

---

# Recherche arborescente bornée pour la résolution de CSP valués

---

**Philippe Jégou**

LSIS-CNRS,

Université d'Aix-Marseille 3

Av. Escadrille Normandie-Niemen

13397 Marseille Cedex 20

email : [philippe.jegou@univ.u-3mrs.fr](mailto:philippe.jegou@univ.u-3mrs.fr)

**Cyril Terrioux**

LSIS-CNRS,

Université d'Aix-Marseille 3

Av. Escadrille Normandie-Niemen

13397 Marseille Cedex 20

email : [cyril.terrioux@univ.u-3mrs.fr](mailto:cyril.terrioux@univ.u-3mrs.fr)

## Résumé

Nous proposons une nouvelle méthode pour résoudre des problèmes de satisfaction de contraintes valués. Cette méthode est basée à la fois sur des techniques de backtracking (branch and bound) et sur la notion de décomposition arborescente d'un réseau de contraintes valué. Elle vise à bénéficier des avantages des deux approches, à savoir l'efficacité pratique de l'énumération et la garantie de bornes de complexité en temps offerte par les méthodes structurales. En effet, la complexité en temps est en  $O(d^{w^++1})$  avec  $w^+$  une approximation de la tree-width du graphe de contrainte et  $d$  la taille du plus grand domaine.

L'obtention d'une telle borne de complexité repose sur l'exploitation de valuations locales qui fournissent des bornes optimales sur des sous-problèmes définis via la décomposition arborescente. De plus, nous associons ces valuations locales à des affectations partielles appelées "goods structurels valués". La mémorisation et l'emploi de ces goods permettent d'une part d'éviter de réexplorer plusieurs fois certaines parties de l'espace de recherche, et d'autre part, de limiter la quantité de mémoire qui est généralement requise par des méthodes basées sur la programmation dynamique. Enfin, cette méthode constitue une extension naturelle de la méthode BTD [Jégou et Terrioux, 2002; 2003] proposée dans le cadre des CSP classiques.

## 1 Introduction

De nombreux problèmes réels peuvent être représentés sous la forme d'un problème de satisfaction de contraintes (CSP). En particulier, le formalisme CSP permet d'exprimer des problèmes de coloration de graphes, d'ordonnancement, de vision, de conception, de configuration, etc. Il vise à représenter sous forme de contraintes les propriétés et les relations qui existent entre les objets manipulés. Ces contraintes peuvent être décrites de multiples façons (par une équation, une inéquation, un prédicat, une fonction booléenne, une énumération des combinaisons de valeurs autorisées, etc.). Elles traduisent l'autorisation ou l'interdiction d'une combinaison de valeurs. Dans le formalisme CSP, la recherche d'une solution requiert de satisfaire toutes les contraintes. Dans la mesure où ces contraintes doivent être obligatoirement satisfaites, on les qualifie généralement de contraintes "dures". Cependant, pour certains problèmes réels, certaines contraintes (dites "molles") ne traduisent, dans la réalité, qu'une préférence, une possibilité, ... Leur satisfaction n'est donc pas forcément nécessaire. Représenter ces contraintes par des contraintes dures rend souvent les CSP correspondants inconsistants. Aussi, afin de pouvoir exprimer de telles contraintes, plusieurs extensions [Bistarelli *et al.*, 1995; Schiex *et al.*, 1995] du formalisme CSP ont été proposées parmi lesquelles les CSP valués (VCSP [Schiex *et al.*, 1995]). Le formalisme VCSP augmente le pouvoir d'expression du formalisme CSP en introduisant une graduation dans la violation des contraintes. Une valeur (appelée *valuation*) est associée à chaque contrainte. La valuation d'une contrainte traduit l'importance de la violation de cette contrainte. Autrement dit, le cadre VCSP autorise la violation de certaines contraintes, les contraintes étant soit dures, soit molles. L'objectif est alors de trouver une affectation de toutes les variables qui optimise un critère donné portant sur la satisfaction des contraintes. En d'autres termes, une solution du problème est une affectation qui peut éventuellement violer certaines contraintes et dont l'importance des violations est minimale suivant un critère et un ordre donnés. Le formalisme VCSP permet donc l'expression de problèmes d'optimisation.

La méthode de base pour la résolution de VCSP est l'algorithme Branch and Bound (séparation et évaluation). Bien sûr, de nombreuses améliorations ont été proposées, notamment à partir du cadre CSP. Néanmoins, à l'heure actuelle, les approches basées sur la programmation dynamique (comme la méthode des poupées russes [Verfaillie *et al.*, 1996] ou la méthode structurale de Koster [Koster, 1999]) semblent fournir les résultats les plus prometteurs. Il s'agit de méthodes qui divisent le problème en plusieurs sous-problèmes et qui exploitent les informations qu'elles produisent durant la résolution de chacun de ces sous-problèmes.

Dans cet article, nous proposons une nouvelle méthode énumérative pour la résolution de VCSP. Cette méthode, appelée  $BTD_{val}$ , est une généralisation naturelle de la méthode  $BTD$  [Jégou et Terrioux, 2002; 2003] définie dans le cadre de la résolution de CSP classiques. Cette généralisation requiert entre autres l'extension du cadre formel employé pour  $BTD$  et les CSP classiques. Toutefois, comme  $BTD$ , la méthode  $BTD_{val}$  est basée à la fois sur des techniques de backtracking (branch and bound) et sur la notion de décomposition arborescente d'un réseau de contraintes valué. Cette hybridation vise à bénéficier des avantages des deux approches : l'efficacité pratique de l'énumération et la garantie de borne de complexité en temps offerte par les méthodes structurales. La notion de décomposition arborescente permet à  $BTD_{val}$  de diviser le problème initial en

plusieurs sous-problèmes.  $\text{BTD}_{val}$  mémorise alors le résultat de la résolution de chaque sous-problème sous la forme de *goods structurels valués*. Ces goods structurels valués sont ensuite exploités afin de ne pas résoudre plusieurs fois un même sous-problème. Autrement dit, leur emploi évite certaines redondances dans la recherche, ce qui permet à  $\text{BTD}_{val}$  de fournir une borne de complexité en temps meilleure que celle des méthodes énumératives classiques. Notons que cette borne ne dépend alors que de paramètres structurels propres à la décomposition arborescente employée.

Cet article est organisé selon le plan suivant. La section 2 introduit les principales définitions concernant le formalisme VCSP. Puis, dans la section 3, nous effectuons quelques rappels concernant la notion de décomposition arborescente. Ensuite, dans la section 4, nous décrivons la méthode que nous proposons avant de présenter quelques résultats théoriques. Enfin, nous consacrons la section 5 à une discussion portant sur les travaux les proches, puis nous concluons dans la section 6.

## 2 CSP valués

Un **problème de satisfaction de contraintes** (CSP) se définit par la donnée d'un quadruplet  $(X, D, C, R)$ .  $X$  est un ensemble  $\{x_1, \dots, x_n\}$  de  $n$  variables, chaque variable  $x_i$  prenant ses valeurs dans un domaine fini  $d_{x_i}$  issu de  $D$ . Ces variables sont soumises à des contraintes issues de  $C$ . Chaque contrainte  $c$  est définie comme un ensemble  $\{x_{c_1}, \dots, x_{c_k}\}$  de variables. Une relation  $r_c$  (issue de  $R$ ) est associée à chaque contrainte  $c$  telle que  $r_c$  représente l'ensemble des tuples autorisés sur  $d_{x_{c_1}} \times \dots \times d_{x_{c_k}}$ . Etant donné  $Y \subseteq X$  tel que  $Y = \{x_1, \dots, x_k\}$ , une **instanciation** des variables de  $Y$  est un tuple  $\mathcal{A} = (v_1, \dots, v_k)$  de  $d_{x_1} \times \dots \times d_{x_k}$ . Une contrainte  $c$  est dite **satisfaite** par  $\mathcal{A}$  si  $c \subseteq Y$ ,  $(v_1, \dots, v_k)[c] \in r_c$ , **violée** sinon ( $\mathcal{A}[c]$  désignant la restriction de l'affectation  $\mathcal{A}$  aux variables de  $c$ ). Par la suite, on notera une affectation  $(v_1, \dots, v_k)$  sous la forme plus explicite  $(x_1 \leftarrow v_1, \dots, x_k \leftarrow v_k)$ .

À la différence du cadre CSP, les contraintes dans le cadre VCSP peuvent être soit dures soit molles. Résoudre un problème VCSP revient alors à rechercher une affectation qui optimise une fonction portant sur la satisfaction des contraintes. Autrement dit, une solution du problème est une affectation qui peut éventuellement violer certaines contraintes et dont l'importance des violations est minimale suivant un critère et un ordre donnés. Pour quantifier l'importance d'une violation, une valeur (appelée **valuation**) est associée à chaque contrainte. Lorsque plusieurs contraintes sont violées simultanément, les valuations correspondantes doivent être agrégées pour déterminer l'importance de la violation de l'ensemble de ces contraintes. Dans ce but, on se dote d'une structure de valuation :

**Définition 1** ([Schiex et al., 1995]) *Une **structure de valuation** est un tuple  $(E, \preceq, \oplus, \perp, \top)$  avec un ensemble  $E$  de valuations totalement ordonné par  $\preceq$ , muni d'un élément minimum (noté  $\perp$ ), d'un élément maximum (noté  $\top$ ) et d'une loi de composition interne (notée  $\oplus$ ) commutative, associative, monotone et telle que  $\perp$  soit un élément neutre pour  $\oplus$  et  $\top$  un élément absorbant.*

Les valuations permettent d'exprimer différents niveaux de violation. Par exemple,  $\perp$  caractérise une absence de violation (i.e. la satisfaction d'une contrainte) et  $\top$  une violation inacceptable. Quant à la loi  $\oplus$ , elle permet de calculer la valuation correspondant

à la violation simultanée de plusieurs contraintes. Notons que, dans certains cas, elle peut posséder d'autres propriétés comme l'idempotence ou la stricte monotonie. A partir de cette structure de valuation, on peut définir formellement la notion de CSP valué :

**Définition 2** ([Schiex *et al.*, 1995]) *Un CSP valué (VCSP) est un CSP classique  $\mathcal{P} = (X, D, C, R)$  doté d'une structure de valuation  $S = (E, \preceq, \oplus, \perp, \top)$  et d'une application  $\phi$  de  $C$  dans  $E$ , qui associe une valuation à chaque contrainte du CSP. On le note comme un sextuplet  $(X, D, C, R, S, \phi)$ .*

*Il est dit **binnaire** si chaque contrainte de  $C$  implique au plus deux variables.*

La valuation d'une instantiation complète  $\mathcal{A}$  des variables de  $X$  correspond à la combinaison (ou agrégation) des valuations des contraintes violées par  $\mathcal{A}$  :

