Bounded backtracking for the valued constraint satisfaction problems

Cyril Terrioux and Philippe Jégou

LSIS - Université d'Aix-Marseille 3
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{cyril.terrioux, philippe.jegou}@univ.u-3mrs.fr

Abstract. We propose a new method for solving Valued Constraint Satisfaction Problems based both on backtracking techniques - branch and bound - and the notion of tree-decomposition of valued constraint networks. This mixed method aims to benefit from the practical efficiency of enumerative algorithms while providing a warranty of a bounded time complexity. Indeed the time complexity of our method is $O(d^{w^+ + 1})$ with w^+ an approximation of the tree-width of the constraint network and d the maximum size of domains.
Such a complexity is obtained by exploiting optimal bounds on the sub-problems defined from the tree-decomposition. These bounds associated to some partial assignments are called "*structural valued goods*". Recording and exploiting these goods may allow our method to save some time and space with respect to ones required by classical dynamic programming methods. Finally, this method is a natural extension of the BTD algorithm [1] proposed in the classical CSP framework.

1 Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. In particular, many academic or real problems can be formulated in this framework which allows the expression of NP-complete problems. However, in this formalism, we can't express some notions like possibility or preference because the constraints are either satisfied or violated. In other words, there is no graduation in violation. To avoid this drawback, many extensions of the CSP formalism have been proposed (for instance [2–4]). In this paper, we focus on the valued CSP formalism (VCSP [4]) which allows the violations of some constraints by associating a cost (called a *valuation*) to each violated constraint. Solving the problem then consists in finding a complete assignment which optimizes a given criterion about the cost of constraint violations. Generally, we are interested by finding a complete assignment which minimizes the cost of all the violations. So, thanks to the VCSP framework, we can express optimization problems.

The basic method for solving VCSP is the Branch and Bound algorithm. Many improvements have been proposed from the CSP framework [4–8]. On

the other hand, some methods based on dynamic programming ([9, 10]) often provide good results on such problems.

In this article, we propose a new enumerative method for solving VCSPs. This method, called BTD$_{val}$, is a natural generalization of the BTD method [1] defined in the classical CSP framework. Such a generalization requires the extension of the theoretical frame used for BTD and the classical CSPs. Nevertheless, like BTD, BTD$_{val}$ relies on backtracking techniques (branch and bound) and the notion of tree-decomposition of valued constraint graphs. Such an hybrid method aims to benefit from the advantage of the two approaches, namely the practical efficiency of enumerative algorithms and the time complexity bounds of structural decomposition methods. Thanks to the tree-decomposition notion, BTD$_{val}$ divides the initial problem into several subproblems. Then, it solves each subproblem and records the optimal valuation of each subproblem. These optimal valuations associated with some assignments are called *structural valued goods*. Structural valued goods are then exploited in order to solve each subproblem only once, what allows BTD$_{val}$ to provide time complexity bounds better than ones of classical enumerative methods. Indeed, the time complexity of BTD$_{val}$ is $O(ns^2m\log(d).d^{w^+ +1})$ while the space complexity is $O(nsd^s)$ with $w^+ + 1$ an approximation of the tree-width of the constraint graph, s the size of the biggest minimal separator, n the number of variables and d the size of the largest domain. These bounds only depend on the used tree-decomposition (i.e. on some structural parameters). In [1], experimental results show that on classical CSPs, BTD clearly outperforms an approach founded on dynamic programming like Tree-Clustering [11, 12]. So, for VCSPs, we can hope that this behaviour will be confirmed in practice.

The paper is organized as follows. Section 2 introduces the main definitions about the VCSP formalism. Section 3 is devoted to the tree-decomposition notion. Then, section 4 describes the method we propose and present some theoretical results. Finally, in section 5, we discuss about some related works, before concluding in section 6.

2 Valued CSPs

A *constraint satisfaction problem* (CSP) is defined by a quadruplet (X, D, C, R). X is a set $\{x_1, \ldots, x_n\}$ of n variables. Each variable x_i takes its values in the finite domain d_{x_i} from D. Variables are subject to constraints from C. Each constraint c is defined as a set $\{x_{c_1}, \ldots, x_{c_k}\}$ of variables. A relation r_c (from R) is associated with each constraint c such that r_c represents the set of allowed tuples over $d_{x_{c_1}} \times \cdots \times d_{x_{c_k}}$. Given $Y \subseteq X$ such that $Y = \{x_1, \ldots, x_k\}$, an *assignment* of variables from Y is a tuple $\mathcal{A} = (v_1, \ldots, v_k)$ from $d_{x_1} \times \cdots \times d_{x_k}$. A constraint c is said *satisfied* by $\mathcal{A}$ if $c \subseteq Y, (v_1, \ldots, v_k)[c] \in r_c$, *violated* otherwise. We note the assignment $(v_1, \ldots, v_k)$ in the more meaningful form $(x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k)$.

