secondly, we have noted that solving some clusters (not necessarily the root cluster) and their corresponding subproblems is more expensive for some choice of the root cluster than for another. This is explained by the choice of the root cluster which induces some particular ordering on the clusters and the variables. In particular, since for a cluster E_i, BTD only considers the variables of $E_i \backslash (E_i \cap E_{p(i)})$, it does not handle the same variable set for E_i depending on the chosen root cluster. Unfortunately, it seems to be utopian to propose a choice for the root cluster based only on features of the instance to solve because this choice is too strongly related to the solving efficiency. In [7], an approach has been proposed to choose a variable ordering with more freedom but its efficiency still depends on the choice of the root cluster. So, an alternative to limit the impact of the choice of the cluster is required. In section 3, we propose a possible one consisting in exploiting restart techniques.

3 EXPLOITING RESTARTS WITHIN BTD

It is well known that any method exploiting restart techniques must as much as possible avoid exploring the same part of the search space several times and that randomization and learning are two possible ways to reach this aim. Regarding the learning, BTD already exploits structural (no)goods. However, depending on when the restart occurs, we have no warranty that a (no)good has been recorded yet. Hence, another form of learning is required to ensure a good practical efficiency. Here, we consider the reduced nld-nogoods (for negative last decision nogoods) whose practical interest has been highlighted in the MAC+RST+NG algorithm [9]. We first recall the notion of nogood in the case of MAC:

Definition 2 ([9]) *Given a CSP $P = (X, D, C)$ and a set of decisions Δ, $P_{|\Delta}$ is the CSP (X, D', C) with $D' = (d'_{x_1}, \ldots, d'_{x_n})$ such that for any positive decision $x_i = v_i$, $d'_{x_i} = \{v_i\}$ and for any negative decision $x_i \neq v_i$, $d'_{x_i} = d_{x_i} \backslash \{v_i\}$. Δ is a* nogood *of P if $P_{|\Delta}$ is inconsistent.*

In the following, we assume that for any variable x_i and value v_i, the positive decision $x_i = v_i$ is considered before the decision $x_i \neq v_i$.

Proposition 1 ([9]) *Let $\Sigma = \langle \delta_1, \ldots, \delta_k \rangle$ be the sequence of decisions taking along the branch of the search tree when solving a CSP P. For any subsequence $\Sigma' = \langle \delta_1, \ldots, \delta_\ell \rangle$ of Σ s.t. δ_ℓ is a negative decision, the set $Pos(\Sigma') \cup \{\neg\delta_\ell\}$ is a nogood (called a* reduced nld-nogood*) of P with $\neg\delta_\ell$ the positive decision corresponding to δ_ℓ.*

In other words, given a sequence Σ of decisions taking along the branch of a search tree, each reduced nld-nogood characterizes a visited inconsistent part of this search tree. When a restart occurs, an algorithm like MAC+RST+NG can record several new reduced nld-nogoods and exploit them later to prevent from exploring again an already visited part of the search tree. These nld-nogoods can be efficiently computed and stored as a global constraint with an efficient specific propagator for enforcing AC [9].

The use of learning in BTD may endanger its correctness as soon as we add to the initial problem a constraint whose scope is not included in a cluster. So recording reduced nld-nogoods in a global constraint involving all the variables like proposed in [9] is impossible. However, by exploiting the features of a compatible variable ordering, Property 2 shows that this global constraint can be safely decomposed in a global constraint per cluster E_i.

Proposition 2 *Let $\Sigma = \langle \delta_1, \ldots, \delta_k \rangle$ be the sequence of decisions taking along the branch of the search tree when solving a CSP P by exploiting a tree-decomposition (E, T) and a compatible variable ordering. Let $\Sigma[E_i]$ be the subsequence built by considering only the decisions of Σ involving the variables of E_i. For any prefix subsequence $\Sigma'_{E_i} = \langle \delta_{i_1}, \ldots, \delta_{i_\ell} \rangle$ of $\Sigma[E_i]$ s.t. δ_{i_ℓ} is a negative decision, and every variable in $E_i \cap E_{p(i)}$ appears in a decision in $Pos(\Sigma'_{E_i})$, the set $Pos(\Sigma'_{E_i}) \cup \{\neg\delta_{i_\ell}\}$ is a reduced nld-nogood of P.*

Proof: Let P_{E_i} be the subproblem induced by the variables of $Desc(E_i)$ and Δ_{E_i} the set of the decisions of $Pos(\Sigma_{E_i})$ related to the variables of $E_i \cap E_{p(i)}$. As $E_i \cap E_{p(i)}$ is a separator of the constraint graph, $P_{E_i|\Delta_{E_i}}$ is independent from the remaining part of the problem P. Let us consider $\Sigma[E_i]$ the maximal subsequence of Σ which only contains decisions involving variables of E_i. According to Proposition 1 applied to $\Sigma[E_i]$ and $P_{E_i|\Delta_{E_i}}$, $Pos(\Sigma'_{E_i}) \cup \{\neg\delta_{i_\ell}\}$ is necessarily a reduced nld-nogood. □

It ensues that we can bound the size of produced nogoods and compare them with those produced by Proposition 1:

Corollary 1 *Given a tree-decomposition of width w^+, the size of reduced nld-nogood produced by proposition 2 is at most $w^+ + 1$.*

Corollary 2 *Under the same assumptions as Proposition 2, for any reduced nld-nogood Δ produced by Proposition 1, there is at least one reduced nld-nogood Δ' produced by Proposition 2 s.t. $\Delta' \subseteq \Delta$.*

BTD already exploits a particular form of learning by recording structural (no)goods. Any structural (no)good of a cluster E_i w.r.t. to a child cluster E_j is by definition oriented from E_i to E_j. This orientation is directly induced by the choice of the root cluster. When a restart occurs, BTD may choose a different cluster as root cluster. If so, we have to consider structural (no)goods with different orientations. Proposition 3 states how these structural (no)goods can be safely exploited when BTD uses the restart technique.

Proposition 3 *A structural good of E_i w.r.t. E_j can only be used if the choice of the current root cluster induces that E_j is a child cluster of E_i. A structural nogood of E_i w.r.t. E_j can be used regardless the choice of the root cluster.*

Proof: Let us consider a good Δ of E_i w.r.t. E_j produced for a root cluster E_r. By definition of structural goods, the subproblem $P_{E_j|\Delta}$ has a solution and its definition only depends on Δ and the fact that E_j is a child cluster of E_i. So, for any choice of the root cluster s.t. E_j is a child cluster of E_i, Δ will be a structural good of E_i w.r.t. E_j and can be used to prune safely redundant

part of the search. Regarding structural nogoods, any structural nogood Δ of E_i w.r.t. E_j is a nogood and so any decision sequence Σ s.t. $\Delta \subseteq Pos(\Sigma)$ cannot be extended to a solution, independently from the choice of the root cluster. Hence, structural nogoods can be used regardless the choice of the root cluster. □

It follows that unlike the nogoods, for the goods, the orientation is required. So, it could be better to call them *oriented structural goods*.

Algorithm 2 describes the algorithm BTD-MAC+RST which exploits restart techniques jointly with recording reduced nld-nogoods and structural (no)goods. Exploiting the restart techniques can be seen as choosing a root cluster (line 3) and running a new instance of BTD-MAC+NG (line 4) at each restart until the problem is solved by proving there is a solution or none. Algorithm 1 presents the algorithm BTD-MAC+NG. Like BTD-MAC, given a current cluster E_i and the current decision sequence Σ, BTD-MAC+NG explores the cluster E_i (lines 16-27) by assigning the variables of V_{E_i} (with V_{E_i} the set of unassigned variables of E_i). When E_i is consistently assigned, it manages the children of E_i and so uses and records structural (no)goods (lines 1-14). The used structural (no)goods may have been recorded during the current call to BTD-MAC or during a previous one. Indeed, if the first call of BTD-MAC+NG is achieved with empty sets G and N of structural goods and nogoods, G and N are not reset at each restart. Note that their uses (lines 7-8) are performed according to Proposition 3. Then, unlike BTD-MAC, BTD-MAC+NG may stop its search as soon as a restart condition is reached (line 21). If so, it records reduced nld-nogoods w.r.t. the decision sequence Σ restricted to the decisions involving variables of E_i (line 22) according to Proposition 2. We consider that a global constraint is associated to each cluster E_i to handle the nld-nogoods recorded w.r.t. E_i and that their use is performed via a specific propagator when the arc-consistency is enforced (lines 19 and 25) like in [9]. The restart condition may involve some global parameters (e.g. the number of backtracks achieved since the begin of the current call to BTD-MAC+NG), some local ones (e.g. the number of backtracks performed in the current cluster or the number of recorded structural (no)goods) or a combination of these two approaches.

BTD-MAC+NG(P,Σ,E_i,V_{E_i},G,N) returns $true$ if it succeeds in extending consistently Σ on $Desc(E_i)\backslash(E_i\backslash V_{E_i})$, $false$ if it proves that Σ cannot be consistently extended on $Desc(E_i)\backslash(E_i\backslash V_{E_i})$ or $unknown$ if a restart occurs. BTD-MAC+RST(P) returns $true$ if P has at least a solution, $false$ otherwise.

Theorem 1 *BTD-MAC+RST is sound, complete and terminates.*

Proof: BTD-MAC+NG differs from BTD-MAC by exploiting restart techniques, recording reduced nld-nogoods and starting its search with sets G and N which are not necessarily empty. When a restart occurs, the search is stopped and reduced nld-nogoods are safely recorded from Proposition 2. Regarding structural (no)goods, N and G only contain valid structural (no)goods and their uses (lines 7-8) are safe according to Proposition 3. So, as BTD-MAC is sound and terminates and as these properties are not endangered by the differences between BTD-MAC and BTD-MAC+NG, it is the same for BTD-MAC+NG. Then, as BTD-MAC is complete, BTD-MAC+NG is complete under the condition that no restart occurs. Moreover, restarts stop the search without changing the fact that if a solution exists in the part of the search space visited by BTD-MAC+NG, BTD-MAC+NG would find it. As BTD-MAC+RST only performs several calls to BTD-MAC+NG, it is sound. For the completeness, if the call to BTD-MAC+NG is not stopped by a restart (what is necessarily the case of the last call to BTD-MAC+NG if BTD-MAC+RST terminates), the completeness of BTD-MAC+NG implies one of BTD-MAC+RST. Furthermore, recording reduced nld-nogoods at each restart prevents from exploring a part of the search space already explored by a previous call to BTD-MAC+NG. It ensues that, over successive calls to BTD-MAC+NG, one has to explore a more and more reduced part of the search space. Hence, the termination and completeness of BTD-MAC+RST are ensured by the unlimited nogood recording achieved by the different calls to BTD-MAC+NG and by the termination and completeness of BTD-MAC+NG. □

Algorithm 1: BTD-MAC+NG (**InOut**: $P = (X, D, C)$: CSP; **In**: Σ: sequence of decisions, E_i: Cluster, V_{E_i}: set of variables; **InOut**: G: set of goods, N: set of nogoods)

```
1  if V_{E_i} = ∅ then
2      result ← true
3      S ← Sons(E_i)
4      while result = true and S ≠ ∅ do
5          Choose a cluster E_j ∈ S
6          S ← S\{E_j}
7          if Pos(Σ)[E_i ∩ E_j] is a nogood in N then result ← false
8
9          else if Pos(Σ)[E_i ∩ E_j] is not a good of E_i w.r.t. E_j in G then
10             result ← BTD-MAC+NG(P,Σ,E_j,E_j\(E_i ∩ E_j),G,N)
11             if result = true then
12                 Record Pos(Σ)[E_i ∩ E_j] as good of E_i w.r.t. E_j in G
13             else if result = false then
14                 Record Pos(Σ)[E_i ∩ E_j] as nogood of E_i w.r.t. E_j in N
15     return result
16 else
17     Choose a variable x ∈ V_{E_i}
18     Choose a value v ∈ d_x
19     d_x ← d_x\{v}
20     if AC (P,Σ ∪ ⟨x = v⟩) ∧ BTD-MAC+NG(P, Σ ∪ ⟨x = v⟩, E_i,
       V_{E_i}\{x}, G, N)= true then return true
21
22     else
23         if must restart then
24             Record nld-nogoods w.r.t. the decision sequence Σ[E_i]
25             return unknown
26         else
27             if AC (P,Σ ∪ ⟨x ≠ v⟩) then
28                 return BTD-MAC+NG(P,Σ ∪ ⟨x ≠ v⟩,E_i,V_{E_i},G,N)
29             else return false
```

Algorithm 2: BTD-MAC+RST (**In**: $P = (X, D, C)$: CSP)

```
1 G ← ∅; N ← ∅
2 repeat
3     Choose a cluster E_r as root cluster
4     result ← BTD-MAC+NG (P,∅,E_r,E_r,G,N)
5 until result ≠ unknown
6 return result
```

Theorem 2 *BTD-MAC+RST has a time complexity in $O(R.((n.s^2.e.\log(d) + w^+.N).d^{w^+ +2} + n.(w^+)^2.d))$ and a space complexity in $O(n.s.d^s + w^+.(d + N))$ with w^+ the width of the considered tree-decomposition, s the size of the largest intersection $E_i \cap E_j$, R the number of restarts and N the number of recorded reduced nld-nogoods.*

Proof: BTD-MAC without nld-nogoods has a time complexity in $O(n.s^2.e.\log(d).d^{w^+ +2})$. According to Propositions 4 and 5 of [9], storing and managing nld-nogoods of size at most n can be achieved respectively in $O(n^2.d)$ and $O(n.N)$. As, according to Corollary 1, the size of nld-nogoods is at most $w^+ + 1$, this two operations can be achieved respectively in $O((w^+)^2.d)$ and $O(w^+.N)$. BTD-MAC+RST makes at most R calls to BTD-MAC. So we obtain a time complexity for BTD-MAC+RST in $O(R.((n.s^2.e.\log(d) + w^+.N).d^{w^+ +2} + n.(w^+)^2.d))$.

By exploiting the data structure proposed in [9], the worst case space complexity for storing reduced nld-nogoods is $O(w^+.(d+N))$ since according to Corollary 1, BTD-MAC+RST records N nogoods of size at most $w^+ + 1$. Regarding the storage of structural (no)goods, BTD-MAC+RST has the same space complexity as BTD, namely $O(n.s.d^s)$. So, its whole space complexity is $O(n.s.d^s + w^+.(d+N))$. □

If BTD-MAC+RST exploits a geometric restart policy [17] based on the number of allowed backtracks (i.e. a restart occurs as soon as the number of performed backtracks exceeds the number of allowed backtracks which is initially set to n_0 and increased by a factor r at each restart), we can bound the number of restarts:

Proposition 4 *Given a geometric policy based on the number of backtracks with an initial number n_0 of allowed backtracks and a ratio r, the number of restarts R is bounded by* $\left\lceil \frac{\log(n)+(w^+ +1).\log(d)-\log(n_0)}{\log(r)} \right\rceil$.

4 EXPERIMENTATIONS

In this section, we assess the benefits of restarts when solving CSPs thanks to a decomposition-based method. With this aim in view, we compare BTD-MAC+RST with BTD-MAC and MAC+RST+NG on 647 instances (of arbitrary arity) among the instances of the CSP 2008 Competition. The selected instances are ones which have suitable tree-decompositions (i.e. a ratio n/w^+ at least equal to 2). These tree-decompositions are computed thanks to Min-Fill [13] which is considered as the best heuristic of the state of the art [5]. The runtime of BTD-MAC(+RST) includes the time required to compute the tree-decomposition. All the methods exploit the *dom/wdeg* variable heuristic [2]. We have tried several heuristics for the choice of the root cluster. We present here the best ones:

- RW: we choose the cluster maximizing the sum of weights of constraints whose scope intersects the cluster (the weights are those of dom/wdeg). This heuristic is also one exploited by BTD-MAC.
- RA: we choose alternatively the cluster containing the next variable according to dom/wdeg applied on all the variables and maximizing sum of weights of constraints whose scope intersects the cluster or a cluster according to the decreasing ratio number of constraints over size of the cluster minus one.

Both heuristics RW and RA aim to follow the first-fail principle. The second case of RA brings some diversity in the search. The used restart policies rely on the number of allowed backtracks. The presented values below are ones providing the best results among the tested values. More precisely, for MAC+RST+NG, we exploit a geometric policy where the initial number of allowed backtracks is 100 while the increasing factor is 1.1. BTD-MAC+RST with RW uses a geometric policy with a ratio 1.1 and initially 50 allowed backtracks. For RA, we apply a geometric policy with a ratio 1.1 and initially 75 allowed backtracks when the cluster is chosen according to the first case. In the second case, we use a constant number of allowed backtracks set to 75. All the implementations are written in C++. The experimentations are performed on a linux-based PC with an Intel Pentium IV 3.2 GHz and 1 GB of memory. The runtime limit is set to 1,200 s (except for Table 1).

Figure 2 presents the cumulative number of solved instances for each considered algorithm. First, we can note that the two heuristics RW and RA globally lead to a similar behavior for BTD-MAC+RST. Then it appears clearly that BTD-MAC+RST solves more instances

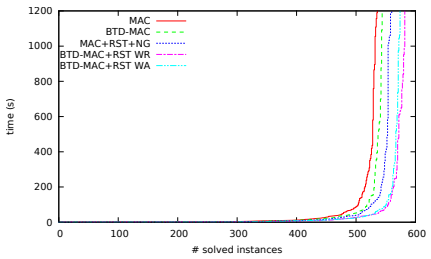


Figure 2. The cumulative number of solved instances per algorithm.