**Définition 3** ([Schiex *et al.*, 1995]) *Soient un VCSP  $\mathcal{P} = (X, D, C, R, S, \phi)$  et une instantiation  $\mathcal{A}$  sur  $X$ . La **valuation** de  $\mathcal{A}$  dans  $\mathcal{P}$  se définit par :*

$$\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{c \in C \mid \mathcal{A} \text{ viole } c} \phi(c)$$

Étant donnée une instance  $\mathcal{P}$ , le problème VCSP consiste donc à trouver une affectation de toutes les variables de  $\mathcal{P}$  qui soit de valuation minimum au sens de  $\preceq$ . Cette valuation optimale est appelée **valuation du VCSP**. Par la suite, nous la noterons  $\alpha_{\mathcal{P}}^*$ . Déterminer la valuation d'un VCSP est un problème NP-difficile. Par exemple, considérons le VCSP dont le graphe de contrainte est présenté à la figure 1. Supposons que chaque domaine  $d_x$  soit égal à  $\{1, 2, 3\}$  et que chaque contrainte  $c_{xy} = \{x, y\}$  ait pour relation " $x < y$ " (par exemple la contrainte  $c_{AB}$  impose  $A < B$ ). Nous employons  $S = (\overline{\mathbb{N}}, <, +, 0, +\infty)$  comme structure de valuation. Pour chaque contrainte  $c$ , la valuation associée est 1. Pour ce VCSP, nous obtenons  $\alpha_{\mathcal{P}}^* = 2$ . En effet,  $(A \leftarrow 1, B \leftarrow 1, C \leftarrow 2, D \leftarrow 2, E \leftarrow 3, F \leftarrow 2, G \leftarrow 2, H \leftarrow 3, I \leftarrow 3, J \leftarrow 3)$  est une solution optimale. Elle viole seulement deux contraintes, à savoir  $c_{AB}$  et  $c_{CF}$ .

Cette notion de valuation d'une affectation complète peut être étendue à des instantiations partielles :

**Définition 4** ([Schiex *et al.*, 1995]) *Soient un VCSP  $\mathcal{P} = (X, D, C, R, S, \phi)$  et une instantiation  $\mathcal{A}$  sur  $Y \subset X$ . La **valuation locale** de  $\mathcal{A}$  dans  $\mathcal{P}$  se définit par :*

$$v_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\substack{c \in C \mid c \subseteq Y \\ \text{et } \mathcal{A} \text{ viole } c}} \phi(c)$$

La propriété suivante établit le lien existant entre la valuation d'une affectation complète et la valuation locale :

**Propriété 1** ([Schiex *et al.*, 1995]) *Soit un VCSP  $\mathcal{P} = (X, D, C, R, S, \phi)$ . Soient  $\mathcal{A}$  une instantiation complète et  $\mathcal{B} \subseteq \mathcal{A}$ . On a :  $v_{\mathcal{P}}(\mathcal{B}) \preceq v_{\mathcal{P}}(\mathcal{A}) = \mathcal{V}_{\mathcal{P}}(\mathcal{A})$ .*

La notion de valuation locale permet donc d'obtenir un minorant de la valuation globale. L'intérêt majeur de la valuation locale réside dans la possibilité de la calculer de façon incrémentale.

La méthode de base pour résoudre les CSP valués est l'algorithme branch and bound (séparation et évaluation, noté BB). Cette méthode énumérative utilise la valuation locale

de l'affectation courante comme minorant et la valuation de la meilleure solution connue comme majorant. Si le minorant ne dépasse pas le majorant, alors on étend l'instanciation courante en affectant une nouvelle variable. Sinon, on revient en arrière sur la dernière variable instanciée et on l'affecte avec une nouvelle valeur. Si toutes ses valeurs ont été essayées, on revient en arrière une nouvelle fois, et ainsi de suite. Plusieurs améliorations de cet algorithme ont été proposées. La plupart d'entre elles proviennent du cadre CSP et ont conduit à des méthodes comme les algorithmes Forward-Checking valué (noté  $FC_{val}$  [Schiex *et al.*, 1995]), Nogood Recording [Dago et Verfaillie, 1996], ... De nombreux travaux ont également porté sur l'emploi de la consistance d'arc [Larrosa *et al.*, 1999; Schiex, 2002; Larrosa, 2002; Larrosa et Schiex, 2003; Cooper et Schiex, 2004]. D'autre part, des méthodes basées sur la programmation dynamique ont été proposées (par exemple [Verfaillie *et al.*, 1996; Dechter, 1997; Koster, 1999; Meseguer et Sánchez, 2000; 2001; Dechter *et al.*, 2001; Larrosa *et al.*, 2002; Larrosa et Dechter, 2003]). Ces méthodes divisent le problème en plusieurs sous-problèmes et exploitent les informations qu'elles produisent durant la résolution de chaque sous-problème. A l'heure actuelle, des méthodes comme la méthode des poupées russes (notée RDS pour Russian Doll Search [Verfaillie *et al.*, 1996]) ou la méthode structurale proposée par Koster [Koster, 1999] semblent fournir les résultats les plus prometteurs. En effet, ces deux méthodes parviennent à résoudre les instances réelles du CELAR (il s'agit de problèmes d'allocation de fréquence) alors que les méthodes énumératives semblent éprouver quelques difficultés à résoudre ces mêmes instances.

### 3 Décomposition arborescente de graphes

Parmi tous les algorithmes de résolution de VCSP, seules les méthodes de décomposition basées sur le graphe de contraintes fournissent des garanties en terme de complexité théorique avant la résolution d'un problème. Elles procèdent en isolant des parties *a priori* intraitables en temps polynomial, pour parvenir à une seconde étape qui garantira un temps de résolution polynomial. En général, ces méthodes exploitent les propriétés topologiques du graphe de contraintes et sont basées sur la notion de **décomposition arborescente** (tree-decomposition) de graphes [Robertson et Seymour, 1986], dont la définition est rappelée ci-dessous :

**Définition 5 ([Robertson et Seymour, 1986])** Soit  $G = (X, E)$  un graphe. Une **décomposition arborescente** de  $G$  est une paire  $(\mathcal{C}, \mathcal{T})$  avec  $\mathcal{T} = (I, F)$  un arbre et  $\mathcal{C} = \{C_i : i \in I\}$  une famille de sous-ensembles de  $X$ , telle que chaque élément  $C_i$  correspond à un nœud de  $\mathcal{T}$  et vérifie :

$$(1) \bigcup_{i \in I} C_i = X,$$

(2) pour toute arête  $\{x, y\} \in E$ , il existe  $i \in I$  avec  $\{x, y\} \subseteq C_i$ , et

(3) pour tout  $i, j, k \in I$ , si  $k$  est sur un chemin de  $i$  à  $j$  dans  $\mathcal{T}$ , alors  $C_i \cap C_j \subseteq C_k$ .

La largeur d'une décomposition arborescente  $(\mathcal{C}, \mathcal{T})$  est égale à  $\max_{i \in I} |C_i| - 1$ . La **tree-width** d'un graphe  $G$  est la largeur minimale sur toutes les décompositions arborescentes de  $G$ .

Les éléments  $C_i$  de  $\mathcal{C}$  sont généralement appelés **regroupements** ou **clusters**. Par abus de langage, nous assimilerons les éléments de  $I$  aux clusters auxquels ils sont associés.

Notons, pour le lecteur qui n'est pas familier avec ces notions, que la définition d'un arbre  $\mathcal{T} = (I, F)$  fait intervenir un ensemble  $F$  d'arêtes. Cet ensemble est nécessaire pour satisfaire la partie (3) de la définition 5.

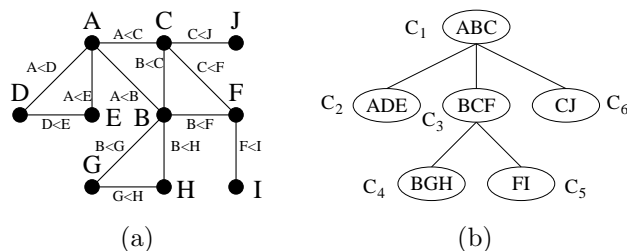


FIG. 1 – (a) Un graphe de contraintes avec 10 variables. (b) Une décomposition arborescente de ce graphe de contraintes.

Le problème de recherche d'une décomposition arborescente est NP-dur [Arnborg *et al.*, 1987]. Toutefois, de nombreux travaux ont été développés dans cette direction [Becker et Geiger, 2001]. Ceux-ci sont fréquemment basés sur l'exploitation de la notion de graphe **triangulé**. Un graphe  $G = (X, E)$  est dit triangulé s'il ne possède pas de cycle de longueur 4 ou plus sans corde (une corde étant une arête joignant deux sommets non consécutifs dans le cycle). Les liens entre graphes triangulés et décompositions arborescentes sont évidents. En effet, étant donné un graphe triangulé, l'ensemble de ses cliques maximales  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$  de  $(X, E)$  correspond à la famille de sous-ensembles associée à cette décomposition. Comme un graphe quelconque  $G = (X, E)$  n'est pas nécessairement triangulé, une décomposition arborescente peut être approximée en triangulant  $G$ . Nous appelons **triangulation** l'ajout à  $G$  d'un ensemble  $E'$  d'arêtes de telle sorte que le graphe  $G' = (X, E \cup E')$  soit triangulé. La largeur d'une triangulation  $G'$  du graphe  $G$  est égale à la taille maximum des cliques moins un dans le graphe résultant  $G'$ . La tree-width de  $G$  est alors égale à la largeur minimale pour toutes les triangulations. Le graphe de la figure 1(a) est déjà triangulé. La taille de la plus grande clique est trois et la tree-width de ce graphe est deux. Dans la figure 1(b), un arbre dont les nœuds correspondent aux cliques maximales du graphe triangulé est une décomposition arborescente du graphe de la figure 1(a). Ainsi, nous avons  $C_1 = \{A, B, C\}$ ,  $C_2 = \{A, D, E\}$ ,  $C_3 = \{B, C, F\}$ ,  $C_4 = \{B, G, H\}$ ,  $C_5 = \{F, I\}$  et  $C_6 = \{C, J\}$ .

La notion de décomposition arborescente est exploitée dans le cadre des CSP classiques par plusieurs méthodes structurales (voir [Gottlob *et al.*, 2000] pour une description et une comparaison théorique de ces méthodes). Ces méthodes présentent l'avantage de fournir les meilleures bornes de complexité théorique en temps. Par exemple, la méthode de décomposition de CSP appelé *Tree-Clustering* [Dechter et Pearl, 1989; Dechter et Fattah, 2001] est en général décrite en employant une approximation d'une triangulation optimale. Elle possède une complexité en temps en  $O(m.d^{w^++1})$  avec  $w^++1$  la taille du plus grand cluster pour une complexité en espace en  $O(n.s.d^s)$  avec  $s$  la taille du plus grand séparateur minimal (c'est-à-dire la taille  $s \leq w^+$  de la plus grande intersection entre deux clusters). Enfin, notons que chaque décomposition induit une valeur  $w^+$  telle que  $w \leq w^+$  avec  $w$  la tree-width du graphe de contraintes initial.