Definition 1 ([4]) *A **valuation structure** is a 5-tuple $(E, \preceq, \oplus, \bot, \top)$ with E a set of valuations which is totally ordered by $\preceq$ with a minimum element*

noted $\bot$ and a maximum element noted $\top$. $\oplus$ is a monotonous, commutative, associative closed binary operation on E such that $\bot$ is an identity element and $\top$ an absorbing element.

The elements of E express different levels of violation. $\bot$ characterizes the satisfaction of a constraint and $\top$ an unacceptable violation. $\oplus$ allows to combine (aggregate) several valuations. Note that, in some cases, it can have some additional properties like idempotency or strict monotonicity. Thanks to the valuation structure, one can formally define the notion of valued CSP [4]:

Definition 2 *A **valued CSP** (VCSP) $\mathcal{P} = (X, D, C, R, S, \phi)$ consists of a classical CSP (X, D, C, R) with a valuation structure $S = (E, \preceq, \oplus, \bot, \top)$ and an application ϕ from C to E which associates a valuation to each constraint of C. A VCSP is called **binary** if each constraint of C involves at most two variables.*

The valuation of an assignment $\mathcal{A}$ on X is obtained by aggregating the valuations of the constraints violated by $\mathcal{A}$:

Definition 3 *Let $\mathcal{P}$ be a VCSP and $\mathcal{A}$ an assignment on X. The **valuation** of $\mathcal{A}$ with respect to $\mathcal{P}$ is defined by $\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{c \in C | \mathcal{A} \text{ violates } c} \phi(c)$.*

Given an instance $\mathcal{P}$, the VCSP problem consists in finding an assignment on X with a minimum valuation according to $\preceq$. This optimal valuation is called the *VCSP valuation* and is denoted $\alpha^*_{\mathcal{P}}$. Determining the valuation of a VCSP is an NP-hard problem. For instance, let us consider the VCSP whose constraint graph is presented in figure 1(a). We suppose that each domain d_x is equal to $\{1, 2, 3\}$ and each constraint c_{xy} means "$x < y$" if the letter represented by x precedes one represented by y in the alphabetical order (for example c_{AB} represents the constraint $A < B$). We exploit the valuation structure $S = (\overline{\mathbb{N}}, <, +, 0, +\infty)$. For each constraint c, the associated valuation is 1. For this VCSP, we obtain $\alpha^*_{\mathcal{P}} = 2$. $(A \leftarrow 1, B \leftarrow 1, C \leftarrow 2, D \leftarrow 2, E \leftarrow 3, F \leftarrow 2, G \leftarrow 2, H \leftarrow 3, I \leftarrow 3, J \leftarrow 3)$ is the best assignment. It violates the constraints c_{AB} and c_{CF}. The assignment valuation notion can be extended to partial assignments:

Definition 4 *Let $\mathcal{P}$ be a VCSP and $\mathcal{A}$ an assignment on $Y \subset X$. The **local valuation** of $\mathcal{A}$ with respect to $\mathcal{P}$ is defined by $v_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\substack{c \in C | c \subseteq Y \text{ and} \\ \mathcal{A} \text{ violates } c}} \phi(c)$.*

The following property establishes the link between the valuation of a complete assignment and the local valuation:

Property 1 *Let $\mathcal{P}$ be a VCSP, $\mathcal{A}$ an assignment on X and $\mathcal{B} \subseteq \mathcal{A}$. $v_{\mathcal{P}}(\mathcal{B}) \preceq v_{\mathcal{P}}(\mathcal{A}) = \mathcal{V}_{\mathcal{P}}(\mathcal{A})$.*

So the local valuation can provide a lower bound of the global valuation. The main interest of the local valuation consists in its computation which can be achieved incrementally.

The basic method for solving VCSPs is the branch and bound algorithm (noted BB). This enumerative method exploits the local valuation of the current assignment as a lower bound and the valuation of the best known solution as a upper bound. If the lower bound doesn't exceed the upper one, it extends the current assignment by assigning a new variable. Otherwise, it backtracks to the last assigned variable and then it tries to assign a new value to this variable. If all the values have been tried, it backtracks again. Many improved methods have been proposed from the classical CSP framework like valued Forward-Checking (noted vFC [4]), Nogood Recording [5], ... The use of the arc-consistency notion has been studied too ([6–8]). On the other hand, some methods based on dynamic programming, like the Russian Dolls Search (noted RDS) or the structural method proposed by Koster [10], often provide good results. These methods divide the problem into different subproblems and solve the initial problem by exploiting some informations recorded during the resolution of each subproblem.