Table 1. Runtime in s (without timeout) for the scen11 instances.

Instance	MAC+RST+NG	BTD-MAC+RST
scen11-f12	0.51	0.30
scen11-f11	0.50	0.30
scen11-f10	0.65	0.35
scen11-f9	1.32	1.54
scen11-f8	1.60	1.78
scen11-f7	12.93	6.81
scen11-f6	20.23	9.86
scen11-f5	102	45.72
scen11-f4	397	202
scen11-f3	1,277	609
scen11-f2	3,813	1,911
scen11-f1	9,937	5,014

than any other algorithm. For instance, BTD-MAC+RST solves 582 instances in 15,863 s with RW (resp. 574 instances in 13,280 s for RA) while MAC+RST+NG only solves 560 instances in 16,943 s. Without restart techniques, the number of solved instances is still smaller with 536 and 544 instances in 18,063 s and 13,256 s for MAC and BTD-MAC respectively.

In order to better analyze the behavior of the different algorithms, we now consider the results obtained per family of instances[5]. Table 2 provides the number of solved instances and the cumulative runtime for each considered algorithm while Table 3 gives the runtime for instances which are solved by all the algorithms. First, we can note that, for some kinds of instances, like graph coloring, the use of restart techniques does not allow to improve the efficiency of BTD-MAC+RST w.r.t. to MAC+RST+NG or BTD-MAC. On the other hand, for the other considered families, we can observe that BTD-MAC+RST provides interesting results. These good results are sometimes due only to the tree-decomposition (e.g. for the families dubois or haystacks) since they are close to ones of BTD-MAC. Likewise, in some cases, they mainly result from the use of restart techniques (e.g. for the families jobshop or geom) and they are then close to ones obtained by MAC+RST+NG. Finally, in other cases, BTD-MAC+RST derives fully benefit of both the tree-decomposition and the restart techniques (e.g. for the families renault, superjobshop or scen11). In such a case, it clearly outperforms the three other algorithms. For example, it is twice faster than MAC+RST+NG for solving the instances of the scen11 family, which contains the more difficult RLFAP instances [3]. Table 1 presents the runtime of MAC+RST+NG and BTD-MAC+RST for these instances. We can remark that BTD-MAC solves only the three easiest instances. This is explained by bad choices for the root cluster. It turns that, for all the instances of this

[5] Note that we do not take into account all the instances of a given family, but only ones having a suitable tree-decomposition.

Table 2. The number of solved instances and the cumulative runtime in s for each considered algorithm.

Family	#inst.	MAC		BTD-MAC		MAC+RST+NG		BTD-MAC+RST RW		BTD-MAC+RST RA	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
dubois	13	5	2,232	13	0.03	5	2,275	13	0.04	13	0.05
geom	83	83	415	83	819	83	479	83	468	83	460
graphColoring	39	29	1,989	33	1,291	29	2,783	34	2,825	33	2,769
haystacks	46	2	5.82	8	169	2	4.43	8	172	8	172
jobshop	46	37	617	35	469	46	14.87	46	13.15	46	10.93
renault	50	50	23.89	50	86.81	50	24.30	50	22.96	50	24.73
pret	8	4	250	8	0.05	4	552	8	0.06	8	0.05
scens11	12	8	1,632	3	1.25	9	537	10	878	10	882
Super-jobShop	46	19	1,648	21	1,179	33	2,315	34	1,553	27	449
travellingSalesman-20	15	15	191	15	229	15	214	15	346	15	294

Table 3. The cumulative runtime in s for each considered algorithm for instances solved by all the algorithms.

Family	#inst.	MAC	BTD-MAC	MAC+RST+NG	BTD-MAC+RST RW	BTD-MAC+RST RA
dubois	5 / 13	2,232	0.01	2,275	0.01	0.01
graphColoring	27 / 39	951	1,051	1,308	846	1,277
haystacks	2 / 46	5.82	0	4.43	0	0.01
jobshop	33 / 46	392	468	5.63	5.10	4.48
pret	4 / 8	250	0.01	552	0.02	0
rlfapScens11	3 / 12	2.75	1.25	1.66	0.95	1.10
Super-jobShop	16 / 46	1,275	830	14.83	9.60	16.04

family, most choices for the root cluster lead to spend a lot of time to solve some subproblems. So, restart techniques are here very helpful.

Finally, we have observed that BTD-MAC+RST is generally more efficient on inconsistent instances than MAC+RST+NG. For example, it requires 4,260 s to solve the inconsistent instances which are solved by all the algorithms while MAC+RST+NG needs 7,105 s. Such a phenomenon is partially explained by the use of the tree-decomposition. Indeed, if BTD-MAC+RST explores an inconsistent cluster at the beginning of the search, it may quickly prove the inconsistency of the problem.

5 CONCLUSION

In this paper, we have firstly presented the integration of MAC in BTD. We have then shown how it is possible to enhance the decomposition-based methods with the integration of the principle of restarts. This has led us to significantly extend the BTD method. We have first described how classic nogoods can be incorporated into a decomposition-based method while preserving the structure induced by a considered decomposition. Next we have introduced the concept of *oriented structural good*. Indeed, if the structural nogoods can be used directly by BTD using restarts, the goods must verify certain properties on the order of exploration of a tree-decomposition, and then, the notion of oriented structural good becomes necessary. In the last part of this paper, the experimentations show clearly the practical interest of exploiting restarts in decomposition-based methods. It effectively overcomes the problem induced by the order of exploration of clusters which harms very often and significantly to their practical effectiveness. These results also show that adding restarts to BTD can significantly outperform the MAC+RST+NG method when the topology of the network constraints has a suitable width.

To extend this work, it would be interesting to define new restart policies specific to the case of the decompositions (e.g. by considering local and/or global policies). Moreover, we can propose smarter choices for the root cluster by exploiting specific information (e.g. the number of (no)goods). Finally, this approach could be applied at a meta level, for instance, to address the problem of choosing a suitable tree-decomposition.

ACKNOWLEDGEMENTS

This work was supported by the French National Research Agency under grant TUPLES (ANR-2010-BLAN-0210).

REFERENCES

[1] C. Berge, *Graphs and Hypergraphs*, Elsevier, 1973.
[2] F. Boussemart, F. Hemery, C. Lecoutre, and L. Saïs, 'Boosting systematic search by weighting constraints', in *ECAI*, pp. 146–150, (2004).
[3] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners, 'Radio Link Frequency Assignment', *Constraints*, **4**, 79–89, (1999).
[4] R. Dechter and J. Pearl, 'Tree-Clustering for Constraint Networks', *Artificial Intelligence*, **38**, 353–366, (1989).
[5] P. Jégou, S. N. Ndiaye, and C. Terrioux, 'Computing and exploiting tree-decompositions for solving constraint networks', in *CP*, pp. 777–781, (2005).
[6] P. Jégou, S.N. Ndiaye, and C. Terrioux, ''Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs', in *IJCAI*, pp. 112–117, (2007).
[7] P. Jégou, S.N. Ndiaye, and C. Terrioux, 'Dynamic Management of Heuristics for Solving Structured CSPs', in *CP*, pp. 364–378, (2007).
[8] P. Jégou and C. Terrioux, 'Hybrid backtracking bounded by tree-decomposition of constraint networks', *AIJ*, **146**, 43–75, (2003).
[9] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal, 'Recording and Minimizing Nogoods from Restarts', *JSAT*, **1**(3-4), 147–167, (2007).
[10] B. Nadel, *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*, 287–342, Search in Artificial Intelligence, 1988.
[11] J. Petke, *On the bridge between Constraint Satisfaction and Boolean Satisfiability*, Ph.D. dissertation, University of Oxford, 2012.
[12] N. Robertson and P.D. Seymour, 'Graph minors II: Algorithmic aspects of treewidth', *Algorithms*, **7**, 309–322, (1986).
[13] D. J. Rose, 'Triangulated Graphs and the Elimination Process', *Journal of Mathematical Analysis and Application*, **32**, 597–609, (1970).
[14] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, Elsevier, 2006.
[15] D. Sabin and E. Freuder, 'Contradicting Conventional Wisdom in Constraint Satisfaction', in *ECAI*, pp. 125–129, (1994).
[16] D. Sabin and E. Freuder, 'Understanding and Improving the MAC Algorithm', in *CP*, pp. 167–181, (1997).
[17] T. Walsh, 'Search in a small world', in *IJCAI*, pp. 1172–1177, (1999).

Bounded Backtracking for the Valued Constraint Satisfaction Problems

Cyril Terrioux and Philippe Jégou

LSIS - Université d'Aix-Marseille 3
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20, France
{cyril.terrioux,philippe.jegou}@univ.u-3mrs.fr

Abstract. We propose a new method for solving Valued Constraint Satisfaction Problems based both on backtracking techniques - branch and bound - and the notion of tree-decomposition of valued constraint networks. This mixed method aims to benefit from the practical efficiency of enumerative algorithms while providing a warranty of a bounded time complexity. Indeed the time complexity of our method is $O(d^{w^+ + 1})$ with w^+ an approximation of the tree-width of the constraint network and d the maximum size of domains.
Such a complexity is obtained by exploiting optimal bounds on the subproblems defined from the tree-decomposition. These bounds associated to some partial assignments are called "*structural valued goods*". Recording and exploiting these goods may allow our method to save some time and space with respect to ones required by classical dynamic programming methods. Finally, this method is a natural extension of the BTD algorithm [1] proposed in the classical CSP framework.

1 Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. In particular, many academic or real problems can be formulated in this framework which allows the expression of NP-complete problems. However, in this formalism, we can't express some notions like possibility or preference because the constraints are either satisfied or violated. In other words, there is no graduation in violation. To avoid this drawback, many extensions of the CSP formalism have been proposed (for instance [2,3,4]). In this paper, we focus on the valued CSP formalism (VCSP [4]) which allows the violations of some constraints by associating a cost (called a *valuation*) to each violated constraint. Solving the problem then consists in finding a complete assignment which optimizes a given criterion about the cost of constraint violations. Generally, we are interested by finding a complete assignment which minimizes the cost of all the violations. So, thanks to the VCSP framework, we can express optimization problems.

The basic method for solving VCSP is the Branch and Bound algorithm. Many improvements have been proposed from the CSP framework [4,5,6,7,8].

F. Rossi (Ed.): CP 2003, LNCS 2833, pp. 709–723, 2003.

On the other hand, some methods based on dynamic programming ([9,10]) often provide good results on such problems.

In this article, we propose a new enumerative method for solving VCSPs. This method, called BTD$_{val}$, is a natural generalization of the BTD method [1] defined in the classical CSP framework. Such a generalization requires the extension of the theoretical frame used for BTD and the classical CSPs. Nevertheless, like BTD, BTD$_{val}$ relies on backtracking techniques (branch and bound) and the notion of tree-decomposition of valued constraint graphs. Such an hybrid method aims to benefit from the advantage of the two approaches, namely the practical efficiency of enumerative algorithms and the time complexity bounds of structural decomposition methods. Thanks to the tree-decomposition notion, BTD$_{val}$ divides the initial problem into several subproblems. Then, it solves each subproblem and records the optimal valuation of each subproblem. These optimal valuations associated with some assignments are called *structural valued goods*. Structural valued goods are then exploited in order to solve each subproblem only once, what allows BTD$_{val}$ to provide time complexity bounds better than ones of classical enumerative methods. Indeed, the time complexity of BTD$_{val}$ is $O(ns^2m\log(d).d^{w^+ +1})$ while the space complexity is $O(nsd^s)$ with $w^+ + 1$ an approximation of the tree-width of the constraint graph, s the size of the biggest minimal separator, n the number of variables and d the size of the largest domain. These bounds only depend on the used tree-decomposition (i.e. on some structural parameters). In [1], experimental results show that on classical CSPs, BTD clearly outperforms an approach founded on dynamic programming like Tree-Clustering [11,12]. So, for VCSPs, we can hope that this behaviour will be confirmed in practice.

The paper is organized as follows. Section 2 introduces the main definitions about the VCSP formalism. Section 3 is devoted to the tree-decomposition notion. Then, section 4 describes the method we propose and present some theoretical results. Finally, in section 5, we discuss about some related works, before concluding in section 6.

2 Valued CSPs

A *constraint satisfaction problem* (CSP) is defined by a quadruplet (X, D, C, R). X is a set $\{x_1, \ldots, x_n\}$ of n variables. Each variable x_i takes its values in the finite domain d_{x_i} from D. Variables are subject to constraints from C. Each constraint c is defined as a set $\{x_{c_1}, \ldots, x_{c_k}\}$ of variables. A relation r_c (from R) is associated with each constraint c such that r_c represents the set of allowed tuples over $d_{x_{c_1}} \times \cdots \times d_{x_{c_k}}$. Given $Y \subseteq X$ such that $Y = \{x_1, \ldots, x_k\}$, an *assignment* of variables from Y is a tuple $\mathcal{A} = (v_1, \ldots, v_k)$ from $d_{x_1} \times \cdots \times d_{x_k}$. A constraint c is said *satisfied* by $\mathcal{A}$ if $c \subseteq Y, (v_1, \ldots, v_k)[c] \in r_c$, *violated* otherwise. We note the assignment $(v_1, \ldots, v_k)$ in the more meaningful form $(x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k)$.

Definition 1 ([4]) *A **valuation structure** is a 5-tuple $(E, \preceq, \oplus, \bot, \top)$ with E a set of valuations which is totally ordered by $\preceq$ with a minimum element*

noted $\bot$ *and a maximum element noted* $\top$. $\oplus$ *is a monotonous, commutative, associative closed binary operation on* E *such that* $\bot$ *is an identity element and* $\top$ *an absorbing element.*

The elements of E express different levels of violation. $\bot$ characterizes the satisfaction of a constraint and $\top$ an unacceptable violation. $\oplus$ allows to combine (aggregate) several valuations. Note that, in some cases, it can have some additional properties like idempotency or strict monotonicity. Thanks to the valuation structure, one can formally define the notion of valued CSP [4]:

Definition 2 *A* ***valued CSP*** *(VCSP)* $\mathcal{P} = (X, D, C, R, S, \phi)$ *consists of a classical CSP* (X, D, C, R) *with a valuation structure* $S = (E, \preceq, \oplus, \bot, \top)$ *and an application* ϕ *from* C *to* E *which associates a valuation to each constraint of* C. *A VCSP is called* ***binary*** *if each constraint of* C *involves at most two variables.*

The valuation of an assignment $\mathcal{A}$ on X is obtained by aggregating the valuations of the constraints violated by $\mathcal{A}$:

Definition 3 *Let* $\mathcal{P}$ *be a VCSP and* $\mathcal{A}$ *an assignment on* X. *The* ***valuation*** *of* $\mathcal{A}$ *with respect to* $\mathcal{P}$ *is defined by* $\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{c \in C | \mathcal{A} \text{ violates } c} \phi(c)$.

Given an instance $\mathcal{P}$, the VCSP problem consists in finding an assignment on X with a minimum valuation according to $\preceq$. This optimal valuation is called the *VCSP valuation* and is denoted $\alpha^*_{\mathcal{P}}$. Determining the valuation of a VCSP is an NP-hard problem. For instance, let us consider the VCSP whose constraint graph is presented in figure 1(a). We suppose that each domain d_x is equal to $\{1, 2, 3\}$ and each constraint c_{xy} means "$x < y$" if the letter represented by x precedes one represented by y in the alphabetical order (for example c_{AB} represents the constraint $A < B$). We exploit the valuation structure $S = (\overline{\mathbb{N}}, <, +, 0, +\infty)$. For each constraint c, the associated valuation is 1. For this VCSP, we obtain $\alpha^*_{\mathcal{P}} = 2$. $(A \leftarrow 1, B \leftarrow 1, C \leftarrow 2, D \leftarrow 2, E \leftarrow 3, F \leftarrow 2, G \leftarrow 2, H \leftarrow 3, I \leftarrow 3, J \leftarrow 3)$ is the best assignment. It violates the constraints c_{AB} and c_{CF}. The assignment valuation notion can be extended to partial assignments:

Definition 4 *Let* $\mathcal{P}$ *be a VCSP and* $\mathcal{A}$ *an assignment on* $Y \subset X$. *The* ***local valuation*** *of* $\mathcal{A}$ *with respect to* $\mathcal{P}$ *is defined by* $v_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\substack{c \in C | c \subseteq Y \text{ and} \\ \mathcal{A} \text{ violates } c}} \phi(c)$.

The following property establishes the link between the valuation of a complete assignment and the local valuation:

Property 1 *Let* $\mathcal{P}$ *be a VCSP,* $\mathcal{A}$ *an assignment on* X *and* $\mathcal{B} \subseteq \mathcal{A}$. $v_{\mathcal{P}}(\mathcal{B}) \preceq v_{\mathcal{P}}(\mathcal{A}) = \mathcal{V}_{\mathcal{P}}(\mathcal{A})$.

So the local valuation can provide a lower bound of the global valuation. The main interest of the local valuation consists in its computation which can be achieved incrementally.