La notion de décomposition arborescente est également à la base de la méthode BTD [Jégou et Terrioux, 2002; 2003]. Cette méthode qui utilise aussi des techniques de recherche arborescente bénéficie d’une part de l’efficacité pratique de l’énumération, et d’autre part des garanties en termes de complexité théorique offertes par les méthodes de décomposition de CSP, la complexité en temps et en espace de BTD étant similaires à celles du Tree-Clustering.

Dans le cadre des CSP valués, l’approche par programmation dynamique [Koster, 1999] exploite également une décomposition arborescente. Elle a une complexité en temps en  $O(nd^{3(w^++1)})$  et une complexité en espace en  $O(d^{w^++1})$ . Dans le cadre classique comme dans le cadre valué, la quantité de mémoire requise pour mettre en œuvre ces méthodes constitue, d’un point de vue pratique, leur principal handicap. Cet handicap peut se révéler suffisamment important pour rendre de telles approches inutilisables en pratique.

Dans la prochaine section, nous présentons une méthode énumérative pour la résolution de VCSP qui, en exploitant une décomposition arborescente, fournit des bornes de complexité du même ordre de grandeur (ou meilleures) que celles données ci-dessus.

## 4 L’algorithme $\text{BTD}_{val}$

### 4.1 Présentation

Comme BTD, la méthode  $\text{BTD}_{val}$  (pour Backtracking sur Tree-Decomposition) procède par une recherche énumérative qui est guidée par un ordre partiel statique préétabli à partir d’une décomposition arborescente du graphe de contraintes. Aussi, la première étape de  $\text{BTD}_{val}$  consiste à calculer une décomposition arborescente ou une approximation de décomposition arborescente. L’ordre partiel considéré permet d’exploiter quelques propriétés structurelles du graphe pendant la recherche afin d’élaguer certaines branches de l’arbre de recherche. En fait, ce qui distingue  $\text{BTD}_{val}$  des autres techniques de backtracking concerne les points suivants :

- l’ordre d’instanciation des variables est induit par une décomposition arborescente du graphe de contraintes,
- certaines parties de l’espace de recherche ne seront plus visitées dès qu’on aura calculé leur valuation optimale (notion de *good structural valué*).

Notons que si, dans notre description,  $\text{BTD}_{val}$  repose sur l’algorithme Branch and Bound, nous pouvons également utiliser des variantes plus sophistiquées comme, par exemple, l’algorithme  $\text{FC}_{val}$ .

### 4.2 Justifications formelles

Soit  $\mathcal{P} = (X, D, C, R, S, \phi)$  une instance VCSP. Soit  $(\mathcal{C}, \mathcal{T})$  une décomposition arborescente (ou une approximation d’une décomposition arborescente) du graphe  $(X, C)$ . Nous supposons que les éléments de  $\mathcal{C} = \{C_i : i \in I\}$  sont indicés suivant la notion de numérotation compatible :

**Définition 6** Une numérotation sur  $\mathcal{C}$  compatible avec une numérotation préfixée de  $\mathcal{T} = (I, F)$  dont  $C_1$  est la racine est appelée **numérotation compatible**  $N_{\mathcal{C}}$ .

La figure 1(b) présente une numérotation compatible sur  $\mathcal{C}$ . Nous notons  $Desc(\mathcal{C}_j)$  l'ensemble des variables appartenant à l'union des descendants  $\mathcal{C}_k$  de  $\mathcal{C}_j$  dans l'arbre enraciné en  $\mathcal{C}_j$ ,  $\mathcal{C}_j$  inclus. Par exemple,  $Desc(\mathcal{C}_3) = \mathcal{C}_3 \cup \mathcal{C}_4 \cup \mathcal{C}_5 = \{B, C, F, G, H, I\}$ . La numérotation  $N_{\mathcal{C}}$  définit un ordre partiel sur les variables qui permet d'établir un ordre d'énumération sur les variables de  $\mathcal{P}$  :

**Définition 7** *Un ordre d'énumération compatible est un ordre  $\preceq_X$  sur les variables de  $X$  tel que  $\forall x, y \in X, x \preceq_X y$  si  $\exists \mathcal{C}_i \ni x, \forall \mathcal{C}_j \ni y, i \leq j$ .*

Pour notre exemple, l'ordre alphabétique  $A, B, \dots, I, J$  est un ordre d'énumération compatible. La décomposition arborescente avec la numérotation  $N_{\mathcal{C}}$  permet de partitionner l'ensemble des contraintes.

**Définition 8** *Soit  $\mathcal{C}_i$  un cluster. L'ensemble  $E_{\mathcal{P}, \mathcal{C}_i}$  des contraintes propres au cluster  $\mathcal{C}_i$  est défini par  $E_{\mathcal{P}, \mathcal{C}_i} = \{c \in C \mid c \subseteq \mathcal{C}_i \text{ et } c \not\subseteq \mathcal{C}_{p(i)}\}$  avec  $\mathcal{C}_{p(i)}$  le cluster père de  $\mathcal{C}_i$ .*

L'ensemble  $E_{\mathcal{P}, \mathcal{C}_i}$  contient donc les contraintes de la forme  $c = \{x, y\}$  avec  $x$  et  $y$  deux variables de  $\mathcal{C}_i$  telle que  $x$  et  $y$  n'appartiennent pas simultanément à  $\mathcal{C}_{p(i)}$  le cluster père de  $\mathcal{C}_i$ . Par exemple, pour le problème décrit à la figure 1, nous avons  $E_{\mathcal{P}, \mathcal{C}_1} = \{c_{AB}, c_{AC}, c_{BC}\}$ ,  $E_{\mathcal{P}, \mathcal{C}_2} = \{c_{AD}, c_{AE}, c_{DE}\}$ ,  $E_{\mathcal{P}, \mathcal{C}_3} = \{c_{BF}, c_{CF}\}$ ,  $E_{\mathcal{P}, \mathcal{C}_4} = \{c_{BG}, c_{BH}, c_{GH}\}$ ,  $E_{\mathcal{P}, \mathcal{C}_5} = \{c_{FI}\}$  et  $E_{\mathcal{P}, \mathcal{C}_6} = \{c_{CJ}\}$ .

**Propriété 2** *Les  $(E_{\mathcal{P}, \mathcal{C}_i})_i$  forment une partition de  $C$ .*

*Preuve :*

(i) Montrons que  $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P}, \mathcal{C}_i} = C$ .

Comme il est évident que  $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P}, \mathcal{C}_i} \subset C$ , il reste à démontrer que  $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P}, \mathcal{C}_i} \supset C$ .

Soit  $c \in C$ . D'après la définition 5, il existe au moins un cluster  $\mathcal{C}_i$  tel que  $c \subseteq \mathcal{C}_i$ . En particulier, on a nécessairement  $c \subseteq \mathcal{C}_k$  où  $k = \min\{i \mid c \subseteq \mathcal{C}_i\}$  et  $c \not\subseteq \mathcal{C}_{p(k)}$ . Donc,  $c \in E_{\mathcal{P}, \mathcal{C}_k}$ .

D'où  $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P}, \mathcal{C}_i} \supset C$ . Donc  $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P}, \mathcal{C}_i} = C$ .

(ii) Montrons que  $\forall \mathcal{C}_i, \mathcal{C}_j, E_{\mathcal{P}, \mathcal{C}_i} \cap E_{\mathcal{P}, \mathcal{C}_j} = \emptyset$ .

Supposons qu'il existe  $\mathcal{C}_i$  et  $\mathcal{C}_j$  tels que  $E_{\mathcal{P}, \mathcal{C}_i} \cap E_{\mathcal{P}, \mathcal{C}_j} \neq \emptyset$ . Soit  $c \in E_{\mathcal{P}, \mathcal{C}_i} \cap E_{\mathcal{P}, \mathcal{C}_j}$ . Par définition,  $c \subseteq \mathcal{C}_i \cap \mathcal{C}_j$ .

D'après la définition 5, il existe un chemin allant de  $\mathcal{C}_i$  à  $\mathcal{C}_j$  tel que si  $\mathcal{C}_k$  est sur ce chemin,  $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$ . Sur un tel chemin, on rencontre nécessairement le père de  $\mathcal{C}_i$  ou le père de  $\mathcal{C}_j$ . Donc  $c \subseteq \mathcal{C}_{p(i)}$  ou  $c \subseteq \mathcal{C}_{p(j)}$ . D'où une contradiction puisque  $c \in E_{\mathcal{P}, \mathcal{C}_i}$  et  $c \in E_{\mathcal{P}, \mathcal{C}_j}$ .

Donc  $\forall i, j, E_{\mathcal{P}, \mathcal{C}_i} \cap E_{\mathcal{P}, \mathcal{C}_j} = \emptyset$

Les  $(E_{\mathcal{P}, \mathcal{C}_i})_i$  forment bien une partition de  $C$ .  $\square$

Remarquons que cette propriété s'avère fondamentale quand la loi  $\oplus$  n'est pas idempotente. En effet, dans un tel cas, lorsque nous calculons la valuation d'une affectation donnée, nous ne devons pas prendre en compte plusieurs fois la même contrainte sous peine de surestimer cette valuation. Comme les ensembles  $(E_{\mathcal{P}, \mathcal{C}_i})_i$  forment une partition



de  $C$ , leur exploitation nous permettra par la suite de ne pas rencontrer un tel problème. Nous pourrions alors garantir que la méthode  $\text{BTD}_{val}$  calculera correctement la valuation de chaque instanciation. A présent, nous pouvons définir la notion de VCSP induit :

**Définition 9** Soient  $\mathcal{C}_i$  et  $\mathcal{C}_j$  deux clusters avec  $\mathcal{C}_j$  fils de  $\mathcal{C}_i$ . Soit  $\mathcal{A}$  une affectation sur  $\mathcal{C}_i \cap \mathcal{C}_j$ .  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j} = (X_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}}, D_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}}, C_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}}, R_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}}, S, \phi)$  est le **VCSP induit** par  $\mathcal{A}$  sur la descendance de  $\mathcal{C}_i$  enracinée en  $\mathcal{C}_j$  avec :

- $X_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}} = \text{Desc}(\mathcal{C}_j)$ ,
- $D_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}} = \{d_x, \mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j} = d_x | x \in \text{Desc}(\mathcal{C}_j) \setminus (\mathcal{C}_i \cap \mathcal{C}_j)\} \cup \{d_x, \mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j} = \{\mathcal{A}[x]\} | x \in \mathcal{C}_i \cap \mathcal{C}_j\}$ ,
- $C_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}} = E_{\mathcal{P}, \mathcal{C}_j} \cup \bigcup_{\mathcal{C}_d \text{ descendant de } \mathcal{C}_j} E_{\mathcal{P}, \mathcal{C}_d}$ ,
- $R_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}} = \{r_c \cap \prod_{x \in c} d_x, \mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j} | c \in C_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}} \text{ et } r_c \in R\}$ .

