Rapport de recherche du LSIS (UMR CNRS 6168) Marseille France

Recherche arborescente bornée pour la résolution de CSP valués Rapport de Recherche Numéro LSIS/2003/003, 11 Mars 2003

Philippe Jégou, Cyril Terrioux

Laboratoire des Sciences de l'Information et des Systèmes
LSIS (UMR CNRS 6168)
Campus Scientifique de St Jérôme
avenue Escadrille Normandie Niemen
13397 MARSEILLE Cedex 20

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 : l'efficacité pratique de l'énumération et la garantie de borne de complexité en temps. 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 "*nogoods structurels valués*". La mémorisation et l'emploi de ces nogoods 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 [8, 9] proposée dans le cadre des CSP classiques.

Mots-clés : Problème de satisfaction de contraintes valué, décomposition arborescente.

---

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 [3, 15] du formalisme CSP ont été proposées parmi lesquelles les CSP valués (VCSP [15]). 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 meilleurs résultats sont souvent fournis par des méthodes comme la méthode des poupées russes (notée RDS pour Russian Doll Search [17]) ou par des méthodes issues de la programmation dynamique [10]. 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 [8, 9] 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 structurelles. La notion de décomposition arborescente permet à BTD-val de diviser le problème initial en plusieurs sous-problèmes. BTD-val mémorise alors le résultat de la résolution de chaque sous-problème sous la forme de *nogoods structurels valués*. Ces nogoods 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 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 $\{x_1, \ldots, x_n\}$ 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 $\{x_{c_1}, \ldots, x_{c_k}\}$ 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 $d_{x_{c_1}} \times \cdots \times d_{x_{c_k}}$. Etant donné $Y \subseteq X$ tel que $Y = \{x_1, \ldots, x_k\}$, une **instanciation** des variables de Y est un tuple $\mathcal{A} = (v_1, \ldots, v_k)$ de $d_{x_1} \times \cdots \times d_{x_k}$. Une contrainte c est dite **satisfaite** par $\mathcal{A}$ si $c \subseteq Y, (v_1, \ldots, v_k)[c] \in r_c$, **violée** sinon ($\mathcal{A}[c]$ désignant la restriction de l'affectation $\mathcal{A}$ aux variables de c). Par la suite, on notera une affectation $(v_1, \ldots, v_k)$ sous la forme plus explicite $(x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k)$.
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 ([15]) *Une **structure de valuation** est un triplet $(E, \preceq, \oplus)$ avec un ensemble E de valuations totalement ordonné par $\preceq$, muni d'un élément minimum (noté $\bot$), d'un élément maximum (noté $\top$) et d'une loi de composition interne (notée $\oplus$) commutative, associative, monotone et telle que $\bot$ soit un élément neutre pour $\oplus$ et $\top$ un élément absorbant.*

Les valuations permettent d'exprimer différents niveaux de violation. Par exemple, $\bot$ 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 ([15]) *Un **CSP valué** (VCSP) est un CSP classique $\mathcal{P} = (X, D, C, R)$ doté d'une structure de valuation $S = (E, \preceq, \oplus)$ et d'une application ϕ de C dans E, qui associe une valuation à chaque contrainte du CSP. On le note comme un sextuplet (X, D, C, R, S, ϕ).*
*Il est dit **binaire** si chaque contrainte de C implique au plus deux variables.*

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

Définition 3 ([15]) *Soient un VCSP $\mathcal{P} = (X, D, C, R, S, \phi)$ et une instanciation $\mathcal{A}$ 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 $c_{xy} = \{x, y\}$ a 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 $\alpha^*_{\mathcal{P}} = 2$.
Cette notion de valuation d'une affectation complète peut être étendue à des instanciations partielles :

Définition 4 ([15]) *Soient un VCSP $\mathcal{P} = (X, D, C, R, S, \phi)$ et une instanciation $\mathcal{A}$ sur $Y \subset X$. La **valuation locale** de $\mathcal{A}$ dans $\mathcal{P}$ se définit par :*

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

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

Propriété 1 ([15]) *Soit un VCSP $\mathcal{P} = (X, D, C, R, S, \phi)$. Soient $\mathcal{A}$ une instanciation 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 [15]), Forward-Checking with Nogood Recording [4], ... Cependant, actuellement, les résultats les plus prometteurs sont fournis par des algorithmes comme la méthode des poupées russes (notée RDS pour Russian Doll Search [17]) ou des méthodes basées sur l'approche par programmation dynamique [10]. Ces méthodes divisent le problème en plusieurs sous-problèmes et exploitent les informations qu'elles produisent durant la résolution de chaque sous-problème.