The basic method for solving VCSPs is the branch and bound algorithm (noted BB). This enumerative method exploits the local valuation of the current assignment as a lower bound and the valuation of the best known solution as a upper bound. If the lower bound doesn't exceed the upper one, it extends the current assignment by assigning a new variable. Otherwise, it backtracks to the last assigned variable and then it tries to assign a new value to this variable. If all the values have been tried, it backtracks again. Many improved methods have been proposed from the classical CSP framework like valued Forward-Checking (noted vFC [4]), Nogood Recording [5], ... The use of the arc-consistency notion has been studied too ([6,7,8]). On the other hand, some methods based on dynamic programming, like the Russian Dolls Search (noted RDS) or the structural method proposed by Koster [10], often provide good results. These methods divide the problem into different subproblems and solve the initial problem by exploiting some informations recorded during the resolution of each subproblem.

3 Tree-Decomposition

The only guarantees which can exist in terms of theoretical complexity before solving a problem are offered by structural decomposition methods. These methods proceed by isolating the parts intrinsically exponential (i.e. intractable in polynomial theoretical time) to induce a second step which guarantees a polynomial time of resolution. These methods generally exploit topological properties of the constraint graph and are based on the notion of tree-decomposition of graphs as defined below by Robertson and Seymour [13].

Definition 5 ([13]) *Let $G = (X, E)$ be a graph. A **tree-decomposition** of G is a pair $(\mathcal{C}, \mathcal{T})$ with $\mathcal{T} = (I, F)$ a tree and $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ a family of subsets of X, such that each cluster $\mathcal{C}_i$ is a node of $\mathcal{T}$ and verifies:*

1. *$\cup_{i \in I} \mathcal{C}_i = X$,*
2. *for all edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subseteq \mathcal{C}_i$, and*
3. *for all $i, j, k \in I$, if k is in a path from i to j in $\mathcal{T}$, then $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$*

The width of a tree-decomposition $(\mathcal{C}, \mathcal{T})$ is equal to $max_{i \in I}|\mathcal{C}_i| - 1$. The tree-width of G is the minimal width over all the tree-decompositions of G.

For the reader who isn't familiar with these notions, note that the above definition refers to a tree $\mathcal{T} = (I, F)$ where F is a set of edges which is required to satisfy the part (3) of this definition.

Even if finding an optimal tree-decomposition is an NP-Hard problem [14], many works have been developed in this direction [15], which often exploit equivalent definitions of this notion, including one based on an algorithmic approach related to *triangulated* graphs. The link between triangulated graphs and tree-decomposition is obvious. Indeed, given a triangulated graph, the set of maximal cliques $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ of (X, E) corresponds to the family of subsets associated with a tree-decomposition. As any graph $G = (X, E)$ is not necessarily triangulated, a tree-decomposition can be approximated by a triangulation of

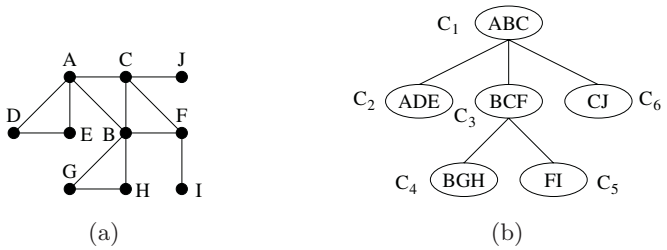


Fig. 1. (a) A constraint graph on 10 variables. (b) A tree-decomposition of this constraint graph.

G which computes a triangulated graph G'. The width of G' is equal to the maximal size of cliques minus one in the resulting graph G'. The tree-width of G is then equal to the minimal width over all triangulations.

The graph in figure 1(a) is already triangulated. The maximum size of cliques is three and the tree-width of this graph is two. In figure 1(b), a tree whose nodes correspond to maximal cliques of the triangulated graph is a possible tree-decomposition for the graph of figure 1(a). So, we get $\mathcal{C}_1 = \{A, B, C\}$, $\mathcal{C}_2 = \{A, D, E\}$, $\mathcal{C}_3 = \{B, C, F\}$, $\mathcal{C}_4 = \{B, G, H\}$, $\mathcal{C}_5 = \{F, I\}$ and $\mathcal{C}_6 = \{C, J\}$.

The notion of tree-decomposition is exploited in the classical CSPs framework by many structural decomposition methods (see [16] for a survey about such methods and a theoretical comparison). These methods have the advantage of providing the best known bounds for the theoretical time complexity. For instance, the CSP decomposition method called *Tree-Clustering* [11,12] is generally presented using an approximation of an optimal triangulation. It has a time complexity in $O(m.d^{w^+ + 1})$ with $w^+ + 1$ the size of the biggest cluster ($w^+ + 1 \leq n$). However, the space complexity is in $O(n.s.d^s)$ with s the maximal size of minimal separators (i.e. the size $s \leq w^+$ of the biggest intersection between two clusters). Finally, note that for every decomposition which induces a value w^+, we have $w \leq w^+$ with w the tree-width of the initial constraint graph.

The BTD method [1] solves classical CSPs by using the tree-decomposition notion jointly with backtracking techniques. Then, it benefits from a practical efficiency (thanks to enumerative techniques) while providing time complexity bounds equivalent to ones of structural decomposition methods (thanks to the tree-decomposition notion). Its time and space complexities are then similar to Tree-Clustering's ones. However, in practice, BTD obtains better results than Tree-Clustering while performing either as good as classical enumerative methods or better.

In the VCSP framework, the dynamic programming approach proposed by Koster [10] also exploits a tree-decomposition. It has a time complexity in $O(nd^{3(w^+ + 1)})$ and a space complexity in $O(d^{w^+ + 1})$. In the both frameworks, the required space can make the structural methods unusable in practice.

In the next section, we present an enumerative method for solving VCSPs which, by exploiting a tree-decomposition, provides complexity bounds similar to ones given above.

4 The BTD$_{val}$ Algorithm for Solving VCSPs

4.1 Presentation

Like BTD, BTD$_{val}$ (for Backtracking with Tree-Decomposition) proceeds by an enumerative search guided by a static pre-established partial order induced by a tree-decomposition of the constraint network. So, the first step of BTD$_{val}$ consists in computing a tree-decomposition or an approximation of a tree-decomposition. The obtained partial order allows to exploit some structural properties of the graph, during the search, in order to prune some branches of the search tree. Hence, BTD$_{val}$ differs from other techniques in the following points:

- the variable assignment order is induced by a tree-decomposition of the constraint graph,
- some subproblems won't be visited again if it we have computed yet their optimal valuation (notion of *structural valued good*).

Although our method is called BTD$_{val}$ for Backtracking with Tree-Decomposition, we will see later that the enumerative search can be based on BB or vFC.

4.2 Theoretical Foundations

In the following, let us consider an instance $\mathcal{P} = (X, D, C, R, S, \phi)$ and a tree-decomposition $(\mathcal{C}, \mathcal{T})$ (or an approximation) of the constraint graph (X, C). We assume that the elements of $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ are indexed with respect to the notion of *compatible numbering*:

Definition 6 *A numbering on $\mathcal{C}$ compatible with a prefix numbering of $\mathcal{T} = (I, F)$ with $\mathcal{C}_1$ the root is called* ***compatible numbering*** *$N_{\mathcal{C}}$.*

Remark that in the previous definition, $\mathcal{T} = (I, F)$ is a tree (according to definition 5) with I the set of indices and F the set of edges. For example, figure 1(b) presents a compatible numbering on $\mathcal{C}$. We note $Desc(\mathcal{C}_j)$ the set of variables belonging to the union of the descendants $\mathcal{C}_k$ of $\mathcal{C}_j$ in the tree rooted in $\mathcal{C}_j$, $\mathcal{C}_j$ included. For instance, $Desc(\mathcal{C}_3) = \mathcal{C}_3 \cup \mathcal{C}_4 \cup \mathcal{C}_5 = \{B, C, F, G, H, I\}$. Note that the numbering $N_{\mathcal{C}}$ defines a partial variable ordering that permits to get an enumeration order on the variables of $\mathcal{P}$:

Definition 7 *A* ***compatible enumeration order*** *is an order $\preceq_X$ on the variables of X such that $\forall x, y \in X, x \preceq_X y$ if $\exists \mathcal{C}_i \ni x, \forall \mathcal{C}_j \ni y, i \leq j$.*

For example, the alphabetical order $A, B, \ldots, I, J$ is a compatible enumeration order. The tree-decomposition with the numbering N_C permits to partition the constraint set.

Definition 8 *Let $\mathcal{C}_i$ be a cluster. The* ***set $E_{\mathcal{P},\mathcal{C}_i}$ of proper constraints*** *of cluster $\mathcal{C}_i$ is defined by $E_{\mathcal{P},\mathcal{C}_i} = \{c \in C | c \subseteq \mathcal{C}_i$ and $c \not\subseteq \mathcal{C}_{p(i)}\}$ with $\mathcal{C}_{p(i)}$ the parent cluster of $\mathcal{C}_i$.*

The set $E_{\mathcal{P},\mathcal{C}_i}$ contains each constraint $c_{xy} = \{x, y\}$ with x and y two variables of $\mathcal{C}_i$ such that x and y don't belong both to $\mathcal{C}_{p(i)}$ the parent cluster of $\mathcal{C}_i$. For instance, if we consider the problem described in figure 1, we obtain $E_{\mathcal{P},\mathcal{C}_1} = \{c_{AB}, c_{AC}, c_{BC}\}$, $E_{\mathcal{P},\mathcal{C}_2} = \{c_{AD}, c_{AE}, c_{DE}\}$, $E_{\mathcal{P},\mathcal{C}_3} = \{c_{BF}, c_{CF}\}$, $E_{\mathcal{P},\mathcal{C}_4} = \{c_{BG}, c_{BH}, c_{GH}\}$, $E_{\mathcal{P},\mathcal{C}_5} = \{c_{FI}\}$ and $E_{\mathcal{P},\mathcal{C}_6} = \{c_{CJ}\}$.

Property 2 *The sets $(E_{\mathcal{P},\mathcal{C}_i})_i$ form a partition of C.*

Proof: First, we are going to show that $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} = C$.
As $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} \subset C$ is obvious, we have to prove $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} \supset C$.
Let $c \in C$. According to definition 5, there exists at less a cluster $\mathcal{C}_i$ such that $c \subseteq \mathcal{C}_i$. In particular, we necessarily have $c \subseteq \mathcal{C}_k$ where $k = \min\{i | c \subseteq \mathcal{C}_i\}$ and $c \not\subseteq \mathcal{C}_{p(k)}$. Therefore, $c \in E_{\mathcal{P},\mathcal{C}_k}$. So we obtain $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} \supset C$ and then $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} = C$

Now we have to prove that $\forall \mathcal{C}_i, \mathcal{C}_j,\ E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j} = \emptyset$.
Assume that there exists two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ such that $E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j} \neq \emptyset$. Let $c \in E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j}$. According to definition 8, we have $c \subseteq \mathcal{C}_i \cap \mathcal{C}_j$.
Then, according to definition 5, there exists a path between $\mathcal{C}_i$ and $\mathcal{C}_j$ such that if $\mathcal{C}_k$ belongs to this path, $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$. The parent cluster of $\mathcal{C}_i$ or $\mathcal{C}_j$'s one clearly belongs to this path. Therefore $c \subseteq \mathcal{C}_{p(i)}$ or $c \subseteq \mathcal{C}_{p(j)}$. So we obtain a contradiction since $c \in E_{\mathcal{P},\mathcal{C}_i}$ and $c \in E_{\mathcal{P},\mathcal{C}_j}$. So $\forall i, j,\ E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j} = \emptyset$

Hence, the sets $(E_{\mathcal{P},\mathcal{C}_i})_i$ form a partition of C. □

Note that this property becomes fundamental when $\oplus$ isn't idempotent. Indeed, in such a case, we must be careful not to take into account a constraint several times. Exploiting the sets $E_{\mathcal{P},\mathcal{C}_i}$ prevents such a problem from occurring and so ensures that BTD$_{val}$ safely computes the valuation of assignments. Then, we can define the notion of induced VCSP:

Definition 9 *Let $\mathcal{C}_i$ and $\mathcal{C}_j$ be two clusters with $\mathcal{C}_j$ a son of $\mathcal{C}_i$. Let $\mathcal{A}$ be an assignment on $\mathcal{C}_i \cap \mathcal{C}_j$. $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j} = (X_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, D_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, C_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, R_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, S, \phi)$ is the **VCSP induced by** $\mathcal{A}$ on the descent of $\mathcal{C}_i$ rooted in $\mathcal{C}_j$ (i.e. on $\mathcal{C}_j$ and its descendants) with:*

- $X_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} = Desc(\mathcal{C}_j)$,
- $D_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} = \{d_{x,\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} = \{\mathcal{A}[x]\} | x \in \mathcal{C}_i \cap \mathcal{C}_j\} \cup \{d_{x,\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} = d_x | x \in Desc(\mathcal{C}_j) \backslash (\mathcal{C}_i \cap \mathcal{C}_j)\}$,
- $C_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} = E_{\mathcal{P},\mathcal{C}_j} \cup \bigcup_{\mathcal{C}_d \text{ descendant of } \mathcal{C}_j} E_{\mathcal{P},\mathcal{C}_d}$,
- $R_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} = \{r_c \cap \prod_{x \in c} d_{x,\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} \mid c \in C_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}} \text{ and } r_c \in R\}$.

The induced VCSP $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$ corresponds to the VCSP $\mathcal{P}$ restricted to the subproblem rooted in $\mathcal{C}_j$ such that the domain of each variable x in $\mathcal{C}_i \cap \mathcal{C}_j$ is reduced

to the value assigned to x in $\mathcal{A}$. That is, we consider the subproblem whose variables are ones of $\mathcal{C}_j$ and its descendants. As for the constraint set of $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$, it only contains the constraints which exclusively appear in $\mathcal{C}_j$ and its descendants. For instance, given the assignment $\mathcal{A} = (B \leftarrow 2, C \leftarrow 2)$ on $\mathcal{C}_1 \cap \mathcal{C}_3$, let us consider $\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}$ the VCSP induced by $\mathcal{A}$ on the descent of $\mathcal{C}_1$ rooted in $\mathcal{C}_3$. We have $X_{\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}} = \{B, C, F, G, H, I\}$, $d_B = d_C = \{2\}$, $d_F = d_G = d_H = d_I = \{1, 2, 3\}$ and $C_{\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}} = \{c_{BF}, c_{CF}, c_{BG}, c_{BH}, c_{GH}, c_{FI}\}$. Note that the constraint c_{BC} doesn't belong to the constraint set of $\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}$ because it isn't a proper constraint of $\mathcal{C}_3$ ($c_{BC} \subseteq \mathcal{C}_1$ and $\mathcal{C}_1 = \mathcal{C}_{p(3)}$). Now, from the sets $E_{\mathcal{P},\mathcal{C}_i}$, we can introduce the notion of local valuation for a cluster:

Definition 10 *Given a cluster $\mathcal{C}_i$ and an assignment $\mathcal{A}$ on $Y \subset X$ with $Y \cap \mathcal{C}_i \neq \emptyset$. The **local valuation for the cluster** $\mathcal{C}_i$ of the assignment $\mathcal{A}$ with respect to $\mathcal{P}$ (noted $v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A})$) is the local valuation of $\mathcal{A}$ restricted to the constraints of $E_{\mathcal{P},\mathcal{C}_i}$, that is to say $v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) = \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \\ \text{and } \mathcal{A} \text{ violates } c}} \phi(c)$*

In other words, the valuation local for a cluster $\mathcal{C}_i$ only takes into account the constraints proper to $\mathcal{C}_i$. Remark that the local valuation for a cluster can be computed incrementally. This valuation presents many interesting properties. First, its computation only depends on the variables of the considered cluster.

Property 3 *Let $\mathcal{C}_i$ be a cluster and $\mathcal{A}$ an assignment on $Y \subseteq X$ such that $\mathcal{C}_i \subseteq Y$. $v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}[\mathcal{C}_i])$*

Proof:

$$v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) = \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \\ \text{and } \mathcal{A} \text{ violates } c}} \phi(c) = \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \cap \mathcal{C}_i \\ \text{and } \mathcal{A} \text{ violates } c}} \phi(c)$$

$$= \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \cap \mathcal{C}_i \\ \text{and } \mathcal{A}[\mathcal{C}_i] \text{ violates } c}} \phi(c) = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}[\mathcal{C}_i]) \quad \square$$

Then, the aggregation of local valuations for a cluster allows us to compute the valuation of a complete assignment.

Property 4 *Let $\mathcal{A}$ be an assignment on X.*

$$\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\mathcal{C}_i \subseteq X} v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A})$$

Proof: Since the sets $(E_{\mathcal{P},\mathcal{C}_i})_i$ form a partition of C (property 2), each constraint of C is taken into account only once. So, $\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\mathcal{C}_i \subseteq X} v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A})$. $\square$

It follows from these two properties that we can compute the valuation of a complete assignment $\mathcal{A}$ by exploiting only the local valuation for each cluster $\mathcal{C}_i$ of the assignment $\mathcal{A}[\mathcal{C}_i]$. Finally, the next property ensures that the local valuation for a cluster $\mathcal{C}_j$ of an assignment $\mathcal{B}$ with respect to $\mathcal{P}$ is preserved if we considered an induced subproblem which contains $\mathcal{C}_j$.