Le VCSP induit  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}$  correspond au VCSP  $\mathcal{P}$  restreint au sous-problème enraciné en  $\mathcal{C}_j$ . Autrement dit, nous considérons le problème dont les variables sont celles de la descendance de  $\mathcal{C}_i$  enracinée en  $\mathcal{C}_j$ . Quant aux domaines, le domaine d'une variable  $x$  appartenant à  $\mathcal{C}_i \cap \mathcal{C}_j$  est restreint à la valeur avec laquelle  $x$  est affectée dans l'instanciation  $\mathcal{A}$  alors que le domaine d'une variable n'appartenant pas à  $\mathcal{C}_i \cap \mathcal{C}_j$  reste inchangé. Enfin, concernant l'ensemble des contraintes de  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_i / \mathcal{C}_j}$ , il contient uniquement les contraintes de  $C$  qui portent sur au moins une variable qui n'apparaît que dans la descendance de  $\mathcal{C}_i$  enracinée en  $\mathcal{C}_j$ . Par exemple, étant donnée l'affectation  $\mathcal{A} = (B \leftarrow 2, C \leftarrow 2)$  sur  $\mathcal{C}_1 \cap \mathcal{C}_3$ , nous considérons le VCSP  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_1 / \mathcal{C}_3}$  induit par  $\mathcal{A}$  sur la descendance de  $\mathcal{C}_1$  enraciné en  $\mathcal{C}_3$ . Nous avons alors  $X_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_1 / \mathcal{C}_3}} = \{B, C, F, G, H, I\}$ ,  $d_B = d_C = \{2\}$ ,  $d_F = d_G = d_H = d_I = \{1, 2, 3\}$  et  $C_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_1 / \mathcal{C}_3}} = \{c_{BF}, c_{CF}, c_{BG}, c_{BH}, c_{GH}, c_{FI}\}$ . Notons que la contrainte  $c_{BC}$  n'appartient pas à l'ensemble des contraintes de  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_1 / \mathcal{C}_3}$  car il ne s'agit pas d'une contrainte propre de  $\mathcal{C}_3$ .

A partir des ensembles  $E_{\mathcal{P}, \mathcal{C}_i}$ , nous pouvons introduire la notion de valuation locale à un cluster :

**Définition 10** Soient un cluster  $\mathcal{C}_i$  et une instanciation  $\mathcal{A}$  sur  $Y \subset X$ . La **valuation locale**  $v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A})$  de  $\mathcal{A}$  au cluster  $\mathcal{C}_i$  dans  $\mathcal{P}$  est la valuation locale de  $\mathcal{A}$  restreinte aux contraintes de  $E_{\mathcal{P}, \mathcal{C}_i}$ .

$$v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}) = \bigoplus_{\substack{c \in E_{\mathcal{P}, \mathcal{C}_i} | c \subseteq Y \\ \text{et } \mathcal{A} \text{ viole } c}} \phi(c)$$

En d'autres mots, la valuation locale à un cluster  $\mathcal{C}_i$  ne tient compte que des contraintes propres au cluster  $\mathcal{C}_i$ , c'est-à-dire aux contraintes de  $E_{\mathcal{P}, \mathcal{C}_i}$ . Comme pour la valuation locale, il est possible de calculer la valuation locale à un cluster de façon incrémentale. Cette valuation possède plusieurs propriétés intéressantes. D'abord, son calcul ne dépend que des variables du cluster considéré.

**Propriété 3** Soient  $\mathcal{C}_i$  un cluster et  $\mathcal{A}$  une affectation sur  $Y \subseteq X$  tel que  $\mathcal{C}_i \subseteq Y$ .

$$v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}) = v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}[\mathcal{C}_i])$$

$$\begin{aligned}
\text{Preuve : } v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}) &= \bigoplus_{\substack{c \in E_{\mathcal{P}, \mathcal{C}_i} | c \subseteq Y \\ \text{et } \mathcal{A} \text{ viole } c}} \phi(c) = \bigoplus_{\substack{c \in E_{\mathcal{P}, \mathcal{C}_i} | c \subseteq Y \cap \mathcal{C}_i \\ \text{et } \mathcal{A} \text{ viole } c}} \phi(c) = \bigoplus_{\substack{c \in E_{\mathcal{P}, \mathcal{C}_i} | c \subseteq Y \cap \mathcal{C}_i \\ \text{et } \mathcal{A}[\mathcal{C}_i] \text{ viole } c}} \phi(c) = \\
v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}[\mathcal{C}_i]) &\quad \square
\end{aligned}$$

Ensuite, l'agrégation des valuations locales à un cluster permet de calculer la valuation d'une affectation complète.

**Propriété 4** *Soit une instanciation  $\mathcal{A}$  sur  $X$ .*

$$\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\mathcal{C}_i \subseteq X} v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A})$$

*Preuve :* Puisque les  $(E_{\mathcal{P}, \mathcal{C}_i})_i$  forment une partition de  $C$  (d'après la propriété 2), chaque contrainte de  $C$  est comptabilisée une et une seule fois.

Par conséquent,  $\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\mathcal{C}_i \subseteq X} v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A})$ .  $\square$

Il découle de ces deux propriétés que le calcul de la valuation d'une affectation complète  $\mathcal{A}$  peut s'effectuer en exploitant uniquement la valuation locale à chaque cluster  $\mathcal{C}_i$  sur l'affectation restreinte  $\mathcal{A}[\mathcal{C}_i]$ .

Enfin, la propriété suivante assure que la valuation locale à un cluster  $\mathcal{C}_j$  d'une affectation  $\mathcal{B}$  reste la même que l'on considère le problème  $\mathcal{P}$  ou un sous-problème induit de  $\mathcal{P}$  contenant  $\mathcal{C}_j$ .

**Propriété 5** *Soient  $\mathcal{C}_i$  et  $\mathcal{C}_j$  deux clusters avec  $\mathcal{C}_j$  un descendant de  $\mathcal{C}_i$ . Soient  $\mathcal{A}$  une affectation sur  $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$  et  $\mathcal{P}' = \mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}$ . Si  $\mathcal{B}$  est une affectation sur  $\mathcal{C}_j$  telle que  $\mathcal{B}[\mathcal{C}_j \cap \mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i \cap \mathcal{C}_{p(i)}]$ , alors  $v_{\mathcal{P}, \mathcal{C}_j}(\mathcal{B}) = v_{\mathcal{P}', \mathcal{C}_j}(\mathcal{B})$ .*

*Preuve :* comme  $E_{\mathcal{P}, \mathcal{C}_j} = E_{\mathcal{P}', \mathcal{C}_j}$   $v_{\mathcal{P}, \mathcal{C}_j}(\mathcal{B}) = v_{\mathcal{P}', \mathcal{C}_j}(\mathcal{B})$ .  $\square$

Nous pouvons désormais définir la notion de good structurel valué.

**Définition 11** *Soient  $\mathcal{C}_i$  et  $\mathcal{C}_j$  deux clusters avec  $\mathcal{C}_j$  fils de  $\mathcal{C}_i$ . Un **good structurel valué** de  $\mathcal{C}_i$  par rapport à  $\mathcal{C}_j$  est un couple  $(\mathcal{A}, v)$  avec  $\mathcal{A}$  une affectation sur  $\mathcal{C}_i \cap \mathcal{C}_j$  et  $v = \alpha_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_i/\mathcal{C}_j}}^*$  la valuation optimale du VCSP  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_i/\mathcal{C}_j}$ .*

Par exemple, si nous considérons l'affectation  $\mathcal{A} = (B \leftarrow 2, C \leftarrow 2)$  sur  $\mathcal{C}_1 \cap \mathcal{C}_3$ , nous obtenons le good  $(\mathcal{A}, 2)$  de  $\mathcal{C}_1$  par rapport à  $\mathcal{C}_3$ . Notons que cette notion de good structurel valué englobe simultanément les notions de good et de nogood structurels du cadre classique [Jégou et Terrioux, 2002; 2003]. Nous verrons par la suite que leur emploi est similaire à celui des goods dans BTD.

Étant donnée une affectation  $\mathcal{A}$  sur  $\mathcal{C}_i$ , le théorème suivant exprime le fait que la valuation de la meilleure affectation  $\mathcal{B}$  sur  $Desc(\mathcal{C}_i)$  telle que  $\mathcal{B}[\mathcal{C}_i] = \mathcal{A}$  peut être calculée en exploitant la valuation optimale de chaque sous-problème enraciné en un fils  $\mathcal{C}_f$  de  $\mathcal{C}_i$  et induit par  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_f]$ . Notons que la valuation optimale de chaque sous-problème peut être calculée indépendamment de celles des autres sous-problèmes et qu'elle peut être fournie soit par une résolution du sous-problème, soit par un good structurel valué.

**Théorème 1** *Soient  $\mathcal{C}_i$  un cluster,  $\mathcal{A}$  une instanciation sur  $\mathcal{C}_i$ , et  $\mathcal{P}' = \mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}], \mathcal{C}_{p(i)}/\mathcal{C}_i}$ .*

$$\min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ \text{et } \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} v_{\mathcal{P}'}(\mathcal{B}) = v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_f \text{ fils de } \mathcal{C}_i} \alpha_{\mathcal{P}'_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_f], \mathcal{C}_i/\mathcal{C}_f}}^*$$

Afin de démontrer ce théorème, nous avons besoin du lemme suivant :

**Lemme 1** Soient  $\mathcal{C}_i$  un cluster et  $\mathcal{A}$  une instanciation sur  $\mathcal{C}_i$ . Soit  $\mathcal{P}' = \mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}], \mathcal{C}_{p(i)}/\mathcal{C}_i}$ .

$$\text{Soit } \lambda = \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ \text{et } \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left( \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i)} \left[ \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}) \right] \right).$$

$$\text{Soit } \lambda' = \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i)} \left( \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_j) \cup \mathcal{C}_i \\ \text{et } \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left[ \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}) \right] \right).$$

Les quantités  $\lambda$  et  $\lambda'$  sont égales.