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 [14], dont la définition est rappelée ci-dessous :

Définition 5 ([14]) *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 $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ une famille de sous-ensembles de X, telle que chaque élément $\mathcal{C}_i$ correspond à un nœud de $\mathcal{T}$ et vérifie :*

(1) $\bigcup_{i \in I} \mathcal{C}_i = X$,

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

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

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

Les éléments $\mathcal{C}_i$ 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.

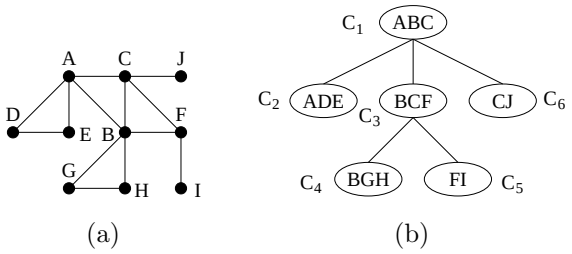

FIG. 1 – (a) Un graphe de contraintes avec 10 variables. (b) Une décomposition arborescente de ce graphe de contraintes.

Le problème de recherche d'une décomposition arborescente est NP-dur [1]. Toutefois, de nombreux travaux ont été développés dans cette direction [2]. 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 $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ 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 $\mathcal{C}_1 = \{A, B, C\}$, $\mathcal{C}_2 = \{A, D, E\}$, $\mathcal{C}_3 = \{B, C, F\}$, $\mathcal{C}_4 = \{B, G, H\}$, $\mathcal{C}_5 = \{F, I\}$ et $\mathcal{C}_6 = \{C, J\}$.

La notion de décomposition arborescente est exploitée dans le cadre des CSP classiques par plusieurs méthodes structurelles (voir [7] 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* [6, 5] 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 [8, 9]. 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 [10] 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 BTD-val

4.1. Présentation

Comme BTD, la méthode 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 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 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 *nogood structurel valué*).
Notons que si, dans notre description, BTD-val repose sur l'algorithme Branch and Bound, nous pouvons également utiliser des variantes plus sophistiquées comme, par exemple, l'algorithme FC-val.

4.2. Justifications formelles

Dans toute cette partie, nous ne fournirons aucune preuve (toutefois celles-ci peuvent être consultées dans [16]).
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 $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ 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 $\mathcal{C}_1$ 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 $\forall x \in \mathcal{C}_i, \forall y \in \mathcal{C}_j$, avec $i < j$, $x \preceq_X y$.*

Pour notre exemple, l'ordre alphabétique $A, B, \ldots, I, J$ est un ordre d'énumération compatible. La décomposition arborescente avec la numérotation N_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.*

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 pourrons alors garantir que la méthode 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 :*

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

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 $Y \subset X$ avec $Y \cap \mathcal{C}_i \neq \emptyset$. 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 $\mathcal{C}_i \subseteq Y$.*

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

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.*

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

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 $v_{\mathcal{P},\mathcal{C}_j}(\mathcal{B}) = v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B})$.*

Nous pouvons désormais définir la notion de nogood structurel valué.

Définition 11 *Soient $\mathcal{C}_i$ et $\mathcal{C}_j$ deux clusters avec $\mathcal{C}_j$ fils de $\mathcal{C}_i$. Un **nogood 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 nogood $(\mathcal{A}, 2)$. Notons que cette notion de nogood structurel valué englobe simultanément les notions de good et de nogood structurels du cadre classique [8, 9]. 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 nogood structurel valué.

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

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

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 $\mathcal{C}_i$ et la valuation optimale de chaque sous-problème enraciné en un fils $\mathcal{C}_j$ de $\mathcal{C}_i$.

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

$$\alpha^*_{\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}} = \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = \mathcal{C}_i \\ et\ \ \mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}}} \left(v_{\mathcal{P},\mathcal{C}_i}(\mathcal{B}) \oplus \bigoplus_{\mathcal{C}_j\ \ fils\ \ de\ \ \mathcal{C}_i} \alpha^*_{\mathcal{P}_{\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j],\mathcal{C}_i/\mathcal{C}_j}} \right)$$

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 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 $\mathcal{C}_i$, il procède de façon classique à la manière de BB en affectant une valeur à une variable, en maintenant et en comparant des bornes inférieures et superieures et en revenant en arrière si la valeur d'un minorant dépasse ou égale celle d'un majorant. Cependant, à la difference de BB, BTD-val emploient deux types de borne : des bornes locales et des bornes globales. Les bornes locales 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 local 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 local 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 $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$. Les bornes globales sont similaires à celles de BB. Le minorant global correspond ainsi à la valuation locale de l'affectation courante tandis que le majorant est donné par la valuation de la meilleure solution connue pour le problème $\mathcal{P}$.