Property 5 *Let $\mathcal{C}_i$ and $\mathcal{C}_j$ two clusters with $\mathcal{C}_j$ a descendant of $\mathcal{C}_i$. Let $\mathcal{A}$ be an assignment on $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$ and $\mathcal{P}' = \mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$. If $\mathcal{B}$ is an assignment on $\mathcal{C}_j$ such that $\mathcal{B}[\mathcal{C}_j \cap \mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i \cap \mathcal{C}_{p(i)}]$, $v_{\mathcal{P},\mathcal{C}_j}(\mathcal{B}) = v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B})$.*

Proof: as $E_{\mathcal{P},\mathcal{C}_j} = E_{\mathcal{P}',\mathcal{C}_j}$ $v_{\mathcal{P},\mathcal{C}_j}(\mathcal{B}) = v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B})$. □

Now, we are able to define the notion of structural valued good.

Definition 11 *Let $\mathcal{C}_i$ and $\mathcal{C}_j$ two clusters with $\mathcal{C}_j$ a son of $\mathcal{C}_i$. A **structural valued good** of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$ is a pair $(\mathcal{A}, v)$ with $\mathcal{A}$ an assignment on $\mathcal{C}_i \cap \mathcal{C}_j$ and v the optimal valuation of the VCSP $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$.*

For instance, if we consider the assignment $\mathcal{A} = (B \leftarrow 2, C \leftarrow 2)$ on $\mathcal{C}_1 \cap \mathcal{C}_3$, we obtain the good $(\mathcal{A}, 2)$ since the best assignment on $Desc(\mathcal{C}_3)$ is $(B \leftarrow 2, C \leftarrow 2, F \leftarrow 3, G \leftarrow 3, H \leftarrow 3, I \leftarrow 3)$. Note that this assignment violates the constraints c_{BC}, c_{GH} and c_{FI}, but c_{BC} is discarded (since $c_{BC} \notin E_{\mathcal{P},\mathcal{C}_3}$).

Given an assignment $\mathcal{A}$ on $\mathcal{C}_i$, the following theorem expresses that we can compute the valuation of the best assignment $\mathcal{B}$ on $Desc(\mathcal{C}_i)$ with $\mathcal{B}[\mathcal{C}_i] = \mathcal{A}$ by exploiting the optimal valuation of each subproblem rooted in a son $\mathcal{C}_f$ of $\mathcal{C}_i$ and induced by $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_f]$. Note that the optimal valuation of each subproblem is provided either by solving the considered subproblem or by exploiting a structural valued good. Finally, remark that this optimal valuation can be computed independently of ones of other subproblems.

Theorem 1 *Let $\mathcal{C}_i$ be a cluster, $\mathcal{A}$ an assignment on $\mathcal{C}_i$ and $\mathcal{P}' = \mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}],\mathcal{C}_{p(i)}/\mathcal{C}_i}$.*

$$\min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} v_{\mathcal{P}'}(\mathcal{B}) = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_f\ son\ of\ \mathcal{C}_i} \alpha^*_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_f],\mathcal{C}_i/\mathcal{C}_f}}$$

The proof of this theorem requires the following lemma:

Lemma 1 *Let $\mathcal{C}_i$ be a cluster and $\mathcal{A}$ an assignment on $\mathcal{C}_i$.*
Let $\mathcal{P}' = \mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}],\mathcal{C}_{p(i)}/\mathcal{C}_i}$.

Let $\lambda = \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left(\bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right)$.

Let $\lambda' = \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_j) \cup \mathcal{C}_i \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right)$.

We have $\lambda = \lambda'$.

Proof (lemma 1):

For each $\mathcal{C}_j$ son of $\mathcal{C}_i$, we note $\lambda_{\mathcal{C}_j} = \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_j) \cup \mathcal{C}_i \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right]$. We then have $\lambda' = \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \lambda_{\mathcal{C}_j}$.

For each $\mathcal{C}_j$ son of $\mathcal{C}_i$, there exists an assignment $\mathcal{B}_{\mathcal{C}_j}$ on $Desc(\mathcal{C}_j) \cup \mathcal{C}_i$ such that $\mathcal{B}_{\mathcal{C}_j}[\mathcal{C}_i] = \mathcal{A}$ and $\lambda_{\mathcal{C}_j} = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_{\mathcal{C}_j})$. Likewise, there is an assignment $\mathcal{B}_\lambda$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{B}_\lambda[\mathcal{C}_i] = \mathcal{A}$ and $\lambda = \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda) \right]$.

We want to prove that for each son $\mathcal{C}_j$ of $\mathcal{C}_i$, we have
$\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) = \lambda_{\mathcal{C}_j}$.

Assume there exists a son $\mathcal{C}_s$ of $\mathcal{C}_i$ such that
$\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i]) \neq \lambda_{\mathcal{C}_s}$.

By definition of $\lambda_{\mathcal{C}_s}$, $\lambda_{\mathcal{C}_s} \prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i])$
$= \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda)$

Let $\mathcal{B}'$ be an assignment on $Desc(\mathcal{C}_i)$ such that $\mathcal{B}'[\mathcal{C}_i] = \mathcal{A}$ and $\forall \mathcal{C}_j \in Sons(\mathcal{C}_i)$, $\mathcal{B}'[Desc(\mathcal{C}_j) \cup \mathcal{C}_i] = \mathcal{B}_{\mathcal{C}_j}$. Such an assignment exists since $\forall \mathcal{C}_j, \mathcal{C}_{j'} \in Sons(\mathcal{C}_i)$, $Desc(\mathcal{C}_j) \cap Desc(\mathcal{C}_{j'}) \subseteq \mathcal{C}_i$.

Furthermore, we have $\lambda_{\mathcal{C}_s} = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}'[Desc(\mathcal{C}_s) \cup \mathcal{C}_i])$.

So, $\bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}') \right] = \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \lambda_{\mathcal{C}_j} = \lambda'$

$\lambda' = \lambda_{\mathcal{C}_s} \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \lambda_{\mathcal{C}_j}$

$\prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i])$

$\oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) \right]$

$\prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda) \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda) \right] = \lambda$

Hence, we obtain a contradiction with the definition of λ. So, for each son $\mathcal{C}_j$ of $\mathcal{C}_i$, $\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) = \lambda_{\mathcal{C}_j}$. It ensues that $\lambda = \lambda'$. □

Proof (theorem 1):
We note $M = \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} v_{\mathcal{P}'}(\mathcal{B})$.

$$M \underset{property\ 1}{=} \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \mathcal{V}_{\mathcal{P}'}(\mathcal{B}).$$

$$\underset{property\ 4}{=} \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left(\bigoplus_{\mathcal{C}_j \subseteq Desc(\mathcal{C}_i)} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right)$$

$$= \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left(v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{B}) \oplus \bigoplus_{\substack{\mathcal{C}_j|j\neq i, \\ \mathcal{C}_j\subseteq Desc(\mathcal{C}_i)}} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right)$$

$$\underset{property\ 3}{=} \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left(v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{B}[\mathcal{C}_i]) \oplus \bigoplus_{\substack{\mathcal{C}_j|j\neq i, \\ \mathcal{C}_j\subseteq Desc(\mathcal{C}_i)}} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right)$$

For every assignment $\mathcal{B}$ such that $X_{\mathcal{B}} = Desc(\mathcal{C}_i)$ and $\mathcal{B}[\mathcal{C}_i] = \mathcal{A}$, we have $v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{B}[\mathcal{C}_i]) = v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A})$. As $v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A})$ is a constant, we have:

$$M = v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left(\bigoplus_{\substack{\mathcal{C}_j|j\neq i, \\ \mathcal{C}_j\subseteq Desc(\mathcal{C}_i)}} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right)$$

$$= v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left(\bigoplus_{\mathcal{C}_j\in Sons(\mathcal{C}_i)} \left[\bigoplus_{\mathcal{C}_k\subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right)$$

$$\underset{lemma\ 1}{=} v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j\in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_j)\cup\mathcal{C}_i \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left[\bigoplus_{\mathcal{C}_k\subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right)$$

$$M = v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j\in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_j)\ and \\ \mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_j]=\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j]}} \left[\bigoplus_{\mathcal{C}_k\subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right)$$

$$\underset{property\ 5}{=} v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j\in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_j)\ and \\ \mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_j]=\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j]}} \left[\bigoplus_{\mathcal{C}_k\subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P},\mathcal{C}_k}(\mathcal{B}) \right] \right)$$

$$\underset{property\ 4}{=} v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j\in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(\mathcal{C}_j) \\ and\ \mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_j]=\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j]}} \mathcal{V}_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j],\mathcal{C}_i/\mathcal{C}_j}}(\mathcal{B}) \right)$$

$$= v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j\ son\ of\ \mathcal{C}_i} \alpha^*_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j],\mathcal{C}_i/\mathcal{C}_j}} \quad \square$$

From theorem 1, we deduce the following corollary. This corollary establishes the link between the optimal valuation of a subproblem rooted in $\mathcal{C}_i$ and the optimal valuation of each subproblem rooted in a son $\mathcal{C}_j$ of $\mathcal{C}_i$.

Corollary 1 *Let $\mathcal{C}_i$ be a cluster and $\mathcal{A}$ an assignment on $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$.*

$$\alpha^*_{\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}} = \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=\mathcal{C}_i\ and \\ \mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_{p(i)}]=\mathcal{A}}} \left(v_{\mathcal{P},\mathcal{C}_i}(\mathcal{B}) \oplus \bigoplus_{\mathcal{C}_j\ son\ of\ \mathcal{C}_i} \alpha^*_{\mathcal{P}_{\mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_j],\mathcal{C}_i/\mathcal{C}_j}} \right)$$

4.3 The BTD$_{val}$ Algorithm

The BTD$_{val}$ method is based on the BB algorithm (note that we can also base it on vFC). It explores the search space by exploiting a compatible order, which begins with the variables of the root cluster $\mathcal{C}_1$. Inside a cluster $\mathcal{C}_i$, it proceeds

classically like BB by assigning a value to a variable, by maintaining and comparing upper and lower bounds and by backtracking if a lower bound is greater than (or equal to) the corresponding upper bound. However, unlike BB, BTD$_{val}$ uses two kinds of bounds: local bounds and global ones. The local bounds only take into account the subproblem rooted in $\mathcal{C}_i$ (namely the induced VCSP $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$ with $\mathcal{A}$ the current assignment on $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$). The local lower bound corresponds to the valuation of the current assignment on $Desc(\mathcal{C}_i)$, that is to say, the local valuation of the current assignment with respect to $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$. The local upper bound is then defined by the valuation of the best known assignment $\mathcal{B}$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}$. In other words, it's the valuation of the best known solution for $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$. The global bounds are similar to BB's ones, that is to say the local valuation of the current assignment for the lower bound and the valuation of the best known solution for the upper one.

When every variable in $\mathcal{C}_i$ is assigned, if each lower bound is less than the corresponding upper bound, BTD$_{val}$ keeps on the search with the first son of $\mathcal{C}_i$ (if there is one). More generally, let us consider a son $\mathcal{C}_j$ of $\mathcal{C}_i$. Given the current assignment $\mathcal{A}$ on $\mathcal{C}_i$, BTD$_{val}$ checks whether the assignment $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ corresponds to a valued structural good:

- if so, BTD$_{val}$ aggregates the valuation associated to this valued good with each lower bound.
- else, it extends $\mathcal{A}$ on $Desc(\mathcal{C}_j)$ in order to compute the valuation v of the best assignment $\mathcal{B}$ such that $\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$. Then, it aggregates v with each lower bound and it records the valued good $(\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], v)$.

If, after having proceeded the son $\mathcal{C}_j$, the two lower bounds don't exceed their respective upper bound, BTD$_{val}$ keeps on the search with the next son of $\mathcal{C}_i$. Remark that by exploiting the structural valued goods, BTD$_{val}$ doesn't solve again some subproblems. So the variables of these subproblems aren't assigned again. Hence we call such a phenomenon a *forward-jump* (by analogy with backjump). For instance, suppose that we use the alphabetical order as variable order and that, after assigning the variable F in $\mathcal{C}_3$, we exploit a good on $\mathcal{C}_3 \cap \mathcal{C}_4$. Then, we try to assign I without exploring again $Desc(\mathcal{C}_4)$. If every son has been proceeded and each lower bound doesn't exceed its corresponding upper bound, then a better solution for $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$ has been found. Finally, if a failure occurs, BTD$_{val}$ tries to modify the current assignment on $\mathcal{C}_i$.

In fact, due to the structural valued good definition, the global lower bound is defined by the valuation of the best extension of $\mathcal{A}$ on every cluster which precedes the current cluster in the used compatible enumeration. It's the same for the local lower bound, but we only consider the clusters belonging to the descent of the current cluster. Remark that we consider an extension of $\mathcal{A}$, and not $\mathcal{A}$, because $\mathcal{A}$ only contains the variables belonging to a cluster located on the path between the root cluster and the current cluster. Finally note that the global upper bound is the same as BB's one, unlike the global lower bound which is better than BB's one.

Figure 2 describes the BTD$_{val}$ algorithm. Given an assignment $\mathcal{A}$ and a cluster $\mathcal{C}_i$, BTD$_{val}$ looks for the best assignment $\mathcal{B}$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{A}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}]$ and $v_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}], \mathcal{C}_{p(i)}/\mathcal{C}_i}}(\mathcal{B}) \prec \alpha_{\mathcal{C}_i}$, where:

- $V_{\mathcal{C}_i}$ is the set of unassigned variables in $\mathcal{C}_i$,
- $\alpha_{\mathcal{C}_1}$ is the valuation of the best known solution,
- l_{tot} is the valuation of the best extension $\mathcal{A}'$ of $\mathcal{A}$ on all the clusters which precede $\mathcal{C}_i$ according to the compatible numbering ($l_{tot} = v_{\mathcal{P}}(\mathcal{A}') \prec \alpha_{\mathcal{C}_1}$),
- $\alpha_{\mathcal{C}_i}$ is the valuation of the best known assignment $\mathcal{B}'$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$
- $l_{\mathcal{C}_i} = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \prec \alpha_{\mathcal{C}_i}$.

If BTD$_{val}$ finds such an assignment, it returns its valuation, otherwise it returns a valuation greater than (or equal to) $\alpha_{\mathcal{C}_i}$. The initial call is BTD$_{val}(\emptyset, \mathcal{C}_1, \mathcal{C}_1, \bot, \top, \bot, \top)$.

Theorem 2 *BTD$_{val}$ is sound, complete and terminates.*

```
    BTD_val(A, C_i, V_{C_i}, l_tot, α_{C_1}, l_{C_i}, α_{C_i})
1.  If V_{C_i} = ∅
2.  Then
3.     If Sons(C_i) = ∅ Then Return l_{C_i}
4.     Else
5.        F ← Sons(C_i)
6.        α ← ⊥
7.        While F ≠ ∅ and α ⊕ l_tot ≺ α_{C_1} and α ⊕ l_{C_i} ≺ α_{C_i} Do
8.           Choose C_j in F
9.           F ← F\{C_j}
10.          If (A[C_j ∩ C_i], v) is a good of C_i/C_j in G Then α ← α ⊕ v
11.          Else
12.             v ← BTD_val(A, C_j, C_j\(C_j ∩ C_i), l_tot ⊕ α, α_{C_1}, ⊥, α_{C_i})
13.             α ← α ⊕ v
14.             Record the good (A[C_j ∩ C_i], v) of C_i/C_j in G
15.          EndIf
16.       EndWhile
17.       Return α ⊕ l_{C_i}
18.    EndIf
19. Else
20.    Choose x ∈ V_{C_i}
21.    d ← d_x
22.    While d ≠ ∅ and l_tot ≺ α_{C_1} and l_{C_i} ≺ α_{C_i} Do
23.       Choose a in d
24.       d ← d\{a}
25.       L ← {c = {x, y} ∈ E_{P,C_i} | y ∉ V_{C_i}}
26.       l_a ← ⊥
27.       While L ≠ ∅ and l_tot ⊕ l_a ≺ α_{C_1} and l_{C_i} ⊕ l_a ≺ α_{C_i} Do
28.          Choose c in L
29.          L ← L\{c}
30.          If c violates A ∪ {x ← a} Then l_a ← l_a ⊕ φ(c)
31.       EndWhile
32.       If l_tot ⊕ l_a ≺ α_{C_1} and l_{C_i} ⊕ l_a ≺ α_{C_i}
33.       Then α_{C_i} ← min(α_{C_i}, BTD_val(A ∪ {x ← a}, C_i, V_{C_i}\{x}, l_tot ⊕ l_a, α_{C_1}, l_{C_i} ⊕ l_a, α_{C_i}))
34.       EndIf
35.    EndWhile
36.    Return α_{C_i}
37. EndIf
```

Fig. 2. The BTD$_{val}$ algorithm.

Finally, we provide the time and space complexities of BTD$_{val}$. We suppose that a tree-decomposition (or an approximation) has been computed. Therefore the parameters used in the next theorem are related to this decomposition. BTD$_{val}$ obtains complexities similar to Tree-Clustering's ones:

Theorem 3 *BTD$_{val}$ has a time complexity in $O(n.s^2.m.\log(d).d^{w^+ +1})$ and a space complexity in $O(n.s.d^s)$ with $w^+ + 1$ the size of the biggest $\mathcal{C}_k$ and s the size of the biggest intersection $\mathcal{C}_i \cap \mathcal{C}_j$ where $\mathcal{C}_j$ is a son of $\mathcal{C}_i$.*

5 Related Works

BTD$_{val}$ is mostly based on tree-decomposition. So, works like Tree-Clustering and its improvements [11,12] or the dynamic programming approach of Koster [10] are close to our approach. BTD$_{val}$ can be considered as an hybrid approach realizing a tradeoff between practical time and space complexity. In [12], Dechter and El Fattah present a time-space tradeoff scheme. This scheme allows them to propose a spectrum of algorithms such that tree-clustering and cycle-cutset conditioning (linear for space complexity) are two extremes in this spectrum. Another interesting idea in their work is the possibility to modify the size of separators to minimize space. This idea can also be exploited in BTD$_{val}$.