3 Tree-decomposition

The only guarantees which can exist in terms of theoretical complexity before solving a problem are offered by structural decomposition methods. These methods proceed by isolating the parts intrinsically exponential (i.e. intractable in polynomial theoretical time) to induce a second step which guarantees a polynomial time of resolution. These methods generally exploit topological properties of the constraint graph and are based on the notion of tree-decomposition of graphs as defined below by Robertson and Seymour [13].

Definition 5 ([13]) *Let $G = (X, E)$ be a graph. A **tree-decomposition** of G is a pair $(\mathcal{C}, \mathcal{T})$ with $\mathcal{T} = (I, F)$ a tree and $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ a family of subsets of X, such that each cluster $\mathcal{C}_i$ is a node of $\mathcal{T}$ and verifies:*

1. *$\cup_{i \in I} \mathcal{C}_i = X$,*
2. *for all edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subseteq \mathcal{C}_i$, and*
3. *for all $i, j, k \in I$, if k is in a path from i to j in $\mathcal{T}$, then $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$*

The width of a tree-decomposition $(\mathcal{C}, \mathcal{T})$ is equal to $max_{i \in I}|\mathcal{C}_i| - 1$. The tree-width of G is the minimal width over all the tree-decompositions of G.

For the reader who isn't familiar with these notions, note that the above definition refers to a tree $\mathcal{T} = (I, F)$ where F is a set of edges which is required to satisfy the part (3) of this definition.

Even if finding an optimal tree-decomposition is an NP-Hard problem [14], many works have been developed in this direction [15], which often exploit equivalent definitions of this notion, including one based on an algorithmic approach related to *triangulated* graphs. The link between triangulated graphs and tree-decomposition is obvious. Indeed, given a triangulated graph, the set of maximal cliques $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ of (X, E) corresponds to the family of subsets associated with a tree-decomposition. As any graph $G = (X, E)$ is not necessarily

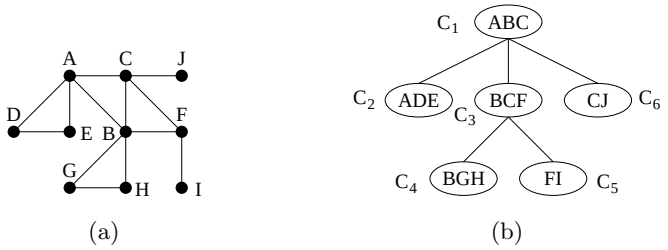

Fig. 1. (a) A constraint graph on 10 variables. (b) A tree-decomposition of this constraint graph.

triangulated, a tree-decomposition can be approximated by a triangulation of G which computes a triangulated graph G'. The width of G' is equal to the maximal size of cliques minus one in the resulting graph G'. The tree-width of G is then equal to the minimal width over all triangulations.

The graph in figure 1(a) is already triangulated. The maximum size of cliques is three and the tree-width of this graph is two. In figure 1(b), a tree whose nodes correspond to maximal cliques of the triangulated graph is a possible tree-decomposition for the graph of figure 1(a). So, we get $\mathcal{C}_1 = \{A, B, C\}$, $\mathcal{C}_2 = \{A, D, E\}$, $\mathcal{C}_3 = \{B, C, F\}$, $\mathcal{C}_4 = \{B, G, H\}$, $\mathcal{C}_5 = \{F, I\}$ and $\mathcal{C}_6 = \{C, J\}$.

The notion of tree-decomposition is exploited in the classical CSPs framework by many structural decomposition methods (see [16] for a survey about such methods and a theoretical comparison). These methods have the advantage of providing the best known bounds for the theoretical time complexity. For instance, the CSP decomposition method called *Tree-Clustering* [11, 12] is generally presented using an approximation of an optimal triangulation. It has a time complexity in $O(m.d^{w^+ + 1})$ with $w^+ + 1$ the size of the biggest cluster ($w^+ + 1 \leq n$). However, the space complexity is in $O(n.s.d^s)$ with s the maximal size of minimal separators (i.e. the size $s \leq w^+$ of the biggest intersection between two clusters). Finally, note that for every decomposition which induces a value w^+, we have $w \leq w^+$ with w the tree-width of the initial constraint graph.

The BTD method [1] solves classical CSPs by using the tree-decomposition notion jointly with backtracking techniques. Then, it benefits from a practical efficiency (thanks to enumerative techniques) while providing time complexity bounds equivalent to ones of structural decomposition methods (thanks to the tree-decomposition notion). Its time and space complexities are then similar to Tree-Clustering's ones. However, in practice, BTD obtains better results than Tree-Clustering while performing either as good as classical enumerative methods or better.

In the VCSP framework, the dynamic programming approach proposed by Koster [10] also exploits a tree-decomposition. It has a time complexity in $O(nd^{3(w^+ + 1)})$ and a space complexity in $O(d^{w^+ + 1})$. In the both frameworks, the required space can make the structural methods unusable in practice.