*Preuve (lemme 1) :*

Pour tout  $\mathcal{C}_j$  fils de  $\mathcal{C}_i$ , on pose  $\lambda_{\mathcal{C}_j} = \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_j) \cup \mathcal{C}_i \\ \text{et } \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left[ \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}) \right]$ . On a alors

$$\lambda' = \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i)} \lambda_{\mathcal{C}_j}.$$

Pour chaque  $\mathcal{C}_j$  fils de  $\mathcal{C}_i$ , il existe une affectation  $\mathcal{B}_{\mathcal{C}_j}$  sur  $Desc(\mathcal{C}_j) \cup \mathcal{C}_i$  qui vérifie  $\lambda_{\mathcal{C}_j} = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_{\mathcal{C}_j})$  et telle que  $\mathcal{B}_{\mathcal{C}_j}[\mathcal{C}_i] = \mathcal{A}$ .

De même, il existe une affectation  $\mathcal{B}_\lambda$  sur  $Desc(\mathcal{C}_i)$  telle que  $\mathcal{B}_\lambda[\mathcal{C}_i] = \mathcal{A}$  et qui vérifie

$$\lambda = \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i)} \left[ \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda) \right].$$

Montrons que pour chaque fils  $\mathcal{C}_j$  de  $\mathcal{C}_i$ ,  $\mathcal{B}_\lambda$  vérifie  $\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) = \lambda_{\mathcal{C}_j}$ .

Supposons qu'il existe un fils  $\mathcal{C}_s$  de  $\mathcal{C}_i$  tel que  $\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i]) \neq \lambda_{\mathcal{C}_s}$ .

Par définition de  $\lambda_{\mathcal{C}_s}$ ,  $\lambda_{\mathcal{C}_s} \prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i]) = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda)$

Soit  $\mathcal{B}'$  une affectation sur  $Desc(\mathcal{C}_i)$  telle que  $\mathcal{B}'[\mathcal{C}_i] = \mathcal{A}$  et  $\forall \mathcal{C}_j \in \text{Fils}(\mathcal{C}_i), \mathcal{B}'[Desc(\mathcal{C}_j) \cup \mathcal{C}_i] = \mathcal{B}_{\mathcal{C}_j}$ .

Une telle affectation existe car  $\forall \mathcal{C}_j, \mathcal{C}_{j'} \in \text{Fils}(\mathcal{C}_i), Desc(\mathcal{C}_j) \cap Desc(\mathcal{C}_{j'}) \subseteq \mathcal{C}_i$ .

De plus, elle vérifie entre autres  $\lambda_{\mathcal{C}_s} = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}'[Desc(\mathcal{C}_s) \cup \mathcal{C}_i])$ .

$$\text{Donc, } \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i)} \left[ \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}') \right] = \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i)} \lambda_{\mathcal{C}_j} = \lambda'$$

$$\lambda' = \lambda_{\mathcal{C}_s} \oplus \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \lambda_{\mathcal{C}_j}$$

$$\prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i]) \oplus \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \left[ \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) \right]$$

$$\prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda) \oplus \bigoplus_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \left[ \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda) \right] = \lambda$$

D'où une contradiction avec la définition de  $\lambda$ .

Donc pour chaque  $\mathcal{C}_j$  fils de  $\mathcal{C}_i$ ,  $\mathcal{B}_\lambda$  vérifie  $\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}', \mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) = \lambda_{\mathcal{C}_j}$ .

Il s'ensuit l'égalité entre les quantités  $\lambda$  et  $\lambda'$ .  $\square$

*Preuve (théorème 1) :*

On pose  $M = \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_i) \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} v_{\mathcal{P}'}(\mathcal{B})$ .

$$\begin{aligned}
M &\stackrel{\text{propriété 1}}{=} \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_i) \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} \mathcal{V}_{\mathcal{P}'}(\mathcal{B}). \\
&\stackrel{\text{propriété 4}}{=} \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_i) \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} \left( \bigoplus_{C_j \subseteq Desc(C_i)} v_{\mathcal{P}', C_j}(\mathcal{B}) \right) \\
&= \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_i) \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} \left( v_{\mathcal{P}', C_i}(\mathcal{B}) \oplus \bigoplus_{\substack{C_j | j \neq i, \\ C_j \subseteq Desc(C_i)}} v_{\mathcal{P}', C_j}(\mathcal{B}) \right) \\
&\stackrel{\text{propriété 3}}{=} \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_i) \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} \left( v_{\mathcal{P}', C_i}(\mathcal{B}[C_i]) \oplus \bigoplus_{\substack{C_j | j \neq i, \\ C_j \subseteq Desc(C_i)}} v_{\mathcal{P}', C_j}(\mathcal{B}) \right)
\end{aligned}$$

Or pour toute affectation  $\mathcal{B}$  telle que  $X_{\mathcal{B}} = Desc(C_i)$  et  $\mathcal{B}[C_i] = \mathcal{A}$ ,  $v_{\mathcal{P}', C_i}(\mathcal{B}[C_i]) = v_{\mathcal{P}', C_i}(\mathcal{A})$ .

Comme  $v_{\mathcal{P}', C_i}(\mathcal{A})$  est une constante, on a :

$$\begin{aligned}
M &= v_{\mathcal{P}', C_i}(\mathcal{A}) \oplus \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_i) \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} \left( \bigoplus_{\substack{C_j | j \neq i, \\ C_j \subseteq Desc(C_i)}} v_{\mathcal{P}', C_j}(\mathcal{B}) \right) \\
&= v_{\mathcal{P}', C_i}(\mathcal{A}) \oplus \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_i) \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} \left( \bigoplus_{C_j \in Fils(C_i)} \left[ \bigoplus_{C_k \subseteq Desc(C_j)} v_{\mathcal{P}', C_k}(\mathcal{B}) \right] \right) \\
&\stackrel{\text{lemme 1}}{=} v_{\mathcal{P}', C_i}(\mathcal{A}) \oplus \bigoplus_{C_j \in Fils(C_i)} \left( \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_j) \cup C_i \\ \text{et } \mathcal{B}[C_i]=\mathcal{A}}} \left[ \bigoplus_{C_k \subseteq Desc(C_j)} v_{\mathcal{P}', C_k}(\mathcal{B}) \right] \right) \\
&= v_{\mathcal{P}', C_i}(\mathcal{A}) \oplus \bigoplus_{C_j \in Fils(C_i)} \left( \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_j) \\ \text{et } \mathcal{B}[C_i \cap C_j]=\mathcal{A}[C_i \cap C_j]}} \left[ \bigoplus_{C_k \subseteq Desc(C_j)} v_{\mathcal{P}', C_k}(\mathcal{B}) \right] \right) \\
&\stackrel{\text{propriété 4}}{=} v_{\mathcal{P}', C_i}(\mathcal{A}) \oplus \bigoplus_{C_j \in Fils(C_i)} \left( \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_j) \\ \text{et } \mathcal{B}[C_i \cap C_j]=\mathcal{A}[C_i \cap C_j]}} \mathcal{V}_{\mathcal{P}', \mathcal{A}[C_i \cap C_j], C_i/C_j}(\mathcal{B}) \right) \\
&\stackrel{\text{propriété 5}}{=} v_{\mathcal{P}, C_i}(\mathcal{A}) \oplus \bigoplus_{C_j \in Fils(C_i)} \left( \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=Desc(C_j) \\ \text{et } \mathcal{B}[C_i \cap C_j]=\mathcal{A}[C_i \cap C_j]}} \mathcal{V}_{\mathcal{P}, \mathcal{A}[C_i \cap C_j], C_i/C_j}(\mathcal{B}) \right) \\
&= v_{\mathcal{P}, C_i}(\mathcal{A}) \oplus \bigoplus_{C_j \text{ fils de } C_i} \alpha_{\mathcal{P}, \mathcal{A}[C_i \cap C_j], C_i/C_j}^* \square
\end{aligned}$$

À partir du théorème 1, on déduit le corollaire suivant. Ce corollaire établit le lien existant entre la valuation optimale d'un problème enraciné en  $C_i$  et la valuation optimale de chaque sous-problème enraciné en un fils  $C_j$  de  $C_i$ .

**Corollaire 1** Soient  $C_i$  un cluster et  $\mathcal{A}$  une instanciation sur  $C_i \cap C_{p(i)}$ .

$$\alpha_{\mathcal{P}, \mathcal{A}, C_{p(i)}/C_i}^* = \min_{\substack{\mathcal{B}|X_{\mathcal{B}}=C_i \\ \text{et } \mathcal{B}[C_i \cap C_{p(i)}]=\mathcal{A}}} \left( v_{\mathcal{P}, C_i}(\mathcal{B}) \oplus \bigoplus_{C_j \text{ fils de } C_i} \alpha_{\mathcal{P}, \mathcal{B}[C_i \cap C_j], C_i/C_j}^* \right)$$

*Preuve :*

$$\begin{aligned}
\alpha_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}}^* &= \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ \text{et } \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}}} \mathcal{V}_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}}(\mathcal{B}) \\
&= \text{propriété 1} \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ \text{et } \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}}} v_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}}(\mathcal{B}) \\
&= \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = \mathcal{C}_i \\ \text{et } \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}}} \left( \min_{\substack{\mathcal{B}' | X_{\mathcal{B}'} = Desc(\mathcal{C}_i) \\ \text{et } \mathcal{B}'[\mathcal{C}_i] = \mathcal{B}[\mathcal{C}_i]}} v_{\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}}(\mathcal{B}') \right) \\
&= \text{théorème 1} \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = \mathcal{C}_i \\ \text{et } \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}}} \left( v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{B}) \oplus \bigoplus_{\mathcal{C}_j \text{ fils de } \mathcal{C}_i} \alpha_{\mathcal{P}_{\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j], \mathcal{C}_i/\mathcal{C}_j}}^* \right) \square
\end{aligned}$$

### 4.3 L'algorithme $BTD_{val}$

La version de l'algorithme  $BTD_{val}$  présentée ici repose sur l'algorithme BB, mais il est également possible d'employer des algorithmes plus sophistiqués comme l'algorithme  $FC_{val}$ . L'algorithme  $BTD_{val}$  (décrit à la figure 2) explore l'espace de recherche en exploitant un ordre compatible sur les variables qui est induit par la décomposition arborescente. Il commence donc son énumération par les variables du cluster racine  $\mathcal{C}_1$ . A l'intérieur d'un cluster  $\mathcal{C}_i$  (lignes 20-36), il procède de façon classique à la manière de BB en affectant une valeur à une variable, en maintenant et en comparant une borne inférieure à une borne supérieure et en revenant en arrière si la valeur du minorant dépasse ou égale celle du majorant. Cependant, à la différence de BB, les bornes employées par  $BTD_{val}$  sont locales à un sous-problème. Plus précisément, elles ne tiennent compte que du sous-problème enracinée en  $\mathcal{C}_i$  (à savoir le VCSP induit  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}$  avec  $\mathcal{A}$  l'affectation courante sur  $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$ ). Le minorant correspond alors à la valuation de l'affectation courante sur  $Desc(\mathcal{C}_i)$ , c'est-à-dire la valuation locale de l'instanciation courante dans  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}$ . Le majorant est, quant à lui, défini par la valuation de la meilleure instanciation  $\mathcal{B}$  connue sur  $Desc(\mathcal{C}_i)$  telle que  $\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}$ . Autrement dit, il s'agit de la valuation de la meilleure solution connue du problème  $\mathcal{P}_{\mathcal{A}, \mathcal{C}_{p(i)}/\mathcal{C}_i}$ .