BTD$_{val}$ presents a better time complexity than the dynamic programming approach of Koster. Then, BTD$_{val}$ differs from this approach in computing a tree-decomposition (or an approximation of a tree-decomposition). BTD$_{val}$ exploits a triangulation of the constraint graph, while the dynamic programming approach uses a heuristic method and network flow techniques. Furthermore, Koster proposes several pretreatments. In particular, one of these pretreatments allows to reduce the size of the constraint graph, which may also reduce the time complexity. So adding such pretreatments may be useful for our approach.

BTD$_{val}$ is close to a method like the russian dolls search [9]. Indeed, in order to find the optimal valuation of a VCSP, BTD$_{val}$ solves many subproblems according a pre-established compatible order. The BTD$_{val}$'s clusters have a role similar to one of variables in RDS. Nevertheless, the two methods exploit differently the optimal valuations of subproblems. Like BTD$_{val}$, the method Tree-RDS [17] (a variant of RDS) takes advantage of the constraint graph in order to determine whether some problems are independent or not. However, if the independence of subproblems is used similarly, the Tree-RDS's subproblems differ conceptually from BTD$_{val}$'s ones. It's the same for the adaptation [18] of the algorithm Pseudo-Tree Search and its combination with a variant of RDS.

6 Conclusion

In this paper, we have defined a new method (called BTD$_{val}$) for solving valued CSPs. This method can actually be based on BB or on vFC. Thanks to the notion of structural valued goods we have introduced, BTD$_{val}$ obtains complexity bounds similar to (or better than) the best known ones. Indeed, the time complexity of BTD$_{val}$ is $O(ns^2m\log(d).d^{w^+ +1})$ with $w^+ + 1$ the size of the biggest

cluster while the space complexity is $O(nsd^s)$ with s the size of the biggest intersection between two clusters. Now, an experimental study is required to assess the practical interest of our approach.

Among the possible extensions of this work, we must base our algorithm on more efficient methods like the russian dolls search or algorithms which use directional arc-consistency [19,20,21]. Using such methods seems natural since BTD_{val} exploits a compatible enumeration order.

References

1. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.
2. E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
3. S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proc. of the 14^{th} IJCAI*, pages 624–630, 1995.
4. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proc. of the 14^{th} IJCAI*, pages 631–637, 1995.
5. P. Dago and G. Verfaillie. Nogood Recording for Valued Constraint Satisfaction Problems. In *Proc. of ICTAI 96*, pages 132–139, 1996.
6. J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.
7. T. Schiex. Une comparaison des cohérences d'arc dans les Max-CSP. In *Actes des JNPC'2002*, pages 209–223, 2002. In french.
8. J. Larrosa. On arc and node consistency. In *Proc. of AAAI*, 2002.
9. G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proc. of the 14^{th} AAAI*, pages 181–187, 1996.
10. A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, November 1999.
11. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
12. R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.
13. N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of tree-width. *Algorithms*, 7:309–322, 1986.
14. S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embedding in a k-tree. *SIAM Journal of Discrete Mathematics*, 8:277–284, 1987.
15. A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125:3–17, 2001.
16. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
17. P. Meseguer and M. Sánchez. Tree-based Russian Doll Search. In *Proc. of Workshop on soft constraint*. CP'2000, 2000.
18. J. Larrosa, P. Meseguer, and M. Sánchez. Pseudo-Tree Search with Soft Constraints. In *Proc. of the 15^{th} ECAI*, pages 131–135, 2002.
19. R. Wallace. Directed arc consistency preprocessing. In *Proc. of the ECAI-94 Workshop on Constraint Processing, LNCS 923*, pages 121–137, 1994.
20. R. Wallace. Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem. In *Proc. of AAAI*, pages 188–195, 1996.
21. J. Larrosa and P. Meseguer. Exploiting the use of DAC in Max-CSP. In *Proc. of the 2^{nd} CP*, pages 308–322, 1996.

Decomposition and good recording for solving Max-CSPs

Jégou Philippe and **Terrioux Cyril** [1]

Abstract. [22] presents a new method called BTD for solving Valued CSPs and so Max-CSPs. This method based both on enumerative techniques and the tree-decomposition notion provides better theoretical time complexity bounds than classical enumerative methods and aims to benefit of the practical efficiency of enumerative methods thanks to the structural goods which are recorded and exploited during the search. However, [22] does not provide any experimental result and it does not discuss the way of finding an optimal solution from the optimal cost (because BTD only computes the cost of the best assignment). Providing an optimal solution is an important task for a solver, especially when we consider real-world instances. So, in this paper, we first raise these two questions. Then we explain how a solution can be efficiently computed and we provide experimental results which emphasize the practical interest of BTD.

1 INTRODUCTION

Many various problems, like boolean formulae satisfiability, configuration, graph coloring, planning, ..., can be expressed as a Constraint Satisfaction Problem (CSP). A CSP is defined by a set of variables (each one having a finite domain) and a set of constraints. Each constraint forbids some combinations of values for a subset of variables. Solving a CSP requires to assign a value to each variable such that the assignment satisfies all constraints. Determining whether a CSP has a solution is a NP-complete task. When we consider real-world problems, they involve two kinds of constraints: hard constraints which express some physical properties and soft constraints which express notions like possibility or preference. The first ones must be satisfied whereas the second ones can be violated. Unfortunately, representing these problems in the CSP formalism (where each constraint must be satisfied) often produces over-constrained problems which do not have any solution. However, even if there is no perfect solution, we can be interested by finding an assignment which optimizes a certain criterion on the constraint satisfaction. Hence, recently, many extensions of the CSP framework have been proposed (e.g. [6, 2, 21]).

In this paper, we focus our study on the Max-CSP problem [6]. Solving a Max-CSP instance requires to find an assignment which maximizes the number of satisfied constraints. Many algorithms have been defined in the past years for solving this problem. On the one hand, they exploit enumerative techniques like Branch and Bound (BB) or the arc-consistency notion [13, 10, 14, 4]. On the other hand, some other methods are based on the dynamic programming approach [23, 8, 15, 16, 17, 11]. Some of them exploit the problem structure like [8, 15, 12, 11]. These different approaches have been provided interesting results in some different cases. In [22], an hybrid method, called BTD, is presented for solving the Valued CSP problem [21] which is a generalization of the Max-CSP problem. This method is based both on enumerative techniques and on the tree-decomposition notion. It aims to benefit from the practical efficiency of enumerative methods while providing better theoretical time complexity bounds than enumerative methods. From [22], two important questions are raised. The first one is how we can compute an optimal solution from the optimal cost (because BTD only computes the cost of the best assignment, and not the assignment itself). Providing an optimal solution is one of the most important tasks for a solver, especially when we consider real-world instances. The second raised question deals with the practical efficiency of this method. BTD presents a good behaviour on classical CSPs [7]. In contrast, its behaviour on the Max-CSP problem is unknown and must be assessed. This article tries to answer these two important questions.

The paper is organized as follows. Section 2 introduces the basic notions about CSPs and Max-CSPs. Section 3 is devoted to the BTD method. Then, section 4 explains how we can compute an optimal solution. Finally, we present some empirical results in section 5, before concluding and giving some ideas of future works in section 6.

2 BASIC NOTIONS

A *constraint satisfaction problem* (CSP) is defined by a tuple (X, D, C, R). X is a set $\{x_1, \ldots, x_n\}$ of n variables. Each variable x_i takes its values in the finite domain d_{x_i} from D. Variables are subject to constraints from C. Each constraint c is defined as a set $\{x_{c_1}, \ldots, x_{c_k}\}$ of variables. A relation r_c (from R) is associated with each constraint c such that r_c represents the set of allowed tuples over $d_{x_{c_1}} \times \cdots \times d_{x_{c_k}}$. Note that we can also define constraints by using functions or predicates for instance. Given $Y \subseteq X$ such that $Y = \{x_1, \ldots, x_k\}$, an *assignment* of variables from Y is a tuple $\mathcal{A} = (v_1, \ldots, v_k)$ from $d_{x_1} \times \cdots \times d_{x_k}$. A constraint c is said *satisfied* by $\mathcal{A}$ if $c \subseteq Y, (v_1, \ldots, v_k)[c] \in r_c$, *violated* otherwise. We note the assignment $(v_1, \ldots, v_k)$ in the more meaningful form $(x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k)$. In this paper, without lost of generality, we only consider binary constraints (i.e. constraints which involve two variables). So, the structure of a CSP can be represented by the graph (X, C), called the *constraint graph*, whose vertices are the variables of X and for which there an edge between two vertices if the corresponding variables share a constraint. Given an instance, the CSP problem consists in determining whether there is an assignment of each variable which satisfies each constraint. This problem is NP-Complete. Unfortunately, representing real-world instances as a CSP may produce over-constrained instances which do not have any solution. In such cases, as there is no perfect solution, we can be interested by finding an assignment which optimizes a certain criterion on the constraint satisfaction. Hence, in the recent years, many extensions of the CSP framework have been proposed (e.g. [6, 2, 21]).

In this paper, we focus our study on the Max-CSP problem [6]. Solving a Max-CSP instance requires to find an assignment which

[1] LSIS, Université d'Aix-Marseille III, Marseille, France. Email: {philippe.jegou,cyril.terrioux}@lsis.org

maximizes the number of satisfied constraints. In other words, we want to minimize the number of violated constraints. The number of constraints violated by an assignment is called the cost of this assignment. Many complete algorithms have been recently developed for solving Max-CSPs. They are often based on enumerative techniques or on dynamic programming approaches. Enumerative methods exploit a lower bound, which underestimates the cost of the best complete extension of the current assignment, and an upper bound which is generally the cost of the best known assignment. Then, if the lower bound does not exceed the upper one, they extend the current assignment by assigning a new variable. Otherwise, they backtrack and try to assign a new value to the last assigned variable. If all the values have been tried, they backtrack again. The efficiency of enumerative methods mostly depends on the quality of the lower and upper bounds. The greater the lower bound (respectively the smaller the upper bound) is, the less nodes are visited and constraint checks performed. The basic enumerative method is Branch and Bound (BB). It simply uses the cost of the current assignment as lower bound. Then, many improvements have been proposed from the classical CSP framework. For instance, the lower bound can be improved by using prospective techniques like Forward-Checking (FC [6]) or the arc-consistency notion [13, 10, 14, 4]. On the other hand, some other methods are based on the dynamic programming approach [23, 8, 15, 16, 17, 12, 11]. These methods divide the problem into different subproblems. Then each subproblem is solved and some informations are recorded during each resolution. These informations are exploited for solving a bigger subproblem, and so on until the whole problem is solved. In particular, they can be used for computing good lower or upper bounds like in Russian dolls search (RDS [23]) and its variants [15, 16, 12, 17]. Some of these methods exploit the problem structure like [8, 15, 12, 11]. From a practical viewpoint, the enumerative methods which use arc-consistency obtain goods results when the instances to solve have a limited size. However, they seem have some difficulties in solving larger instances like the CELAR real-world instances [3]. On the other hand, dynamic programming methods may seem perform many redundant searches or visit some useless parts of the search space. Nonetheless, in practice, they can obtain interesting results. For instance, RDS [23] and the Koster's structural method [8] succeed in solving the SCEN-06 instance of the CELAR (which is one of the hardest instances).

3 THE BTD METHOD

In [22] a new method is proposed for solving Valued CSPs [21] and so Max-CSP. This method called BTD (for Backtracking with Tree-Decomposition) is an enumerative method which is guided by a tree-decomposition of the constraint graph. A *tree-decomposition* [18] of a graph $G = (X, E)$ is a pair $(\mathcal{C}, \mathcal{T})$ with $\mathcal{T} = (I, F)$ a tree and $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ a family of subsets of X, such that each cluster $\mathcal{C}_i$ is a node of $\mathcal{T}$ and verifies: (1) $\cup_{i \in I} \mathcal{C}_i = X$, (2) for each edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subseteq \mathcal{C}_i$, (3) for all $i, j, k \in I$, if k is on a path from i to j in $\mathcal{T}$, then $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$. The width of a tree-decomposition $(\mathcal{C}, \mathcal{T})$ is equal to $max_{i \in I}|\mathcal{C}_i| - 1$. The tree-width of G is the minimal width over all the tree-decompositions of G. Note that finding an optimal tree-decomposition is a NP-Hard problem [1]. However, we can easily compute a good tree-decomposition by using the notion of *triangulated graphs*. Figure 1(b) presents a possible tree-decomposition for the graph of figure 1(a). So, we get $\mathcal{C}_1 = \{x_1, x_2, x_3\}$, $\mathcal{C}_2 = \{x_2, x_3, x_4, x_5\}$, $\mathcal{C}_3 = \{x_4, x_5, x_6\}$ and $\mathcal{C}_4 = \{x_3, x_7, x_8\}$, and the tree-width is 3. In the following, from a tree-decomposition, we consider a rooted tree (I, F) where $\mathcal{C}_1$ is the root

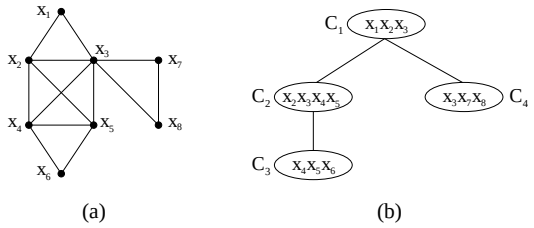


Figure 1. (a) A constraint graph on 8 variables. (b) A tree-decomposition of this constraint graph.

and we note $Desc(\mathcal{C}_j)$ the set of variables which belong to $\mathcal{C}_j$ or to any descendant $\mathcal{C}_k$ of $\mathcal{C}_j$ in the tree rooted in $\mathcal{C}_j$. For instance, $Desc(\mathcal{C}_2) = \mathcal{C}_2 \cup \mathcal{C}_3 = \{x_2, x_3, x_4, x_5, x_6\}$.

The first step of BTD consists in computing a tree-decomposition of the constraint graph. The computed tree-decomposition induces a partial variable ordering which allows BTD to exploit some structural properties of the graph and so to prune some parts of the search tree. In fact, variables are assigned according to a depth-first traversal of the rooted tree. In other words, we first assign the variables of the root cluster $\mathcal{C}_1$, then we assign the variables of $\mathcal{C}_2$, then $\mathcal{C}_3$'s ones, and so on. For example, $x_1, x_2, \ldots, x_8$ is a possible variable ordering. Furthermore, the tree-decomposition and the variable ordering allow BTD to divide the problem $\mathcal{P}$ into many subproblems. Given two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ (with $\mathcal{C}_j$ a $\mathcal{C}_i$'s son), the subproblem rooted in $\mathcal{C}_j$ depends on the current assignment $\mathcal{A}$ on $\mathcal{C}_i \cap \mathcal{C}_j$. It is denoted $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$. Its variable set is equal to $Desc(\mathcal{C}_j)$. The domain of each variable which belongs to $\mathcal{C}_i \cap \mathcal{C}_j$ is restricted to its value in $\mathcal{A}$. Regarding the constraint set, it contains the constraints which involve at least one variable which exclusively appears in $\mathcal{C}_j$ or in a descendant of $\mathcal{C}_j$. For instance, let us consider the CSP whose constraint graph is provided in figure 1(a). We assume that each domain is $\{1, 2, 3\}$ and each constraint $c_{ij} = \{x_i, x_j\}$ means $x_i \neq x_j$. Given $\mathcal{A} = (x_2 \leftarrow 2, x_3 \leftarrow 2)$, the variable set of $\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_2}$ is $Desc(\mathcal{C}_2)$, (with $d_{x_2} = d_{x_3} = \{2\}$ and $d_{x_4} = d_{x_5} = d_{x_6} = \{1, 2, 3\}$) and its constraint set is $\{c_{24}, c_{25}, c_{34}, c_{35}, c_{45}, c_{46}, c_{56}\}$. Note that the constraint c_{23} does not belong to its constraint set because x_2 and x_3 appear both in $\mathcal{C}_1$. Remark that the definition of subproblems defines a partition of the constraint set. Such a partition ensures that BTD takes into account each constraint only once and so that it safely computes the cost of any assignment. Finally, the tree-decomposition notion permits to define the *valued structural good* notion (by analogy with the nogood notion). A structural valued good of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$ (with $\mathcal{C}_j$ a $\mathcal{C}_i$'s son) is a pair $(\mathcal{A}, v)$ with $\mathcal{A}$ the current assignment on $\mathcal{C}_i \cap \mathcal{C}_j$ and v the optimal cost of the subproblem $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$. For instance, if we consider the assignment $\mathcal{A} = (x_2 \leftarrow 2, x_3 \leftarrow 2)$ on $\mathcal{C}_1 \cap \mathcal{C}_2$, we obtain the good $(\mathcal{A}, 0)$ since the best assignment on $Desc(\mathcal{C}_2)$ is $(x_2 \leftarrow 2, x_3 \leftarrow 2, x_4 \leftarrow 1, x_5 \leftarrow 3, x_6 \leftarrow 2)$ (which violates no constraint because c_{23} does not belong to $\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_2}$).