Si toutes les variables du cluster $\mathcal{C}_i$ sont instanciées et que chaque minorant est inférieur au majorant qui lui correspond, 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 nogood structurel valué (avec $\mathcal{A}$ l'instanciation courante sur $\mathcal{C}_i$) :

- Si c'est le cas, on agrège la valuation associée à ce nogood valué à chaque 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 aux deux minorants et on mémorise le nogood structurel valué $(\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], v)$.

Si, après avoir traité le fils $\mathcal{C}_j$ (et sa descendance), les minorants ne dépassent pas leur majorant respectif, on continue la recherche avec le prochain fils de $\mathcal{C}_i$. Lorsque tous les fils ont été examinés, si chaque minorant est inférieur à son majorant respectif, 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 nogoods structurels valués, l'algorithme BTD-val ne résout qu'une seule fois chaque sous-problème. Aussi, l'emploi d'un nogood structurel valué permet à BTD-val de ne pas instancier à nouveau les variables du sous-problème sur lequel porte le nogood. 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 que BTD-val exploite un nogood structurel valué sur $\mathcal{C}_1 \cap \mathcal{C}_3$. Alors, après avoir instancié la variable F, BTD-val essaiera d'affecter ensuite la variable J, et cela sans réexplorer le sous-problème formé par $Desc(\mathcal{C}_3)$ et donc sans instancier à nouveau les variables G, H et I. Du fait des forward-jumps, le minorant global correspond en réalité à la valuation de la meilleure extension de $\mathcal{A}$ sur chacun des clusters qui précèdent le cluster courant dans la numérotation compatible. Il en est de même pour le minorant local, mais ne ne tenant compte que des clusters appartenant à la descendance du cluster courant. Notons que nous considérons des extensions de l'affectation $\mathcal{A}$, et non $\mathcal{A}$ directement, car $\mathcal{A}$ ne contient en fait que les variables appartenant aux clusters localisés sur le chemin allant du cluster racine au cluster courant. Finalement, remarquons que le majorant global est le même que celui de BB alors que le minorant global s'avère meilleur que celui de BB puisque nous considérons la meilleure extension de $\mathcal{A}$.

La figure 2 décrit l'algorithme BTD-val. Etant donnés une instanciation $\mathcal{A}$ et un cluster $\mathcal{C}_i$, BTD-val recherche la meilleure instanciation $\mathcal{B}$ sur $Desc(\mathcal{C}_i)$ telle que $\mathcal{A}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}]$ 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 $\mathcal{C}_i$,
- $\alpha_{\mathcal{C}_1}$ est la valuation de la meilleure solution connue pour $\mathcal{P}$,
- l_{tot} est la valuation de la meilleure extension $\mathcal{A}'$ de $\mathcal{A}$ sur tous les clusters qui précèdent $\mathcal{C}_i$ suivant la numérotation compatible ($l_{tot} = v_{\mathcal{P}}(\mathcal{A}') \prec \alpha_{\mathcal{C}_1}$),

- $\alpha_{\mathcal{C}_i}$ est la valuation de la meilleure instanciation $\mathcal{B}'$ sur $Desc(\mathcal{C}_i)$ telle que $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$
- $l_{\mathcal{C}_i} = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \prec \alpha_{\mathcal{C}_i}$.

Si l'algorithme BTD-val trouve une telle instanciation, il retourne sa valuation, sinon il renvoie une valuation supérieure ou égale à $\alpha_{\mathcal{C}_i}$. L'appel initial est réalisé par BTD-val$(\emptyset, \mathcal{C}_1, \mathcal{C}_1, \top, \bot, \top, \bot)$. Notons que la version de BTD-val présentée ici ne calcule pas explicitement la solution optimale. Elle calcule la valuation d'une solution optimale. Pour le cas où une solution optimale est recherchée, l'algorithme BTD-val peut être facilement adapté sans que les résultats de complexité s'en trouvent altérés.