Si toutes les variables du cluster  $\mathcal{C}_i$  sont instanciées et que le minorant est inférieur au majorant (lignes 3-18),  $BTD_{val}$  poursuit son exploration avec le premier fils de  $\mathcal{C}_i$  (s'il en existe un). Plus généralement, considérons le cas d'un fils  $\mathcal{C}_j$  de  $\mathcal{C}_i$ .  $BTD_{val}$  teste si l'instanciation  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$  correspond à un good structurel valué (avec  $\mathcal{A}$  l'instanciation courante sur  $\mathcal{C}_i$ ) :

- Si c'est le cas, on agrège la valuation associée à ce good valué au minorant.
- Sinon, on doit étendre  $\mathcal{A}$  sur  $Desc(\mathcal{C}_j)$  afin de déterminer la valuation  $v$  de la meilleure affectation  $\mathcal{B}$  telle que  $\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ . Une fois cette valuation  $v$  calculée, on l'agrège au minorant et on mémorise le good structurel valué  $(\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], v)$ .

Si, après avoir traité le fils  $\mathcal{C}_j$  (et sa descendance), le minorant ne dépasse pas le majorant, on continue la recherche avec le prochain fils de  $\mathcal{C}_i$ . Lorsque tous les fils ont été examinés, si le minorant est inférieur au majorant, alors on a trouvé une solution meilleure pour le problème  $\mathcal{P}_{\mathcal{A}[\mathcal{C}_{p(i)} \cap \mathcal{C}_i], \mathcal{C}_{p(i)}/\mathcal{C}_i}$ . Enfin, si on rencontre un échec, alors il faut revenir en arrière et modifier l'instanciation courante sur  $\mathcal{C}_i$ .

Notons qu'en mémorisant et en exploitant les goods structurels valués, l'algorithme  $BTD_{val}$  ne résout qu'une seule fois chaque sous-problème. Aussi, l'emploi d'un good

structurel valué permet à  $\text{BTD}_{val}$  de ne pas instancier à nouveau les variables du sous-problème sur lequel porte le good. Un tel phénomène sera appelé un **forward-jump** (par analogie avec la notion de backjump). Par exemple, supposons que les variables soient ordonnées selon l'ordre alphabétique et qu'après avoir instancié la variable  $F$  dans  $\mathcal{C}_3$ ,  $\text{BTD}_{val}$  exploite un good structurel valué sur  $\mathcal{C}_3 \cap \mathcal{C}_4$ . Alors, après avoir instancié la variable  $F$ ,  $\text{BTD}_{val}$  essaiera d'affecter ensuite la variable  $I$ , et cela sans réexplorer le sous-problème formé par  $\text{Desc}(\mathcal{C}_4)$ . Du fait des forward-jumps, si on considère l'affectation  $\mathcal{A}$  sur  $\mathcal{C}_i$ , le minorant de  $\text{BTD}_{val}$  correspond en réalité à la valuation de la meilleure extension de  $\mathcal{A}$  sur la descendance de chacun des clusters fils de  $\mathcal{C}_i$  déjà examinés.

La figure 2 décrit l'algorithme  $\text{BTD}_{val}$ . Etant donné une instantiation  $\mathcal{A}$  et un cluster  $\mathcal{C}_i$ ,  $\text{BTD}_{val}$  recherche la meilleure instantiation  $\mathcal{B}$  sur  $\text{Desc}(\mathcal{C}_i)$  telle que  $\mathcal{A}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}]$  et  $v_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}, \mathcal{C}_{p(i)}/\mathcal{C}_i]}(\mathcal{B})} \prec \alpha_{\mathcal{C}_i}$ , où :

- $V_{\mathcal{C}_i}$  est l'ensemble des variables non instanciées dans  $\mathcal{C}_i$ ,
- $\alpha_{\mathcal{C}_i}$  est la valuation de la meilleure instantiation  $\mathcal{B}'$  sur  $\text{Desc}(\mathcal{C}_i)$  telle que  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$
- $l_{\mathcal{C}_i} = v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}) \prec \alpha_{\mathcal{C}_i}$ .

```

BTD-val( $\mathcal{A}, \mathcal{C}_i, V_{\mathcal{C}_i}, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i}$ )
1. If  $V_{\mathcal{C}_i} = \emptyset$ 
2. Then
3.   If  $\text{Fils}(\mathcal{C}_i) = \emptyset$  Then Retourner  $l_{\mathcal{C}_i}$ 
4.   Else
5.      $F \leftarrow \text{Fils}(\mathcal{C}_i)$ 
6.      $\alpha \leftarrow l_{\mathcal{C}_i}$ 
7.     While  $F \neq \emptyset$  and  $\alpha \prec \alpha_{\mathcal{C}_i}$  Do
8.       Choisir  $\mathcal{C}_j$  dans  $F$ 
9.        $F \leftarrow F \setminus \{\mathcal{C}_j\}$ 
10.      If  $(\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i], v)$  est un good de  $\mathcal{C}_i/\mathcal{C}_j$  dans  $G$  Then  $\alpha \leftarrow \alpha \oplus v$ 
11.      Else
12.         $v \leftarrow \text{BTD-val}(\mathcal{A}, \mathcal{C}_j, \mathcal{C}_j \setminus (\mathcal{C}_j \cap \mathcal{C}_i), \top, \perp)$ 
13.         $\alpha \leftarrow \alpha \oplus v$ 
14.        Enregistrer le good  $(\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i], v)$  de  $\mathcal{C}_i/\mathcal{C}_j$  dans  $G$ 
15.      EndIf
16.    EndWhile
17.    Retourner  $\alpha$ 
18.  EndIf
19. Else
20.  Choisir  $x \in V_{\mathcal{C}_i}$ 
21.   $d \leftarrow d_x$ 
22.  While  $d \neq \emptyset$  and  $l_{\mathcal{C}_i} \prec \alpha_{\mathcal{C}_i}$  Do
23.    Choisir  $a$  dans  $d$ 
24.     $d \leftarrow d \setminus \{a\}$ 
25.     $L \leftarrow \{c = \{x, y\} \in E_{\mathcal{P}, \mathcal{C}_i} \mid y \notin V_{\mathcal{C}_i}\}$ 
26.     $l_a \leftarrow \perp$ 
27.    While  $L \neq \emptyset$  and  $l_{\mathcal{C}_i} \oplus l_a \prec \alpha_{\mathcal{C}_i}$  Do
28.      Choisir  $c$  dans  $L$ 
29.       $L \leftarrow L \setminus \{c\}$ 
30.      If  $c$  viole  $\mathcal{A} \cup \{x \leftarrow a\}$  Then  $l_a \leftarrow l_a \oplus \phi(c)$ 
31.    EndWhile
32.    If  $l_{\mathcal{C}_i} \oplus l_a \prec \alpha_{\mathcal{C}_i}$ 
33.      Then  $\alpha_{\mathcal{C}_i} \leftarrow \min(\alpha_{\mathcal{C}_i}, \text{BTD-val}(\mathcal{A} \cup \{x \leftarrow a\}, \mathcal{C}_i, V_{\mathcal{C}_i} \setminus \{x\}, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i} \oplus l_a))$ 
34.    EndIf
35.  EndWhile
36.  Retourner  $\alpha_{\mathcal{C}_i}$ 
37. EndIf

```

FIG. 2 – L'algorithme  $\text{BTD}_{val}$ .

Si l'algorithme  $\text{BTD}_{val}$  trouve une telle instantiation, il retourne sa valuation, sinon il renvoie une valuation supérieure ou égale à  $\alpha_{\mathcal{C}_i}$ . L'appel initial est réalisé par  $\text{BTD}_{val}(\emptyset, \mathcal{C}_1, \mathcal{C}_1, \top, \perp)$ . Notons que la version de  $\text{BTD}_{val}$  présentée ici ne calcule que la valuation d'une solution optimale. Elle ne calcule pas explicitement la solution optimale. En effet, on peut remarquer que l'affectation  $\mathcal{A}$  ne contient en fait que les variables appartenant aux clusters localisés sur le chemin allant du cluster racine au cluster courant. Cependant, dans le cas où une solution optimale est recherchée, l'algorithme  $\text{BTD}_{val}$  peut être facilement adapté pour en fournir sans que les résultats de complexité présentés dans le théorème 3 ne s'en trouvent altérés.

**Théorème 2** *L'algorithme  $\text{BTD}_{val}$  est correct, complet et termine.*

*Preuve :*

La preuve s'effectue par induction sur le nombre de variables non instanciées de la descendance du cluster courant  $\mathcal{C}_i$ . L'ensemble de ces variables est noté  $\Delta_{\mathcal{C}_i, V_{\mathcal{C}_i}} = V_{\mathcal{C}_i} \cup \bigcup_{\mathcal{C}_j \in \text{Fils}(\mathcal{C}_i)} (\text{Desc}(\mathcal{C}_j) \setminus (\mathcal{C}_i \cap \mathcal{C}_j))$ .  $\Delta_{\mathcal{C}_i, V_{\mathcal{C}_i}}$  correspond, en fait, à l'ensemble des variables à instancier pour déterminer la valuation de la meilleure affectation  $\mathcal{B}$  sur  $\text{Desc}(\mathcal{C}_i)$  telle que  $\mathcal{A}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}]$ .

Pour montrer la correction de  $\text{BTD}_{val}$ , on doit prouver la propriété  $P(\mathcal{A}, \Delta_{\mathcal{C}_i, V_{\mathcal{C}_i}}, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i})$  définie ainsi :

"étant donné  $\alpha_{\mathcal{C}_i}$  la valuation de la meilleure affectation  $\mathcal{B}'$  connue telle que  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$  et  $l_{\mathcal{C}_i} = v_{\mathcal{P}, \mathcal{C}_i}(\mathcal{A}) \prec \alpha_{\mathcal{C}_i}$ ,  $\text{BTD}_{val}(\mathcal{A}, \mathcal{C}_i, V_{\mathcal{C}_i}, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i})$  renvoie :

- la valuation de la meilleure affectation  $\mathcal{B}$  sur  $\text{Desc}(\mathcal{C}_i)$  telle que  $\mathcal{A}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}]$  et  $v_{\mathcal{P}, \mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}], \mathcal{C}_{p(i)}/\mathcal{C}_i}(\mathcal{B}) \prec \alpha_{\mathcal{C}_i}$  si une telle affectation existe,
- une valuation supérieure ou égale à  $\alpha_{\mathcal{C}_i}$ , sinon".