Figure 2 describes the BTD algorithm based on BB. It explores the search space according to the variable ordering induced by the tree-decomposition. So, it begins with the variables of the root cluster $\mathcal{C}_1$. Inside a cluster $\mathcal{C}_i$, it proceeds classically like BB by assigning a value to a variable, by maintaining and comparing upper and lower bounds and by backtracking if the lower bound exceeds the upper bound. The bounds in BTD are similar to BB's ones but they only take into account the constraints of the subproblem $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$ (with

BTD$(\mathcal{A}, \mathcal{C}_i, V_{\mathcal{C}_i}, l_{\mathcal{C}_i}, \alpha_{\mathcal{C}_i})$
1. **If** $V_{\mathcal{C}_i} = \emptyset$
2. **Then**
3. $F \leftarrow Sons(\mathcal{C}_i)$
4. **While** $F \neq \emptyset$ **and** $l_{\mathcal{C}_i} < \alpha_{\mathcal{C}_i}$ **Do**
5. Choose $\mathcal{C}_j$ in F
6. $F \leftarrow F \backslash \{\mathcal{C}_j\}$
7. **If** $(\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], v)$ is a good of $\mathcal{C}_i/\mathcal{C}_j$ in G **Then** $l_{\mathcal{C}_i} \leftarrow l_{\mathcal{C}_i} + v$
8. **Else**
9. $v \leftarrow$ **BTD**$(\mathcal{A}, \mathcal{C}_j, \mathcal{C}_j \backslash (\mathcal{C}_j \cap \mathcal{C}_i), 0, \alpha_{\mathcal{C}_1})$
10. $l_{\mathcal{C}_i} \leftarrow l_{\mathcal{C}_i} + v$
11. Record the good $(\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], v)$ of $\mathcal{C}_i/\mathcal{C}_j$ in G
12. **EndIf**
13. **EndWhile**
14. Return $l_{\mathcal{C}_i}$
15. **Else**
16. Choose $x \in V_{\mathcal{C}_i}$
17. $d \leftarrow d_x$
18. **While** $d \neq \emptyset$ **and** $l_{\mathcal{C}_i} < \alpha_{\mathcal{C}_i}$ **Do**
19. Choose a in d
20. $d \leftarrow d \backslash \{a\}$
21. $l_a \leftarrow |\{c = \{x, y\} \in C | y \notin V_{\mathcal{C}_i} \; and \; \mathcal{A} \cup \{x \leftarrow a\} \; violates \; c\}|$
22. **If** $l_{\mathcal{C}_i} + l_a < \alpha_{\mathcal{C}_i}$
23. **Then** $\alpha_{\mathcal{C}_i} \leftarrow \min(\alpha_{\mathcal{C}_i},$ **BTD**$(\mathcal{A} \cup \{x \leftarrow a\}, \mathcal{C}_i, V_{\mathcal{C}_i} \backslash \{x\}, l_{\mathcal{C}_i} + l_a, \alpha_{\mathcal{C}_i}))$
24. **EndIf**
25. **EndWhile**
26. Return $\alpha_{\mathcal{C}_i}$
27. **EndIf**

Figure 2. The BTD algorithm.

$\mathcal{C}_{p(i)}$ the $\mathcal{C}_i$'s father and $\mathcal{A}$ the assignment on $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$). The lower bound corresponds to the cost of the current assignment on $Desc(\mathcal{C}_i)$ while the upper one is defined by the cost of the best known solution for the subproblem $\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}$. When every variable in $\mathcal{C}_i$ is assigned, if the lower bound is less than the upper bound, BTD keeps on the search with the first son of $\mathcal{C}_i$ (if there is one). More generally, let us consider a son $\mathcal{C}_j$ of $\mathcal{C}_i$. Given the current assignment $\mathcal{A}$ on $\mathcal{C}_i$, BTD checks whether the assignment $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ corresponds to a valued structural good. If so, BTD adds its associated cost v to the lower bound. Otherwise it extends $\mathcal{A}$ on $Desc(\mathcal{C}_j)$ in order to compute the optimal cost v of the subproblem $\mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], \mathcal{C}_i/\mathcal{C}_j}$. Then, it adds v to the lower bound and it records the valued good $(\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], v)$. If, after having proceeded the son $\mathcal{C}_j$, the lower bound does not exceed the upper bound, BTD keeps on the search with the next son of $\mathcal{C}_i$. Finally, if a failure occurs, BTD tries to modify the current assignment on $\mathcal{C}_i$.

In figure 2, given an assignment $\mathcal{A}$ and a cluster $\mathcal{C}_i$, BTD looks for the best assignment $\mathcal{B}$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{A}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}]$ and the cost of $\mathcal{B}$ is less than $\alpha_{\mathcal{C}_i}$. $V_{\mathcal{C}_i}$ denotes the set of unassigned variables in $\mathcal{C}_i$, $l_{\mathcal{C}_i}$ the lower bound and $\alpha_{\mathcal{C}_i}$ the upper bound with respect to the subproblem $\mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}], \mathcal{C}_{p(i)}/\mathcal{C}_i}$. If BTD finds such an assignment, it returns its cost, otherwise it returns a cost greater than (or equal to) $\alpha_{\mathcal{C}_i}$. The first call is BTD$(\emptyset, \mathcal{C}_1, \mathcal{C}_1, 0, +\infty)$.

Finally, BTD has a space complexity in $O(n.s.d^s)$ and a time complexity in $O(n.s^2.m.\log(d).d^{w+1})$ with $w+1$ the size of the largest $\mathcal{C}_k$ and s the size of the largest intersection $\mathcal{C}_i \cap \mathcal{C}_j$ with $\mathcal{C}_j$ a son of $\mathcal{C}_i$ [22]. These complexities assume that a tree-decomposition has been computed (structural parameters w and s are related to this decomposition).

4 HOW TO COMPUTE A SOLUTION?

BTD only provides the optimal cost α of the instance we want to solve. It does not compute an optimal solution of this instance. Indeed, when BTD exploits a good of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$, it does not assign again the variables of $Desc(\mathcal{C}_j) - (\mathcal{C}_i \cap \mathcal{C}_j)$. What is called in [7, 22] a *forward-jump* (by analogy with the backjump notion). For instance, after having assigned the variable x_3 in $\mathcal{C}_1$, if BTD exploits a good on $\mathcal{C}_1 \cap \mathcal{C}_2$, then, it checks for a good on $\mathcal{C}_1 \cap \mathcal{C}_4$ without exploring again $Desc(\mathcal{C}_2)$. Hence, as many variables may be unassigned, BTD cannot provide a solution of the problem we want to solve. It can only look for its optimal cost. Even if computing the optimal cost may be an important task, the main task in the Max-CSP framework is to provide an assignment which minimizes the number of violated constraints. What raises a fundamental question for BTD: how can we compute an optimal solution from the optimal cost provided by BTD? More generally, this question is often raised for algorithms like BTD which make a trade-off between time and space. As an example, the adaptation of Tree-Clustering proposed in [5] with a limited space-complexity suffers from the same drawback since it only records informations on each separator and then it cannot produce a solution in a backtrack free-manner.

In this section, we explain how we can build a solution from the optimal cost α. A basic way consists in using any enumerative algorithm for looking for an assignment with a cost α. But such a way is clearly inefficient and has a time-complexity worse than BTD's one. By so doing, we do not benefit from the tree-decomposition or from the goods which BTD has recorded during the search. So we can build a solution thanks to a method derived from BTD which would exploit the goods previously recorded. For instance, given a cluster $\mathcal{C}_i$, we can look for an assignment $\mathcal{A}$ on $\mathcal{C}_i$ such that for each son $\mathcal{C}_j$ of $\mathcal{C}_i$, $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ is a good. This method has a time complexity similar to BTD's one. However, it is clear that, in practice, it performs fewer nodes and constraint checks than BTD (except in the case where there is a single cluster). This method is better than the first, but it still seems too expensive because BTD may record a lot of goods. So for efficiency reasons, we must restrict the number of goods which are liable to be exploited for guiding the search for a solution. The ideal case would be to keep a single good per intersection $\mathcal{C}_i \cap \mathcal{C}_j$. In fact, this ideal case can be reached if we memorize some additional informations when we record or use a good.

Keeping a single good per intersection means that for each intersection, we keep the good which participates in an optimal solution. The main difficulty comes from the forward-jumps which may occur during the search. Indeed, when BTD uses a good of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$, it does not visit again the subproblem rooted in $\mathcal{C}_j$. So it does not check the goods of $\mathcal{C}_j$ with respect to any son of $\mathcal{C}_j$. For instance, by using a good on $\mathcal{C}_1 \cap \mathcal{C}_2$, BTD does not check the goods of $\mathcal{C}_2$ with respect to $\mathcal{C}_3$. Therefore, when BTD records a new good g of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$, it must also memorize, for each son $\mathcal{C}_k$ of $\mathcal{C}_j$, the good on $\mathcal{C}_j \cap \mathcal{C}_k$ which is exploited for building the current good g. By applying recursively this concept, we keep exactly one good per intersection. Thanks to a suitable data structure, these additional recordings do not change the space and time complexities of BTD.

Then, for computing a solution, we first assign the variables which appear in at least one intersection $\mathcal{C}_i \cap \mathcal{C}_j$ with the value they have in the corresponding good. We note $\mathcal{U}_{\mathcal{C}_i}$ the set of unassigned variables of $\mathcal{C}_i$. Clearly, we have $\mathcal{U}_{\mathcal{C}_i} = \mathcal{C}_i - (\mathcal{C}_{p(i)} \cup \bigcup_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \mathcal{C}_j)$. For each cluster $\mathcal{C}_i$, we consider the subproblem defined by the subgraph $(\mathcal{C}_i, C_{\mathcal{C}_i})$ of the constraint graph (X, C) with $C_{\mathcal{C}_i} = C \cap (\mathcal{C}_i \times (\mathcal{C}_i \backslash \mathcal{C}_{p(i)}))$. For each subproblem, only the variables of $\mathcal{U}_{\mathcal{C}_i}$ must be assigned. Moreover, we can solve independently each subproblem since each intersection $\mathcal{C}_i \cap \mathcal{C}_j$ is a separator of the graph (X, C). Then, for each subproblem $\mathcal{C}_i$, the initial lower bound (if we use BB) is defined by $|\{c = \{x, y\} \in C_{\mathcal{C}_i} | \exists \mathcal{C}_j \in Sons(\mathcal{C}_i), x \in$

$\mathcal{C}_i$ *and* $y \in \mathcal{C}_i \cap \mathcal{C}_j$ *and* $\mathcal{A}$ *violates* $c\}| + \sum_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \alpha_{\mathcal{C}_j}$ where $\mathcal{A} = \bigcup_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \mathcal{A}_{\mathcal{C}_j}$ and $(\mathcal{A}_{\mathcal{C}_j}, \alpha_{\mathcal{C}_j})$ is the good of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$. Straightforwardly, the two terms of the previous sum involve different constraints. Regarding the upper bound, it is simply $\alpha_{\mathcal{C}_i}$. The time-complexity of this method is, in the worst case, $O(nmd^k)$ where k is the size of the largest set $\mathcal{U}_{\mathcal{C}_i}$. Of course, finding an optimal solution still requires an enumeration, but our method limits this enumeration to its strict minimum.

Finally, if there is a single cluster, obviously, we have $k = n$ and providing an optimal solution requires an enumeration on n variables. To avoid such a redundant work, we add to BTD the ability to record the best known assignment for the variables of the root cluster. This trade-off slightly changes the space-complexity ($O(n+n.s.d^s)$ instead of $O(n.s.d^s)$) while saving many redundant works.

5 EXPERIMENTAL RESULTS

The second main question raised by BTD deals with its practical efficiency. Thanks to theoretical results and some intuitive ideas, we can think that BTD is efficient on some instances (in particular if they have a good structure) but no experimental result is presented in [22]. Indeed, first, the time-complexity bound of BTD is clearly better than one of enumerative methods since $w + 1 \leq n$. Then, by recording and using goods, BTD solves each subproblem only once, which allows BTD to save time and constraint checks. In contrast, BTD uses local lower and upper bounds, what may limit its pruning capacity. Consequently, some experimentations are required in order to really assess the practical interest of BTD.

This section provides empirical results on random and real-world instances. In both cases, the experimentations are realized on a linux-based PC with an Intel Pentium IV 2.4 GHz and 512Mb of memory. For random instances, we limit to half an hour the time spent for solving a given instance. So, sometimes, some instances may be unsolved. For these instances, we consider that the running time is half an hour. For each class of random instances, we solve 50 instances. The presented results are then the averages of results obtained for each instance. For both random and real-world instances, a tree-decomposition is computed by triangulating the constraint graph (thanks to the algorithm proposed in [19]) and by searching the maximal cliques of the triangulated constraint graph. From this tree-decomposition, we produce a tree-decomposition whose parameter s does not exceed 5 for random instances and 10 for real-world ones (see [7] for more details about this computation), which limits the memory requirements of BTD. For efficiency reasons, BTD is based on FC (instead of BB), what does not change any previous theoretical results. Inside each cluster, the variable heuristic for BTD is dom/deg which first chooses the variable x_i which minimizes the ratio $|d_{x_i}|/|\Gamma_{x_i}|$ with d_{x_i} the current domain of x_i and Γ_{x_i} its neighbour set. We do not use a particular value heuristic.

5.1 Random instances

We first assess the behaviour of BTD on classical random instances. In the classical CSP framework, BTD solves classical random instances as efficiently as the best classical enumerative algorithms [7], even if these instances do not present a priori good structural properties. Unfortunately, in the Max-CSP framework, in many cases, BTD can perform worse than algorithms like FC or FC-MRDAC. Indeed, as these instances do not have good structural properties, the clusters are often under-constrained and so BTD spends a lot of time to enumerate all possible solutions (because BTD exploits local bounds).

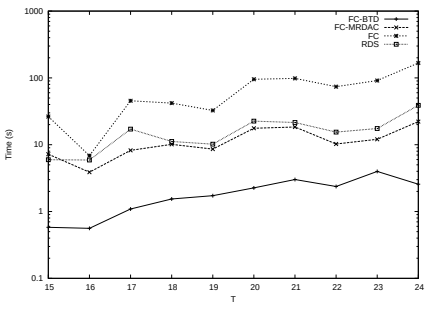


Figure 3. Mean run-time in seconds (with a log scale) for FC-BTD, FC-MRDAC, FC and RDS on class (30,5,10,T,5).

Then, we study the practical interest of BTD on structured random instances, for which one can expect that BTD provides better results than classical algorithms thanks to the exploitation of the structure. For these experiments, we use the model of structured random instances proposed in [7]. We consider several classes $(n, d, r_{max}, T, s_{max})$. An instance of the class $(n, d, r_{max}, T, s_{max})$ has n variables (each one having a domain of size d). Its constraint graph is a clique-tree such that the size of the largest clique is r_{max} and the size of the largest intersection is at most s_{max}. T denotes the tightness of each constraint. We compare BTD with FC, RDS [23] and FC-MRDAC [13]. RDS and FC use dom/deg as a variable heuristic (in a static way for RDS) and no particular value heuristic. For FC-MRDAC, we use the implementation provided by J. Larrosa [9].

Table 1. Mean run-time in seconds (respectively number of unsolved instances) for FC-BTD, FC-MRDAC, FC and RDS on random structured instances.

Classe	FC-BTD	FC-MRDAC	FC	RDS
(30,10,10,78,5)	18.9	280.6 (1)	124.2	154.9 (1)
(40,5,10,15,5)	2.7	144.6 (1)	149.3	152.8 (0)
(40,10,10,55,5)	7.6	318.5 (3)	77.6	503.1 (8)
(40,5,15,9,5)	15.5	160.3 (1)	64.2	109.8 (0)

We can first observe that, for the class of structured random instances presented in figure 3, BTD outperforms the three classical methods for any tightness. In effect, BTD solves these instances between 7 and 14 times faster than FC-MRDAC. This gain is obtained thanks to the exploitation of structural valued goods. Goods allow BTD to avoid visiting some redundant parts of the search. So, BTD achieves less constraint checks than each of the three algorithms. On the average, only a few hundred goods are produced, but each good is used up to 8,000 times. In order to confirm these results, we then compare BTD, FC-MRDAC, FC and RDS on four other classes of structured random instances (see table 1). We first note that, for some classes, FC-MRDAC or RDS are unable to solve every instance while BTD solves each of them in only a few seconds. Then we observe that BTD is significantly more efficient than FC-MRDAC, FC and RDS on these structured instances. It fully benefits from the structure. Indeed, like for the first class, only a few goods are recorded but their use allows BTD to prune a lot of branches and to achieve less constraint checks. Therefore, BTD makes a good trade-off between time and space since this recording does not require much memory.

Finally, we observe that the method proposed in section 4 for com-

puting an optimal solution requires at most a thousand additional constraint checks, which is insignificant with respect to the millions of constraint checks performed by BTD to compute the optimal cost.

5.2 Real-world instances

We experiment BTD on some real-world instances of the CELAR from the FullRLFAP archive[2]. These instances correspond to radio link frequency assignment problems (for more details, see [3]). Some of them can be easily expressed as binary Max-CSPs. We focus our study on the SUBCELAR class which contains five subproblems produced from the SCEN-06 instance (one of the hardest instances in the archive). We exploit the simplification proposed by T. Schiex [20]. It consists in removing the hard equality constraints and dividing by two the number of variables. By so doing, we obtain a smaller constraint graph and so a better tree-decomposition. For example, the smallest instance ($SUBCELAR_0$) has 16 variables and 57 constraints while the largest ($SUBCELAR_4$) has 22 variables and 131 constraints. Domains contain 36 or 44 values.