BTD-val$(\mathcal{A}, \mathcal{C}_i, V_{\mathcal{C}_i}, l_{tot}, \alpha_{\mathcal{C}_1}, l_{\mathcal{C}_i}, \alpha_{\mathcal{C}_i})$
1. **If** $V_{\mathcal{C}_i} = \emptyset$
2. **Then**
3. **If** $Fils(\mathcal{C}_i) = \emptyset$ **Then** Retourner $l_{\mathcal{C}_i}$
4. **Else**
5. $F \leftarrow Fils(\mathcal{C}_i)$
6. $\alpha \leftarrow \bot$
7. **While** $F \neq \emptyset$ **and** $\alpha \oplus l_{tot} \prec \alpha_{\mathcal{C}_1}$ **and** $\alpha \oplus l_{\mathcal{C}_i} \prec \alpha_{\mathcal{C}_i}$ **Do**
8. Choisir $\mathcal{C}_j$ dans F
9. $F \leftarrow F \backslash \{\mathcal{C}_j\}$
10. **If** $(\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i], v)$ est un nogood de $\mathcal{C}_i/\mathcal{C}_j$ dans N **Then** $\alpha \leftarrow \alpha \oplus v$
11. **Else**
12. $v \leftarrow$ BTD-val$(\mathcal{A}, \mathcal{C}_j, \mathcal{C}_j \backslash (\mathcal{C}_j \cap \mathcal{C}_i), l_{tot} \oplus \alpha, \alpha_{\mathcal{C}_1}, \bot, \alpha_{\mathcal{C}_i})$
13. $\alpha \leftarrow \alpha \oplus v$
14. Enregistrer le nogood $(\mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i], v)$ de $\mathcal{C}_i/\mathcal{C}_j$ dans N
15. **EndIf**
16. **EndWhile**
17. Retourner $\alpha \oplus l_{\mathcal{C}_i}$
18. **EndIf**
19. **Else**
20. Choisir $x \in V_{\mathcal{C}_i}$
21. $d \leftarrow d_x$
22. **While** $d \neq \emptyset$ **and** $l_{tot} \prec \alpha_{\mathcal{C}_1}$ **and** $l_{\mathcal{C}_i} \prec \alpha_{\mathcal{C}_i}$ **Do**
23. Choisir a dans d
24. $d \leftarrow d \backslash \{a\}$
25. $L \leftarrow \{c = \{x, y\} \in E_{\mathcal{P},\mathcal{C}_i} | y \notin V_{\mathcal{C}_i}\}$
26. $l_a \leftarrow \bot$
27. **While** $L \neq \emptyset$ **and** $l_{tot} \oplus l_a \prec \alpha_{\mathcal{C}_1}$ **and** $l_{\mathcal{C}_i} \oplus l_a \prec \alpha_{\mathcal{C}_i}$ **Do**
28. Choisir c dans L
29. $L \leftarrow L \backslash \{c\}$
30. **If** c viole $\mathcal{A} \cup \{x \leftarrow a\}$ **Then** $l_a \leftarrow l_a \oplus \ \phi(c)$
31. **EndWhile**
32. **If** $l_{tot} \oplus l_a \prec \alpha_{\mathcal{C}_1}$ **and** $l_{\mathcal{C}_i} \oplus l_a \prec \alpha_{\mathcal{C}_i}$
33. **Then** $\alpha_{\mathcal{C}_i} \leftarrow \min(\alpha_{\mathcal{C}_i},$ BTD-val$(\mathcal{A} \cup \{x \leftarrow a\}, \mathcal{C}_i, V_{\mathcal{C}_i} \backslash \{x\}, l_{tot} \oplus l_a,$
$\alpha_{\mathcal{C}_1}, l_{\mathcal{C}_i} \oplus l_a, \alpha_{\mathcal{C}_i}))$
34. **EndIf**
35. **EndWhile**
36. Retourner $\alpha_{\mathcal{C}_i}$
37. **EndIf**

FIG. 2 – L'algorithme BTD-val.

Théorème 2 *L'algorithme BTD-val est correct, complet et termine.*

4.4. Complexités en temps et en espace

Dans cette section, 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 BTD-val obtient des complexités similaires à celles de l'algorithme Tree-Clustering :

Théorème 3 *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* $\mathcal{C}_k$ *et* s *la taille de la plus grande intersection* $\mathcal{C}_i \cap \mathcal{C}_j$, $\mathcal{C}_j$ *étant un fils de* $\mathcal{C}_i$.