Considérons  $P(\mathcal{A}, \emptyset, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i})$  :

Si  $\Delta_{\mathcal{C}_i, V_{\mathcal{C}_i}} = \emptyset$ , alors  $V_{\mathcal{C}_i} = \emptyset$  et  $\text{Fils}(\mathcal{C}_i) = \emptyset$ . Donc,  $\mathcal{C}_i \setminus V_{\mathcal{C}_i} = \mathcal{C}_i$  et  $l_{\mathcal{C}_i}$  est la valuation de la meilleure affectation  $\mathcal{B}$  sur  $\text{Desc}(\mathcal{C}_i)$  telle que  $\mathcal{A}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \setminus V_{\mathcal{C}_i}]$ . Comme  $\text{BTD}_{val}$  retourne  $l_{\mathcal{C}_i}$ ,  $P(\mathcal{C}_i, \Delta_{\mathcal{C}_i, V_{\mathcal{C}_i}}, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i})$  est vraie.

Pas d'induction :  $P(\mathcal{A}, S, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i})$  avec  $S \neq \emptyset$ . Supposons que  $\forall S' \subset S, P(\mathcal{A}, S', \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i})$  soit vérifiée.

- Si  $V_{\mathcal{C}_i} \neq \emptyset$  :

Durant la boucle **While** (lignes 22-35) l'assertion "pour chaque valeur  $a$  déjà examinée,  $\alpha_{\mathcal{C}_i}$  est la valuation de la meilleure affectation  $\mathcal{B}'$  connue telle que  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$ " est vérifiée.

Lorsque  $\text{BTD}_{val}$  est appelé (ligne 33), l'affectation considérée est  $\mathcal{A} \cup \{x \leftarrow a\}$  et on a  $l_{\mathcal{C}_i} \oplus l_a \prec \alpha_{\mathcal{C}_i}$  et  $\Delta_{\mathcal{C}_i, V_{\mathcal{C}_i} \setminus \{x\}} \subset \Delta_{\mathcal{C}_i, V_{\mathcal{C}_i}}$ . D'après l'hypothèse d'induction,  $\text{BTD}_{val}(\mathcal{A} \cup \{x \leftarrow a\}, \mathcal{C}_i, V_{\mathcal{C}_i} \setminus \{x\}, \alpha_{\mathcal{C}_i}, l_{\mathcal{C}_i} \oplus l_a)$  calcule la valuation de la meilleure affectation  $\mathcal{B}$  sur  $\text{Desc}(\mathcal{C}_i)$  telle que  $\mathcal{A}[\mathcal{C}_i \setminus (V_{\mathcal{C}_i} \setminus \{x\})] = \mathcal{B}[\mathcal{C}_i \setminus (V_{\mathcal{C}_i} \setminus \{x\})]$ . Soit  $\alpha$  cette valuation. La mise à jour de  $\alpha_{\mathcal{C}_i}$  (ligne 33) avec le minimum entre  $\alpha_{\mathcal{C}_i}$  et  $\alpha$  garantit que  $\alpha_{\mathcal{C}_i}$  est toujours la valuation de la meilleure affectation  $\mathcal{B}'$  connue telle que  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$ .

A l'issue de la boucle (ligne 36), pour les valeurs de  $d_x \setminus d$ ,  $\alpha_{\mathcal{C}_i}$  est la valuation de la meilleure affectation  $\mathcal{B}'$  connue telle que  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$ . Si  $d = \emptyset$ ,

comme  $\text{BTD}_{val}$  retourne  $\alpha_{C_i}$ ,  $P(\mathcal{A}, S, \alpha_{C_i}, l_{C_i})$  est vérifiée. Sinon, pour les valeurs de  $d$ , la condition  $l_{C_i} \prec \alpha_{C_i}$  est violée. Donc, toutes les affectations  $\mathcal{A} \cup \{x \leftarrow a\}$  (avec  $a \in d$ ) ont une valuation qui dépasse ou égale  $\alpha_{C_i}$ . Par conséquent,  $\alpha_{C_i}$  est la valuation de la meilleure affectation  $\mathcal{B}'$  connue telle que  $\mathcal{A}[C_i \cap C_{p(i)}] = \mathcal{B}'[C_i \cap C_{p(i)}]$ .  $\text{BTD}_{val}$  retournant  $\alpha_{C_i}$ ,  $P(\mathcal{A}, S, \alpha_{C_i}, l_{C_i})$  est vérifiée.

- Si  $V_{C_i} = \emptyset$  :

Durant la boucle **While** (lignes 7-16), l'assertion " $\alpha = v_{\mathcal{P}, C_i}(\mathcal{A}) \oplus \bigoplus_{C_j \in \text{Fils}(C_i) \setminus F} \alpha_{\mathcal{P}_{\mathcal{A}[C_i \cap C_j], C_i/C_j}}^*$ "

(avec  $F$  l'ensemble des fils de  $C_i$  déjà visités) est vérifiée.

Initialement, l'assertion est vérifiée puisque  $\alpha = l_{C_i}$ . Nous allons montrer que cette assertion est encore vraie à l'issue de la boucle.

Soit  $C_j$  un fils de  $C_i$  à examiner.

- + Si  $(\mathcal{A}[C_i \cap C_j], v)$  est un good valué de  $C_i$  par rapport à  $C_j$ ,  $v$  est la valuation de l'instanciation optimale  $\mathcal{B}$  sur  $\text{Desc}(C_j)$  telle que  $\mathcal{B}[C_i \cap C_j] = \mathcal{A}[C_i \cap C_j]$ . Donc, l'assertion est vérifiée.
- + Sinon,  $\text{BTD}_{val}$  est appelé avec  $\mathcal{A}$  et  $\Delta_{C_j, C_j \setminus (C_j \cap C_i)} \subset \Delta_{C_i, \emptyset}$ . Par conséquent, d'après l'hypothèse d'induction,  $\text{BTD}_{val}(\mathcal{A}, C_j, C_j \setminus (C_j \cap C_i), \top, \perp)$  retourne la valuation de l'instanciation optimale  $\mathcal{B}$  sur  $\text{Desc}(C_j)$  telle que  $\mathcal{B}[C_i \cap C_j] = \mathcal{A}[C_i \cap C_j]$ . L'assertion est alors vérifiée.

A l'issue de la boucle (ligne 17), l'assertion est vérifiée. Si  $\alpha \prec \alpha_{C_i}$ ,  $\alpha$ , d'après le théorème 1, est la meilleure valuation possible pour une affectation  $\mathcal{B}$  telle que  $\mathcal{A}[C_i] = \mathcal{B}[C_i]$ . Sinon, une telle affectation avec une valuation inférieure à  $\alpha_{C_i}$  n'existe pas.  $\text{BTD}_{val}$  retournant  $\alpha$ , la propriété  $P(\mathcal{A}, \Delta_{C_i, V_{C_i}}, \alpha_{C_i}, l_{C_i})$  est satisfaite. Notons que la mémorisation des goods valués se justifie par leur définition.

Pour résumer, comme  $\text{BTD}_{val}$  satisfait la propriété  $P(\mathcal{A}, \Delta_{C_i, V_{C_i}}, \alpha_{C_i}, l_{C_i})$ , et en particulier la propriété  $P(\emptyset, \Delta_{C_1, V_{C_1}}, \top, \perp)$  pour le premier appel,  $\text{BTD}_{val}$  est correct, complet et termine.  $\square$

A présent, nous allons nous intéresser aux complexités en temps et en espace de  $\text{BTD}_{val}$ . Dans ce but, nous supposons qu'une décomposition arborescente (ou une approximation) a déjà été calculée. Aussi, les paramètres utilisés dans le prochain théorème sont relatifs à cette décomposition. L'algorithme  $\text{BTD}_{val}$  obtient des complexités similaires à celles des algorithmes Tree-Clustering et  $\text{BTD}$  définis dans le cadre classique :

**Théorème 3**  *$\text{BTD}_{val}$  a une complexité en temps en  $O(n.s^2.m.\log(d).d^{w^++1})$  pour une complexité en espace en  $O(n.s.d^s)$  avec  $w^+ + 1$  la taille du plus grand cluster  $C_k$  et  $s$  la taille de la plus grande intersection  $C_i \cap C_j$ ,  $C_j$  étant un fils de  $C_i$ .*

*Preuve :*

$\text{BTD}_{val}$  ne mémorise que les goods. Les goods sont des instanciations sur les intersections  $C_i \cap C_j$  avec  $C_j$  fils de  $C_i$ . Donc, si  $s$  est la taille de la plus grande de ces intersections,  $\text{BTD}_{val}$  a une complexité en espace en  $O(n.s.d^s)$  parce que le nombre d'intersections  $C_i \cap C_j$  est majoré par  $n$  tandis que le nombre de goods pour une intersection donnée est majoré par  $d^s$  et la taille d'un good est au plus  $s$ .

Maintenant, nous allons calculer la complexité en temps de  $\text{BTD}_{val}$ . Supposons que nous souhaitons étendre une instanciation sur  $C_j$ . Il existe deux cas :



- Soit  $\mathcal{C}_j = \mathcal{C}_1$ . Trouver toutes les instanciations sur  $\mathcal{C}_j$  telles que le minorant ne dépasse pas le majorant s'avère alors, dans le pire des cas, de complexité en  $O(m.d^{|\mathcal{C}_j|})$ .
- Soit  $\mathcal{C}_j$  est un fils de  $\mathcal{C}_i$ . Soit  $\mathcal{A}$  une affectation sur  $Y$  ( $Y \subset X$  tel que  $Desc(\mathcal{C}_j) \cap Y = \mathcal{C}_i \cap \mathcal{C}_j$ ).  
 Trouver toutes les extensions de  $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$  sur  $\mathcal{C}_j$  telles que le minorant ne dépasse pas le majorant s'avère, dans le pire des cas, de complexité en  $O(m.d^{|\mathcal{C}_j \setminus (\mathcal{C}_j \cap \mathcal{C}_i)|})$ .  
 $BTD_{val}$  ne recherche de telles extensions de  $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$  qu'une seule fois (grâce aux goods valués mémorisés). Comme il existe au plus  $d^{|\mathcal{C}_i \cap \mathcal{C}_j|}$  affectations  $\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i]$ , la complexité en temps pour trouver une extension consistante sur  $\mathcal{C}_j$  est, dans le pire des cas, en  $O(d^{|\mathcal{C}_j|})$ .