As shown in table 2, FC-BTD succeeds in solving all the SUBCELAR instances. These results are mostly due to the exploitation of structural goods. Indeed, on the average, BTD exploits each good between 9 and 261 times, which allows it to save many redundant works. For information, for these instances, computing an optimal solution from the optimal cost requires at most 3,270 constraint checks, which is insignificant with respect to the millions of constraint checks achieved for computing the optimal cost.

Comparing our results with previous ones (e.g. [13, 8, 12, 17, 11]) is quite difficult because the computer architectures are conceptually different and experimental protocols differ. For instance, [13] solves the SUBCELAR instances by using the optimal cost as initial upper bound while BTD does not exploit any initial upper bound. Nevertheless, for information, FC-MRDAC solves $SUBCELAR_2$ in about 23,000 s on a Sun Sparc 2. [11] takes also advantage of the problem structure to provide theoretical time and space complexity bounds but the experimental results are not convincing. In [17], an improved version of RDS obtains, on a Pentium IV 1.8 GHz based PC, either better results or worse ones than BTD's ones. Comparing BTD and the Koster's method [8], the best known method for solving CELAR instances, is not easy because this method exploits many pretreatments which reduce the problem size (and so the size of the constraint graph). Furthermore, we have not studied yet the influence of some structural parameters (like w or s) on the behaviour of BTD. So, by adding to BTD some pretreatments like Koster's ones or thanks to a better choice for some parameters, we can expect to improve the practical efficiency of BTD. In practice, these improvements are needed for solving larger and harder instances like SCEN-06.

Table 2. Results obtained by FC-BTD on SUBCELAR instances

Instance	Time (s)	# goods	# good uses (thousands)	# good checks (millions)
$SUBCELAR_0$	2.5	34,170	306	1.57
$SUBCELAR_1$	308	80,375	1,336	6.48
$SUBCELAR_2$	405	96,980	996	15.64
$SUBCELAR_3$	1,883	515,735	19,661	162.70
$SUBCELAR_4$	122,933	403,282	105,386	844.44

[2] we thank the Centre d'Electronique de l'Armement (France).

6 CONCLUSION AND FUTURE WORKS

In this paper, we have raised two questions about the BTD method. The first one concerns the construction of an optimal solution from the optimal cost provided by BTD. The second one deals with the practical efficiency of BTD. Then we have proposed an efficient method for computing such an optimal solution. Finally, we have shown the practical interest of BTD for solving instances with good structural properties. Indeed, BTD clearly outperforms FC-MRDAC, FC and RDS on random structured instances and it succeeds in solving all the SUBCELAR instances.

Regarding the future works, BTD can be improved by taking into account the constraints between unassigned variables (for instance by using arc-consistency [13, 4]). Then, studying the influence of some structural parameters on the behaviour of BTD can help us to optimize some choices about these parameters, which would allow BTD to obtain better results. Finally, we can add to BTD some preatreatments like Koster's ones [8].

REFERENCES

[1] S. Arnborg, D. Corneil, and A. Proskurowski, 'Complexity of finding embeddings in a k-tree', *SIAM Journal of Discrete Mathematics*, **8**, 277–284, (1987).
[2] S. Bistarelli, U. Montanari, and F. Rossi, 'Constraint solving over semirings', in *Proc. of IJCAI*, pp. 624–630, (1995).
[3] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners, 'Radio Link Frequency Assignment', *Constraints*, **4**, 79–89, (1999).
[4] M. Cooper and T. Schiex, 'Arc consistency for soft constraints', *Artificial Intelligence*, **154**, 199–227, (2004).
[5] R. Dechter and Y. El Fattah, 'Topological Parameters for Time-Space Tradeoff', *Artificial Intelligence*, **125**, 93–118, (2001).
[6] E. Freuder and R. Wallace, 'Partial constraint satisfaction', *Artificial Intelligence*, **58**, 21–70, (1992).
[7] P. Jégou and C. Terrioux, 'Hybrid backtracking bounded by tree-decomposition of constraint networks', *Artificial Intelligence*, **146**, 43–75, (2003).
[8] A. Koster, *Frequency Assignment - Models and Algorithms*, Ph.D. dissertation, University of Maastricht, November 1999.
[9] J. Larrosa. http://www.lsi.upc.es/ larrosa/pfc-mrdac.
[10] J. Larrosa, 'On arc and node consistency', in *Proc. of AAAI*, pp. 48–53, (2002).
[11] J. Larrosa and R. Dechter, 'Boosting Search with Variable Elimination in Constraint Optimization and Constraint Satisfaction Problems', *Constraints*, **8**(3), 303–326, (2003).
[12] J. Larrosa, P. Meseguer, and M. Sánchez, 'Pseudo-Tree Search with Soft Constraints', in *Proc. of ECAI*, pp. 131–135, (2002).
[13] J. Larrosa, P. Meseguer, and T. Schiex, 'Maintaining reversible DAC for Max-CSP', *Artificial Intelligence*, **107**(1), 149–163, (1999).
[14] J. Larrosa and T. Schiex, 'In the quest of the best form of local consistency for Weighted CSP', in *Proc. of IJCAI*, pp. 239–244, (2003).
[15] P. Meseguer and M. Sánchez, 'Tree-based Russian Doll Search', in *Proc. of CP Workshop on soft constraint*, (2000).
[16] P. Meseguer and M. Sánchez, 'Specializing Russian Doll Search', in *Proc. of CP*, pp. 464–478, (2001).
[17] P. Meseguer, M. Sánchez, and G. Verfaillie, 'Opportunistic Specialization in Russian Doll Search', in *Proc. of CP*, pp. 264–279, (2002).
[18] N. Robertson and P.D. Seymour, 'Graph minors II: Algorithmic aspects of tree-width', *Algorithms*, **7**, 309–322, (1986).
[19] D. Rose, R. Tarjan, and G. Lueker, 'Algorithmic Aspects of Vertex Elimination on Graphs', *SIAM Journal on computing*, **5**, 266–283, (1976).
[20] T. Schiex. http://www.inra.fr/bia/t/schiex/doc/celare.html.
[21] T. Schiex, H. Fargier, and G. Verfaillie, 'Valued Constraint Satisfaction Problems: hard and easy problems', in *Proc. of IJCAI*, pp. 631–637, (1995).
[22] C. Terrioux and P. Jégou, 'Bounded backtracking for the valued constraint satisfaction problems', in *Proc. of CP*, pp. 709–723, (2003).
[23] G. Verfaillie, M. Lemaître, and T. Schiex, 'Russian Doll Search for Solving Constraint Optimization Problems', in *Proc. of AAAI*, pp. 181–187, (1996).

Bibliographie

Bibliographie

Third International CSP Solver Competition. http://www.cril.univ-artois.fr/CPAI08, 2008.

Fourth International CSP Solver Competition. http://www.cril.univ-artois.fr/CPAI09, 2009.

Eyal Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 7–15, 2001.

Eyal Amir. Approximation Algorithms For Treewidth. *Algorithmica*, 56(4) :448–479, 2010.

Kamal Amroun, Zineb Habbas, et Wassila Aggoune-Mtalaa. A compressed Generalized Hypertree Decomposition-based solving technique for non-binary Constraint Satisfaction Problems. *AI Communications*, 29(2) :371–392, 2016.

Krzysztof R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

Stefan Arnborg, Dereck Corneil, et Andrzej Proskuroswki. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2) :277–284, 1987.

Jean-François Baget et Yannic S. Tognetti. Backtracking Throught Biconnected Components of a Constraint Graph. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 291–296, 2001.

Roberto J. Bayardo et Daniel P. Miranker. A Complexity Analysis of Space-Bounded Learning Algorithms for the Constraints Satisfaction Problem. In *Proceedings of 13th National Conference on Artificial Intelligence (AAAI)*, pages 298–304, 1996.

Catriel Beeri, Ronald Fagin, David Maier, et Mihalis Yannakakis. On the Desirability of Acyclic Database Schemes. *Journal of the ACM*, 30(3) :479–513, 1983.

Claude Berge. Les problèmes de colorations en théorie des graphes. In *Publ. Inst. Statist. Univ. Paris 9*, pages 123–160, 1960.

Claude Berge. Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 10(1) :114, 1961.

Claude Berge. *Graphs and Hypergraphs*. Elsevier, 1973.

Philip A. Bernstein et Nathan Goodman. The power of natural semijoins. *SIAM Journal on Computing*, 10(4) : 751–771, 1981.

Anne Berry. A Wide-Range Efficient Algorithm for Minimal Triangulation. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1999.

Umberto Bertelè et Francesco Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

Christian Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1) :179–190, 1994.

Christian Bessière et Jean-Charles Régin. MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ ?) on Hard Problems. In *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (CP)*, pages 61–75, 1996.

Christian Bessière, Assef Chmeiss, et Lakhdar Saïs. Neighborhood-based variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 565–569, 2001.

Christian Bessière, Pedro Meseguer, Eugene C. Freuder, et Javier Larrosa. On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141(1/2) :205–224, 2002.

Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, et Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2) :165–185, 2005.

Christian Bessière, Kostas Stergiou, et Toby Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 172(6-7) :800–822, 2008.

Armin Biere, Marijn Heule, Hans van Maaren, et Toby Walsh. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, et Gérard Verfaillie. Semiring-based CSPs and valued CSPs : Basic properties and comparison. *LNCS*, 1106, 1996.

Vincent Bouchitté, Dieter Kratsch, Haiko Müller, et Ioan Todinca. On treewidth approximations. *Discrete Applied Mathematics*, 136(2-3) :183–196, 2004.

Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, et Lakhdar Saïs. Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 146–150, 2004.

Karim Boutaleb, Philippe Jégou, et Cyril Terrioux. (No)good Recording and ROBDDs for Solving Structured (V)CSPs. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 297–304, 2006a.

Karim Boutaleb, Philippe Jégou, et Cyril Terrioux. Optimizing the space to extend the tractability of (valued) structured CSP. In *Proceedings of the Annual Workshop of ERCIM on Constraint Solving and Constraint Logic Programming (CSCLP'06)*, pages 85–99, 2006b.

Karim Boutaleb, Philippe Jégou, et Cyril Terrioux. Storing learnt (no)goods in ROBDDs for solving structured CSPs. In *Proceedings of the AAAI Workshop on Learning for Search*, pages 65–71, 2006c.

Daniel Brélaz. New Methods to Color Vertices of a Graph. *Communications of the ACM*, 22(4) :251–256, 1979.

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, 1986.

Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3) :293–318, 1992.

Binh-Minh Bui-Xuan, Jan Arne Telle, et Martin Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412(39) :5187–5204, 2011.

Clément Carbonnel et Martin C. Cooper. Tractability in constraint satisfaction problems : a survey. *Constraints*, 21(2) :115–144, 2016.

Xinguang Chen. *A Theoretical Comparison of Selected CSP Solving and Modeling Techniques*. PhD thesis, University of Alberta, 2000.

Assef Chmeiss et Philippe Jégou. A generalization of chordal graphs and the maximum clique problem. *Information Processing Letters*, 62 :111–120, 1997.

Assef Chmeiss et Philippe Jégou. Efficient Path-Consistency Propagation. *International Journal of Artificial Intelligence Tools*, 7 :121–142, 1998.

Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul D. Seymour, et Kristina Vušković. Recognizing Berge Graphs. *Combinatorica*, 25(2) :143–186, 2005.

Maria Chudnovsky, Neil Robertson, Paul D. Seymour, et Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164(1) :51–229, 2006.

David A. Cohen. A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 807–811, 2003.

David A. Cohen et Peter G. Jeavons. *The Complexity of Constraint Languages*, chapter 8, pages 245–280. Handbook of Constraint Programming (Rossi et al., 2006). Elsevier, 2006.

David A. Cohen, Martin C. Cooper, Martin J. Green, et Dániel Marx. On guaranteeing polynomially bounded search tree size. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 160–171, 2011.

Martin C. Cooper. An Optimal k-Consistency Algorithm. *Artificial Intelligence*, 41, 1989.

Martin C. Cooper, David A. Cohen, et Peter G. Jeavons. Characterising Tractable Constraints. *Artificial Intelligence*, 65(2) :347–361, 1994.

Martin C. Cooper, Peter G. Jeavons, et András Z. Salamon. Hybrid tractable CSPs which generalize tree structure. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI)*, pages 530–534, 2008.

Martin C. Cooper, Peter G. Jeavons, et András Z. Salamon. Generalizing constraint satisfaction on trees : hybrid tractability and variable elimination. *Artificial Intelligence*, 174 :570–584, 2010.

Martin C. Cooper, Achref El Mouelhi, Cyril Terrioux, et Bruno Zanuttini. On Broken Triangles. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 9–24, 2014. Best technical paper.

Martin C. Cooper, Aymeric Duchein, et Guillaume Escamocher. Broken triangles revisited. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, pages 58–73, 2015a.

Martin C. Cooper, Philippe Jégou, et Cyril Terrioux. A Microstructure-based Family of Tractable Classes for CSPs. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, pages 74–88, 2015c.

Martin C. Cooper, Aymeric Duchein, Achref El Mouelhi, Guillaume Escamocher, Cyril Terrioux, et Bruno Zanuttini. Broken Triangles : From Value Merging to a Tractable Class of General-Arity Constraint Satisfaction Problems. *Artificial Intelligence*, 234 :196–218, 2016a.

Martin C. Cooper, Achref El Mouelhi, et Cyril Terrioux. Extending Broken Triangles and Enhanced Value-merging. In *Proceedings of the 22nd International Conference on Principle and Practice of Constraint Programming (CP)*, pages 173–188, 2016b.

Bruno Courcelle et Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3) :77–114, 2000.

Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126 :5–41, 2001.

Simon de Givry, Thomas Schiex, et Gérard Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency In Weighted CSP. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, pages 22–27, 2006.

Romuald Debruyne. A Property of Path Inverse Consistency Leading to an Optimal PIC Algorithm. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, pages 88–92, 2000.

Romuald Debruyne et Christian Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 412–417, 1997a.

Romuald Debruyne et Christian Bessière. From Restricted Path Consistency to Max-Restricted Path Consistency. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP)*, pages 312–326, 1997b.

Romuald Debruyne et Christian Bessière. Domain Filtering Consistencies. *Journal of Artificial Intelligence Research (JAIR)*, 14 :205–230, 2001.

Rina Dechter. Learning While Searching in Constraint-Satisfaction-Problems. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*, page 178–183, 1986.

Rina Dechter. Enhancement Schemes for Constraint Processing : Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41 :273–312, 1990.

Rina Dechter. From Local to Global Consistency. *Artificial Intelligence*, 55(1) :87–108, 1992.

Rina Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

Rina Dechter. *Tractable Structures for Constraint Satisfaction Problems*, chapter 7, pages 209–244. Handbook of Constraint Programming (Rossi et al., 2006). Elsevier, 2006.

Rina Dechter et Robert Mateescu. The Impact of AND/OR Search Spaces on Constraint Satisfaction and Counting. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 731–736, 2004.

Rina Dechter et Robert Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171 : 73–106, 2007.

Rina Dechter et Itay Meiri. Experimental Evaluation of Preprocessing Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 68(2) :211–241, 1994.

Rina Dechter et Judea Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34 :1–38, 1987.

Rina Dechter et Judea Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38(3) :353–366, 1989.

Yves Deville, Olivier Barette, et Pascal Van Hentenryck :. Constraint Satisfaction over Connected Row Convex Constraints. *Artificial Intelligence*, 109(1-2) :243–271, 1999.

Reinhard Diestel et Malte Müller. Connected tree-width. *arXiv*, 1211.7353v2, 2014.

Mehmet Dincbas, Pascal Van Hentenryck, Helmut Simonis, Abderrahmane Aggoun, Thomas Graf, et Françoise Berthier. The Constraint Logic Programming Language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 693–702, 1988.

Vida Dujmović, Gasper Fijavž, Gwenaël Joret, Thom Sulanke, et David R. Wood. On the maximum number of cliques in a graph embedded in a surface. *European Journal of Combinatorics*, 32(8) :1244–1252, 2011.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. On the Efficiency of Backtracking Algorithms for Binary Constraint Satisfaction Problems. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2012a.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. Sur la complexité des algorithmes de backtracking et quelques nouvelles classes polynomiales pour CSP. In *Actes des* $8^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 160–169, 2012b.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Microstructures for CSPs with Constraints of Arbitrary Arity. In *10th Symposium on Abstraction, Reformulation, and Approximation (SARA)*, 2013a.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 947–954, 2013b.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Sur une classe polynomiale hybride pour les CSP d'arité quelconque. In *Actes des* $9^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 237–247, 2013c.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. Some New Tractable Classes of CSPs and their Relations with Backtracking Algorithms. In *Proceedings of the 10th International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (CPAIOR)*, pages 61–76, 2013d.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Different Classes of Graphs to Represent Microstructures for CSPs. In *Graph Structures for Knowledge Representation and Reasoning, LNAI 8323*, pages 21–38. Springer, 2014a.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Hidden Tractable Classes : from Theory to Practice. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2014b.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. Microstructures pour CSP d'arité quelconque. In *Actes des* $10^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 193–202, 2014c.

Achref El Mouelhi, Philippe Jégou, Cyril Terrioux, et Bruno Zanuttini. Some New Tractable Classes of CSPs and their Relations with Backtracking Algorithms. In *Fourth International Workshop on the Cross-Fertilization Between CSP and SAT*, 2014d.