A présent, nous allons montrer que BTD-val développe au plus autant de nœuds que l'algorithme BB. Afin de rendre possible une telle comparaison, nous supposons que l'algorithme BB exploite le même ordre d'instanciation pour les variables et pour les valeurs que BTD-val. Une telle hypothèse nous permet de faciliter la comparaison entre BB et BTD-val. Cependant, il est évident qu'un ordre compatible ne correspond pas nécessairement au meilleur ordre possible pour BB. Aussi, dans le futur, il serait souhaitable d'étendre cette comparaison afin de pouvoir exploiter des ordres différents.

Théorème 4 *Etant donné un ordre compatible, BTD-val développe au plus autant de nœuds que BB.*

Preuve : BTD-val emploie deux tests entre un minorant et un majorant. Une de ces deux comparaisons (à savoir celle concernant les bornes globales) se révèle meilleure que celle utilisée par BB. De plus, l'exploitation des nogoods structurels valués permet à BTD-val d'éviter certaines redondances dans la recherche. Par conséquent, BTD-val développe au plus autant de nœuds que l'algorithme BB. □

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 [6, 5] ou ceux s'intéressant à l'approche par programmation dynamique [10] 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 [5], 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 [10], 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 par programmation dynamique 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 BTD-val est également proche de la méthode des poupées russes [17]. En effet, afin de déterminer la valuation optimale d'un VCSP, BTD-val résout plusieurs sous-problèmes suivant un ordre compatible préétabli. Les clusters de 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 [13] 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 BTD-val, les sous-problèmes de Tree-RDS sont conceptuellement différents des nôtres. Il en est de même pour l'adaptation [12] 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 BTD-val) pour la résolution de CSP valués. Cette méthode peut actuellement reposer sur BB ou sur certaines de ses améliorations comme l'algorithme FC-val. Grâce à la notion de nogoods structurels valués que nous avons introduite, l'algorithme BTD-val obtient des bornes de complexité équivalentes aux meilleures bornes connues. En effet, 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. Ensuite, nous avons établi théoriquement que BTD-val développe au plus autant de nœuds que BB. D'un point de vue pratique, l'intérêt de notre approche reste à démontrer. Cependant, les bons résultats obtenus par la méthode BTD [9] dans le cadre classique sont très encourageants puisque BTD se révèle soit aussi efficace, soit plus efficace que les algorithmes énumératifs classiques tout en consommant une quantité de mémoire raisonnable. L'approche semble donc prometteuse et une étude expérimentale 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 FC-val devra être étudiée. En particulier, vouloir intégrer l'algorithme des poupées russes ou des algorithmes exploitant de la consistance d'arc directionnelle [18, 19, 11] semble être une extension naturelle par rapport à la notion d'ordre d'énumération compatible employée par notre méthode.

Rfrences

1. S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embedding in a k-tree. *SIAM Journal of Discrete Mathematics*, 8 :277–284, 1987.
2. A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125 :3–17, 2001.
3. S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 624–630, 1995.
4. P. Dago and G. Verfaillie. Nogood Recording for Valued Constraint Satisfaction Problems. In *Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pages 132–139, 1996.
5. R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125 :93–118, 2001.
6. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38 :353–366, 1989.
7. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.
8. P. Jégou and C. Terrioux. Recherche arborescente bornée. In $8^{ème}$ *Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC'2002)*, pages 127–141, Nice, 2002.
9. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 2003. À paraître.
10. A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, Novembre 1999.
11. J. Larrosa and P. Meseguer. Exploiting the use of DAC in Max-CSP. In *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (CP'96)*, pages 308–322, 1996.
12. J. Larrosa, P. Meseguer, and M. Sánchez. Pseudo-Tree Search with Soft Constraints. In *Proc. of the 15^{th} European Conference on Artificial Intelligence (ECAI 2002)*, pages 131–135, 2002.
13. P. Meseguer and M. Sánchez. Tree-based Russian Doll Search. In *Proceedings of the Workshop on soft constraint*. CP'2000, 2000.
14. N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of tree-width. *Algorithms*, 7 :309–322, 1986.
15. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems : hard and easy problems. In *Proceedings of the 14^{th} International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 631–637, 1995.

16. C. Terrioux. *Approches structurelles et coopératives pour la résolution des problèmes de satisfaction de contraintes*. PhD thesis, Université de Provence, Décembre 2002.
17. G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 181–187, 1996.
18. R. Wallace. Directed arc consistency preprocessing. In *Proceedings of the ECAI-94 Workshop on Constraint Processing, LNCS 923*, pages 121–137, 1994.
19. R. Wallace. Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 188–195, 1996.