Donc, si  $w^+ + 1$  est la taille du plus grand  $\mathcal{C}_i$ , le calcul de la valuation du VCSP par  $BTD_{val}$  a une complexité en  $O(n.m.d^{w^++1})$ , à laquelle doit être ajouté le coût de la gestion et de l'exploitation des goods. Comme ce coût est nul pour  $\mathcal{C}_1$ , nous nous focalisons sur le cas où  $\mathcal{C}_j$  est un fils de  $\mathcal{C}_i$ .

La comparaison entre  $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$  et un good mémorisé requiert  $O(|\mathcal{C}_i \cap \mathcal{C}_j|)$ . L'ajout ou la recherche d'un good est en  $O(|\mathcal{C}_i \cap \mathcal{C}_j| \log(d^{|\mathcal{C}_i \cap \mathcal{C}_j|}))$ . Par conséquent, la gestion et l'exploitation des goods ont une complexité en  $O(d^{|\mathcal{C}_i|} |\mathcal{C}_i \cap \mathcal{C}_j| \log(d^{|\mathcal{C}_i \cap \mathcal{C}_j|}))$ , étant donné  $\mathcal{C}_i$  et un des ces fils  $\mathcal{C}_j$ .

Donc, sur l'ensemble de la recherche, le coût est en  $O(n.s.m.d^{w^++1} \log(d^s))$ .

Ainsi, la complexité en temps de  $BTD_{val}$  est en  $O(n.m.d^{w^++1} + n.s.m.d^{w^++1} \log(d^s))$ , i.e. une complexité en  $O(n.s^2.m.d^{w^++1} \log(d))$ .  $\square$

## 5 Discussion

La méthode  $BTD_{val}$  repose essentiellement sur la notion de décomposition arborescente. Par conséquent, des travaux comme ceux portant sur la méthode Tree-Clustering et ses améliorations [Dechter et Pearl, 1989; Dechter et Fattah, 2001] ou ceux s'intéressant à l'approche par programmation dynamique [Koster, 1999] sont relativement proches de notre approche. L'algorithme  $BTD_{val}$  peut être considéré comme une approche hybride réalisant un compromis entre le temps et l'espace. Dans [Dechter et Fattah, 2001], Dechter et El Fattah proposent aussi un compromis entre le temps et l'espace. Ce compromis leur permet de définir toute une gamme d'algorithmes dont le tree-clustering et la méthode coupe-cycle (dont la complexité en espace est linéaire) sont les deux extrêmes. Une autre idée intéressante dans leur travail consiste à modifier la taille des séparateurs afin de minimiser la quantité de mémoire requise. Cette idée peut (et devra) aussi être exploitée lors de la mise en œuvre de la méthode  $BTD_{val}$ .

Par rapport à la méthode par programmation dynamique de Koster [Koster, 1999],  $BTD_{val}$  présente d'une part une meilleure borne de complexité en temps. D'autre part,  $BTD_{val}$  diffère de cette approche au niveau du calcul d'une décomposition arborescente (ou de son approximation). En effet,  $BTD_{val}$  exploite une triangulation du graphe de contraintes tandis que l'approche de Koster utilise une heuristique et des techniques de résolution de problème de flots dans les réseaux. De plus, Koster propose plusieurs prétraitements dont l'objectif est de réduire la taille du problème à résoudre. En particulier, un de ces prétraitements permet de réduire la taille du graphe de contraintes, ce qui

peut induire une réduction aussi bien de la complexité en temps que de la complexité en espace. Aussi, d'un point de vue pratique, il pourrait être particulièrement intéressant pour notre méthode d'intégrer de tels prétraitements.

L'algorithme  $\text{BTD}_{val}$  est également proche de la méthode des poupées russes [Verfaillie *et al.*, 1996]. En effet, afin de déterminer la valuation optimale d'un VCSP,  $\text{BTD}_{val}$  résout plusieurs sous-problèmes suivant un ordre compatible préétabli. Les clusters de  $\text{BTD}_{val}$  jouent alors un rôle similaire à celui des variables dans RDS. Néanmoins, les deux méthodes exploitent différemment les valuations optimales des sous-problèmes. Parmi les variantes et améliorations de RDS, la méthode Tree-RDS [Meseguer et Sánchez, 2000] tire partie du graphe de contraintes pour déterminer si certains sous-problèmes sont indépendants ou non. Cependant, si la notion d'indépendance de deux sous-problèmes est relativement proche de celle de  $\text{BTD}_{val}$ , les sous-problèmes de Tree-RDS sont conceptuellement différents des nôtres. Il en est de même pour l'adaptation [Larrosa *et al.*, 2002] de l'algorithme Pseudo-Tree Search et pour sa combinaison avec une variante de RDS.

## 6 Conclusion

Dans ce papier, nous avons défini une nouvelle méthode (appelé  $\text{BTD}_{val}$ ) pour la résolution de CSP valués. Cette méthode peut actuellement reposer sur BB ou sur certaines de ses améliorations comme l'algorithme  $\text{FC}_{val}$ . Grâce à la notion de goods structurels valués que nous avons introduite, l'algorithme  $\text{BTD}_{val}$  obtient des bornes de complexité équivalentes aux meilleures bornes connues. En effet,  $\text{BTD}_{val}$  possède une complexité en temps en  $O(ns^2m \log(d).d^{w^++1})$  avec  $w^+ + 1$  la taille du plus grand cluster pour une complexité en espace en  $O(nsd^s)$  avec  $s$  la taille de la plus grande intersection entre deux clusters. D'un point de vue pratique, l'intérêt de notre approche reste à démontrer. Cependant, les bons résultats obtenus par la méthode BTM [Jégou et Terrioux, 2003] dans le cadre classique sont très encourageants puisque BTM se révèle soit aussi efficace, soit plus efficace que les algorithmes énumératifs classiques tout en consommant une quantité de mémoire raisonnable. Nous avons, par ailleurs, déjà obtenu quelques résultats préliminaires du même ordre sur des VCSP aléatoires structurés. L'approche semble donc prometteuse et une étude expérimentale complète devra être menée afin d'évaluer réellement son intérêt pratique dans le cadre des CSP valués.

Parmi les différentes extensions possibles de ce travail, l'utilisation comme algorithme de base de méthodes plus performantes que BB ou  $\text{FC}_{val}$  devra être étudiée. En particulier, vouloir intégrer l'algorithme des poupées russes ou des algorithmes exploitant de la consistance d'arc directionnelle [Wallace, 1994; 1996; Larrosa et Meseguer, 1996; Larrosa *et al.*, 1999] semble être une extension naturelle par rapport à la notion d'ordre d'énumération compatible employée par notre méthode.

## Références

- [Arnborg *et al.*, 1987] S. Arnborg, D. Corneil, et A. Proskuroski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics*, 8 :277–284, 1987.
- [Becker et Geiger, 2001] A. Becker et D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125 :3–17, 2001.

- [Bistarelli *et al.*, 1995] S. Bistarelli, U. Montanari, et F. Rossi. Constraint solving over semirings. Dans *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 624–630, 1995.
- [Cooper et Schiex, 2004] M. Cooper et T. Schiex. Arc consistency for soft constraints. *Artificial Intelligence*, 2004. A paraître.
- [Dago et Verfaillie, 1996] P. Dago et G. Verfaillie. Nogood Recording for Valued Constraint Satisfaction Problems. Dans *Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pages 132–139, 1996.
- [Dechter *et al.*, 2001] R. Dechter, K. Kask, et J. Larrosa. A General Scheme for Multiple Lower Bound Computation in Constraint Optimization. Dans *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP'2001)*, pages 346–360, 2001.
- [Dechter et Fattah, 2001] R. Dechter et Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125 :93–118, 2001.
- [Dechter et Pearl, 1989] R. Dechter et J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38 :353–366, 1989.
- [Dechter, 1997] R. Dechter. Bucket-Elimination : A Unifying Framework For Processing Hard and Soft Constraints. *Constraints*, 2 :51–55, 1997.
- [Gottlob *et al.*, 2000] G. Gottlob, N. Leone, et F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.
- [Jégou et Terrioux, 2002] P. Jégou et C. Terrioux. Recherche arborescente bornée. Dans *Actes des 8èmes Journées Nationales sur la résolution Pratique de Problèmes NP-Complets (JNPC'2002)*, pages 127–141, Nice, 2002.
- [Jégou et Terrioux, 2003] P. Jégou et C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.
- [Koster, 1999] A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, Novembre 1999.
- [Larrosa *et al.*, 1999] J. Larrosa, P. Meseguer, et T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1) :149–163, 1999.
- [Larrosa *et al.*, 2002] J. Larrosa, P. Meseguer, et M. Sánchez. Pseudo-Tree Search with Soft Constraints. Dans *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'2002)*, pages 131–135, 2002.
- [Larrosa et Dechter, 2003] J. Larrosa et R. Dechter. Boosting Search with Variable Elimination in Constraint Optimization and Constraint Satisfaction Problems. *Constraints*, 8(3) :303–326, 2003.
- [Larrosa et Meseguer, 1996] J. Larrosa et P. Meseguer. Exploiting the use of DAC in Max-CSP. Dans *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (CP'96)*, pages 308–322, 1996.
- [Larrosa et Schiex, 2003] J. Larrosa et T. Schiex. In the quest of the best form of local consistency for Weighted CSP. Dans *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 239–244, 2003.
- [Larrosa, 2002] J. Larrosa. Node and Arc Consistency in Weighted CSP. Dans *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-2002)*, pages 48–53, 2002.

- [Meseguer et Sánchez, 2000] P. Meseguer et M. Sánchez. Tree-based Russian Doll Search. Dans *Proceedings of the Workshop on soft constraint*. CP'2000, 2000.
- [Meseguer et Sánchez, 2001] P. Meseguer et M. Sánchez. Specializing Russian Doll Search. Dans *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP'2001)*, volume LNCS 2239, pages 464–478, 2001.
- [Robertson et Seymour, 1986] N. Robertson et P.D. Seymour. Graph minors II : Algorithmic aspects of tree-width. *Algorithms*, 7 :309–322, 1986.
- [Schiex *et al.*, 1995] T. Schiex, H. Fargier, et G. Verfaillie. Valued Constraint Satisfaction Problems : hard and easy problems. Dans *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 631–637, 1995.
- [Schiex, 2002] T. Schiex. Une comparaison des cohérences d'arc dans les Max-CSP. Dans *Actes des 8èmes Journées Nationales sur la résolution Pratique de Problèmes NP-Complets (JNPC'2002)*, pages 209–223, 2002.
- [Verfaillie *et al.*, 1996] G. Verfaillie, M. Lemaître, et T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. Dans *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 181–187, 1996.
- [Wallace, 1994] R. Wallace. Directed arc consistency preprocessing. Dans *Proceedings of the ECAI-94 Workshop on Constraint Processing, LNCS 923*, pages 121–137, 1994.
- [Wallace, 1996] R. Wallace. Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem. Dans *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 188–195, 1996.