Achref El Mouelhi, Philippe Jégou, et Cyril Terrioux. A Hybrid Tractable Class for Non-Binary CSPs. *Constraints*, 20(4) :383–413, 2015a.

Eugene C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21(11) :958–966, 1978.

Eugene C. Freuder. A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM*, 29 (1) :24–32, 1982.

Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI)*, pages 4–9, 1990.

Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)*, pages 227–233, 1991.

Eugene C. Freuder et Charles D. Elfe. Neighborhood inverse consistency preprocessing. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 202–208, 1996.

Eugene C. Freuder et Michael J. Quinn. Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1076–1078, 1985.

Daniel Frost et Rina Dechter. Dead-End Driven Learning. In *Proceedings of he 12th National Conference on Artificial Intelligence (AAAI)*, pages 294–300, 1994.

Jakub Gajarský, Michael Lampis, et Sebastian Ordyniak. Parameterized Algorithms for Modular-Width. In *Proceedings of the 8th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 163–176, 2013.

Michael R. Garey et David S. Johnson. *Computer and Intractability*. Freeman, 1979.

John Gaschnig. Performance Measurement and Analysis of Certain Search Algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, 1979.

Fănică 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.

Pieter A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 31–35, 1992.

Matthew L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1 :25–46, 1993.

Solomon W. Golomb et Leonard D. Baumert. Backtrack programming. *Journal of the ACM*, 12 :516–524, 1965.

Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

Carla P. Gomes, Bart Selman, Nuno Crato, et Henry A. Kautz. Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning*, 24(1/2) :67–100, 2000.

Georg Gottlob, Nicola Leone, et Francesco Scarcello. Hypertree Decompositions and Tractable Queries. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 21–32, 1999b.

Georg Gottlob, Nicola Leone, et Francesco Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :243–282, 2000.

Marc H Graham. On the universal relation. Technical report, University of Toronto, 1979.

Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.

Martin Grohe et Dániel Marx. Constraint Solving via Fractional Edge Covers. *ACM Transactions on Algorithms*, 11(1) :1–20, 2014.

Zineb Habbas, Kamal Amroun, et Daniel Singer. Generalized Hypertree Decomposition for solving non binary CSP with compressed table constraints. *RAIRO - Operations Research*, 50(2) :241–267, 2016.

Djamal Habet, Lionel Paris, et Cyril Terrioux. A Tree Decomposition Based Approach to Solve Structured SAT Instances. In *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 115–122, 2009a.

Matthias Hamann et Daniel Weißauer. Bounding connected tree-width. *arXiv*, 1503.01592, 2015.

Ching-Chih Han et Chia-Hoang Lee. Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence*, 36 :125–130, 1988.

Robert M. Haralick et Gordon L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313, 1980.

William D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Stanford University, 1995.

Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

Holger H. Hoos et Thomas Stützle. *Stochastic Local Search : Foundations and Applications*. Elsevier / Morgan Kaufmann, 2004.

Holger H. Hoos et Edward P. K. Tsang. *Local Search Methods*, chapter 5, pages 135–167. Handbook of Constraint Programming (Rossi et al., 2006). Elsevier, 2006.

Joey Hwang et David G. Mitchell. 2-Way vs. d-Way Branching for CSP. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 343–357, 2005.

Philippe Janssen, Philippe Jégou, Bernard Nouguier, et Marie-Catherine Vilarem. A filtering process for general constraint satisfaction problems : achieving pairwise-consistency using an associated binary representation. In *Proceedings of the IEEE Workshop on Tools for Artificial Intelligence*, pages 420–427, 1989.

Peter G. Jeavons et Martin C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2) : 327–339, 1995.

Peter G. Jeavons, David A. Cohen, et Marc Gyssens. Closure properties of constraints. *Journal of the ACM*, 44 : 527–548, 1997.

Philippe Jégou. Cyclic-clustering : A compromise between tree-clustering and cycle-cutset method for improving search efficiency. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI)*, pages 369–371, 1990.

Philippe 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, 1991.

Philippe Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI)*, pages 731–736, 1993a.

Philippe Jégou. On the Consistency of General Constraint-Satisfaction Problems. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI)*, pages 114–119, 1993b.

Philippe Jégou et Cyril Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003a.

Philippe Jégou et Cyril Terrioux. Recherche arborescente bornée pour la résolution de CSP valués. *Journal électronique d'intelligence artificielle (JEDAI)*, 3(28), 2004a.

Philippe Jégou et Cyril Terrioux. Decomposition and Good Recording. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 196–200, 2004b.

Philippe Jégou et Cyril Terrioux. A time-space trade-off for constraint networks decomposition. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 234–239, 2004c.

Philippe Jégou et Cyril Terrioux. A New Filtering Based on Decomposition of Constraint Sub-Networks. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 263–270, 2010a.

Philippe Jégou et Cyril Terrioux. Structural Consistency : A New Filtering Approach for Constraint Networks. In *Proceedings of the 1st Workshop on Constraint Reasoning and Graphical Structures*, 2010b.

Philippe Jégou et Cyril Terrioux. Bag-Connected Tree-Width : A New Parameter for Graph Decomposition. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2014a.

Philippe Jégou et Cyril Terrioux. Structural Consistency : A New Filtering Approach for Constraint Networks. In *Graph Structures for Knowledge Representation and Reasoning, LNAI 8323*, pages 74–91. Springer, 2014b.

Philippe Jégou et Cyril Terrioux. Combining Restarts, Nogoods and Decompositions for Solving CSPs. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, pages 465–470, 2014c.

Philippe Jégou et Cyril Terrioux. Tree-Decompositions with Connected Clusters for Solving Constraint Networks. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 407–423, 2014d.

Philippe Jégou et Cyril Terrioux. The Extendable-Triple Property : A New CSP Tractable Class beyond BTP. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pages 3746–3754, 2015.

Philippe Jégou et Cyril Terrioux. Combining Restarts, Nogoods and Bag-Connected Decompositions for Solving CSPs. *Constraints*, 2016. À paraître.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 777–781, 2005a.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Sur la génération et l'exploitation de décompositions pour la résolution de réseaux de contraintes. In *Actes des* $1^{ère}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 149–158, 2005b.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. An extension of complexity bounds and dynamic heuristics for tree-decompositions of CSP. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 741–745, 2006a.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Strategies and Heuristics for Exploiting Tree-decompositions of Constraint Networks. In *Proceedings of the Inference methods based on graphical structures of knowledge (WIGSK'06), ECAI workshop*, pages 13–18, 2006b.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic heuristics for branch and bound search on tree-decomposition of weighted csps. In *Proceedings of the Eighth International Workshop on Preferences and Soft Constraints (Soft-2006)*, pages 63–77, 2006c.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre. In *Actes des* $2^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 219–228, 2006d.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 112–117, 2007a.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic Management of Heuristics for Solving Structured CSPs. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 364–378, 2007b.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Dynamic Heuristics for Branch and Bound on Tree-Decomposition of Weighted CSPs. In *Trends in Constraint Programming*, chapter 20, pages 317–332. ISTE, 2007d.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. A New Evaluation of Forward Checking and its Consequences on Efficiency of Tools for Decomposition of CSPs. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 486–490, 2008a.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Extending to Soft and Preference Constraints a Framework for Solving Efficiently Structured Problems. In *Proceedings of the 4th Multidisciplinary Workshop on Advances in Preference Handling (M-PREF 2008), AAAI workshop*, pages 61–66, 2008b.

Philippe Jégou, Samba Ndojh Ndiaye, et Cyril Terrioux. Combined Strategies for Decomposition-based Methods for solving CSPs. In *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 184–192, 2009a.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. De nouvelles approches pour la décomposition de réseaux de contraintes. In *Actes des* $11^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 140–149, 2015a.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. An Algorithmic Framework for Decomposing Constraint Networks. In *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1–8, 2015b.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. Towards a Dynamic Decomposition of CSPs with Separators of Bounded Size. In *Proceedings of the 22nd International Conference on Principle and Practice of Constraint Programming (CP)*, pages 298–315, 2016.

Philippe Jégou, Hanan Kanso, et Cyril Terrioux. Vers une décomposition dynamique des réseaux de contraintes. In *Actes des* $12^{\text{ème}}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 123–132, 2016.

Jisu Jeong, Sigve Hortemo Sæther, et Jan Arne Telle. Maximum matching width : new characterizations and a fast algorithm for dominating set. In *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 212–223, 2015.

Narendra Jussien, Romuald Debruyne, et Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 249–261, 2000.

George Katsirelos et Fahiem Bacchus. Unrestricted Nogood Recording in CSP Search. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 873–877, 2003.

George Katsirelos et Fahiem Bacchus. Generalized NoGoods in CSPs. In *Proceedings of 20th National Conference on Artificial Intelligence (AAAI)*, pages 390–396, 2005.

Christophe Lecoutre. *Constraint Networks - Techniques and Algorithms*. ISTE/Wiley, 2009.

Christophe Lecoutre et Patrick Prosser. Maintaining singleton arc-consistency. In *Proceedings of the 3rd International Workshop on Constraint Propagation And Implementation*, pages 47–61, 2006.

Christophe Lecoutre, Stéphane Cardon, et Julien Vion. Path Consistency by Dual Consistency. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 438–452, 2007a.

Christophe Lecoutre, Stéphane Cardon, et Julien Vion. Conservative Dual Consistency. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 237–242, 2007b.

Christophe Lecoutre, Lakhdar Saïs, Sébastien Tabary, et Vincent Vidal. Recording and Minimizing Nogoods from Restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1(3-4) :147–167, 2007c.

Christophe Lecoutre, Chavalit Likitvivatanavong, Scott G. Shannon, Roland H. C. Yap, et Yuanlin Zhang. Maintaining Arc Consistency with Multiple Residues. *Constraint Programming Letters (CPL)*, 2, 2008.

Christophe Lecoutre, Stéphane Cardon, et Julien Vion. Second-Order Consistencies. *Journal of Artificial Intelligence Research (JAIR)*, 40, 2011.

David Lesaint, Nader Azarmi, R. Laithwaite, et P. Walker. Engineering Dynamic Scheduler for Work Manager. *BT Technology Journal*, 16(3), 1998.

Paolo Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1-2), 2000.

Chavalit Likitvivatanavong et Roland H. C. Yap. Many-to-many interchangeable sets of values in csps. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC)*, pages 86–91, 2013.

Michael Luby, Alistair Sinclair, et David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4) :173–180, 1993.

Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8 :99–118, 1977.

João P. Marques Silva, Inês Lynce, et Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, pages 131–153. Handbook of Satisfiability. 2009.

Pedro Meseguer, Francesca Rossi, et Thomas Schiex. *Soft Constraints*, chapter 9, pages 281–328. Handbook of Constraint Programming (Rossi et al., 2006). Elsevier, 2006.

David G. Mitchell. Resolution and Constraint Satisfaction. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 555–569, 2003.

Roger Mohr et Thomas C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28(2), 1986.

Ugo Montanari. Networks of Constraints : Fundamental Properties and Applications to Picture Processing. *Artificial Intelligence*, 7 :95–132, 1974.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, et Sharad Malik. Chaff : Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 530–535, 2001.

Malte Müller. Connected tree-width. *arXiv*, 1211.7353, 2012.

Wady Naanaa. Unifying and extending hybrid tractable classes of csps. *Journal of Experimental and Theoretical Artificial Intelligence*, 25(4) :407–424, 2013.

Wady Naanaa. Extending the broken triangle property tractable class of binary csps. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence (SETN 2016)*, 2016.

Bernard Nadel. *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*, pages 287–342. In *Search in Artificial Intelligence*. Springer-Verlag, 1988.

Samba Ndojh Ndiaye et Cyril Terrioux. Un schéma générique d'algorithmes énumératifs avec (no)good recording pour la résolution bornée de CSP. In *Actes des* $3^{ème}$ *Journées Francophones de Programmation par Contraintes (JFPC)*, pages 209–218, 2007a.

Samba Ndojh Ndiaye et Cyril Terrioux. A generic bounded backtracking framework for solving CSPs. In *Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming (CSCLP)*, pages 107–121, 2007b.

Samba Ndojh Ndiaye, Philippe Jégou, et Cyril Terrioux. Extending to Soft and Preference Constraints a Framework for Solving Efficiently Structured Problems. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 299–306, 2008.

Jaroslav Nešetřil et Patrice Ossona de Mendez. *Bounded Height Trees and Tree-Depth*, chapter 6, pages 245–280. Sparsity - Graphs, Structures, and Algorithms. Algorithms and combinatorics 28. Springer, 2012.

Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory*, Series B 95(1) :79–100, 2005.

Charles Sanders Peirce, Charles Hartshorne, et Paul Weiss. *Collected Papers of Charles Sanders Peirce*, volume volume III. Harvard University Press, 1933.

Justyna Petke et Peter G. Jeavons. Tractable benchmarks. In *Proceedings of the 8th International Workshop on Constraint Modelling and Reformulation (ModRef)*, pages 102–116, 2009.

Cédric Pinto et Cyril Terrioux. A New Method for Computing Suitable Tree-decompositions with Respect to Structured CSP Solving. In *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 491–495, 2008.

Cédric Pinto et Cyril Terrioux. A generalized Cyclic-Clustering Approach for Solving Structured CSPs. In *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 724–728, 2009a.

Cédric Pinto et Cyril Terrioux. Une généralisation de l'approche Cyclic-Clustering pour la résolution. In *Actes des $5^{ème}$ Journées Francophones de Programmation par Contraintes (JFPC)*, pages 5–14, 2009b.

Patrick Prosser. Hybrid Algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9 :268–299, 1993.

Patrick Prosser. MAC-CBJ : maintening arc consistency with conflict-directed backjumping, 1995.

Lisa Purvis et Peter G. Jeavons. Constraint tractability theory and its application to the product development process for a constraint-based scheduler. In *Proceedings of the 1st International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP)*, page 63–79, 1999.

Philippe Refalo. Understanding and Improving the MAC Algorithm. In *Proceedings of the 10th International Conference on Principles and Practices of Constraint Programming (CP)*, pages 557–571, 2004.

Neil Robertson et Paul D. Seymour. Graph minors II : Algorithmic aspects of treewidth. *Algorithms*, 7 :309–322, 1986.

Donald J. Rose. A graph theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

Donald J. Rose, Robert Endre Tarjan, et George S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on computing*, 5 :266–283, 1976.

Francesca Rossi, Charles J. Petrie, et Vasant Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI)*, pages 550–556, 1990.

Francesca Rossi, Peter van Beek, et Toby Walsh. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

Daniel Sabin et Eugene C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of 11th European Conference on Artificial Intelligence (ECAI)*, pages 125–129, 1994.

Daniel Sabin et Eugene C. Freuder. Understanding and Improving the MAC Algorithm. In *Proceedings of the 3rd International Conference on Principles and Practices of Constraint Programming (CP)*, pages 167–181, 1997.

András Z. Salamon et Peter G. Jeavons. Perfect Constraints Are Tractable. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 524–528, 2008.

Martí Sánchez, Sylvain Bouveret, Simon de Givry, Federico Heras, Philippe Jégou, Javier Larrosa, Samba Ndojh Ndiaye, Emma Rollon, Thomas Schiex, Cyril Terrioux, Gérard Verfaillie, et Matthias Zytnicki. Max-CSP competition 2008 : toulbar2 solver description. In *Proceedings of the 3rd CSP Solver Competition, CP workshop*, pages 63–70, 2008.

Thomas Schiex et Gérard Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In *Proceedings of the 5th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 48–55, 1993.

Thomas Schiex et Gérard Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, 3(2) :187–207, 1994.

Thomas Schiex, Hélène Fargier, et Gérard Verfaillie. Valued Constraint Satisfaction Problems : hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 631–637, 1995.

Paul D. Seymour et Robin Thomas. Call Routing and the Ratcatcher. *Combinatorica*, 14(2) :217–241, 1994.

Richard M. Stallman et Gerald J. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artifical Intelligence*, 9(2) :135–196, 1977.

Kostas Stergiou. Restricted Path Consistency Revisited. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP)*, pages 419–428, 2015.

Kostas Stergiou et Toby Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI)*, pages 163–168, 1999.

Robert Endre Tarjan et Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3) :566–579, 1984.

Cyril Terrioux. Cooperative Search and Nogood Recording. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 260–265, 2001a.

Cyril Terrioux. Recherche Coopérative et Nogood Recording. In *Actes des Journées Francophones de Programmation en Logique et par Contraintes (JFPLC)*, pages 173–187, 2001b.

Cyril Terrioux. *Approches structurelles et coopératives pour la résolution des problèmes de satisfaction de contraintes*. PhD thesis, Université de Provence, 2002.

Cyril Terrioux et Philippe Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 709–723, 2003.

Peter van Beek et Rina Dechter. On the minimality and decomposability of row-convex constraint networks. *Journal of the ACM*, 42(3) :543–561, 1995.

Toby Walsh. Search in a Small World. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1172–1177, 1999.

Ryan Williams, Carla P. Gomes, et Bart Selman. Backdoors to typical case complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1173–1178, 2003.

Paul Wollan. The structure of graphs not admitting a fixed immersion. *Journal of Combinatorial Theory*, Series B 110 :47–66, 2015.

David R. Wood. On the maximum number of cliques in a graph. *Graphs and Combinatorics*, 23 :337–352, 2007.

Yuanlin Zhang, Roland H. C. Yap, et Joxan Jaffar. Functional Elimination and 0/1/All Constraints. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI)*, pages 175–180, 1999.