In the next section, we present an enumerative method for solving VCSPs which, by exploiting a tree-decomposition, provides complexity bounds similar to ones given above.

4 The BTD$_{val}$ algorithm for solving VCSPs

4.1 Presentation

Like BTD, BTD$_{val}$ (for Backtracking with Tree-Decomposition) proceeds by an enumerative search guided by a static pre-established partial order induced by a tree-decomposition of the constraint network. So, the first step of BTD$_{val}$ consists in computing a tree-decomposition or an approximation of a tree-decomposition. The obtained partial order allows to exploit some structural properties of the graph, during the search, in order to prune some branches of the search tree. Hence, BTD$_{val}$ differs from other techniques in the following points:

- the variable assignment order is induced by a tree-decomposition of the constraint graph,
- some subproblems won't be visited again if it we have computed yet their optimal valuation (notion of *structural valued good*).

Although our method is called BTD$_{val}$ for Backtracking with Tree-Decomposition, we will see later that the enumerative search can be based on BB or vFC.

4.2 Theoretical foundations

In the following, let us consider an instance $\mathcal{P} = (X, D, C, R, S, \phi)$ and a tree-decomposition $(\mathcal{C}, \mathcal{T})$ (or an approximation) of the constraint graph (X, C). We assume that the elements of $\mathcal{C} = \{\mathcal{C}_i : i \in I\}$ are indexed with respect to the notion of *compatible numbering*:

Definition 6 *A numbering on $\mathcal{C}$ compatible with a prefix numbering of $\mathcal{T} = (I, F)$ with $\mathcal{C}_1$ the root is called* ***compatible numbering*** *$N_{\mathcal{C}}$.*

Remark that in the previous definition, $\mathcal{T} = (I, F)$ is a tree (according to definition 5) with I the set of indices and F the set of edges. For example, figure 1(b) presents a compatible numbering on $\mathcal{C}$. We note $Desc(\mathcal{C}_j)$ the set of variables belonging to the union of the descendants $\mathcal{C}_k$ of $\mathcal{C}_j$ in the tree rooted in $\mathcal{C}_j$, $\mathcal{C}_j$ included. For instance, $Desc(\mathcal{C}_3) = \mathcal{C}_3 \cup \mathcal{C}_4 \cup \mathcal{C}_5 = \{B, C, F, G, H, I\}$. Note that the numbering $N_{\mathcal{C}}$ defines a partial variable ordering that permits to get an enumeration order on the variables of $\mathcal{P}$:

Definition 7 *A* ***compatible enumeration order*** *is an order $\preceq_X$ on the variables of X such that $\forall x, y \in X, x \preceq_X y$ if $\exists \mathcal{C}_i \ni x, \forall \mathcal{C}_j \ni y, i \leq j$.*

For example, the alphabetical order $A, B, \ldots, I, J$ is a compatible enumeration order. The tree-decomposition with the numbering N_C permits to partition the constraint set.

Definition 8 *Let $\mathcal{C}_i$ be a cluster. The* ***set*** $E_{\mathcal{P},\mathcal{C}_\mathrm{i}}$ ***of proper constraints*** *of cluster $\mathcal{C}_i$ is defined by $E_{\mathcal{P},\mathcal{C}_i} = \{c \in C | c \subseteq \mathcal{C}_i$ and $c \not\subseteq \mathcal{C}_{p(i)}\}$ with $\mathcal{C}_{p(i)}$ the parent cluster of $\mathcal{C}_i$.*

The set $E_{\mathcal{P},\mathcal{C}_i}$ contains each constraint $c_{xy} = \{x, y\}$ with x and y two variables of $\mathcal{C}_i$ such that x and y don't belong both to $\mathcal{C}_{p(i)}$ the parent cluster of $\mathcal{C}_i$. For instance, if we consider the problem described in figure 1, we obtain $E_{\mathcal{P},\mathcal{C}_1} = \{c_{AB}, c_{AC}, c_{BC}\}$, $E_{\mathcal{P},\mathcal{C}_2} = \{c_{AD}, c_{AE}, c_{DE}\}$, $E_{\mathcal{P},\mathcal{C}_3} = \{c_{BF}, c_{CF}\}$, $E_{\mathcal{P},\mathcal{C}_4} = \{c_{BG}, c_{BH}, c_{GH}\}$, $E_{\mathcal{P},\mathcal{C}_5} = \{c_{FI}\}$ and $E_{\mathcal{P},\mathcal{C}_6} = \{c_{CJ}\}$.

Property 2 *The sets $(E_{\mathcal{P},\mathcal{C}_i})_i$ form a partition of C.*

Proof: First, we are going to show that $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} = C$.
As $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} \subset C$ is obvious, we have to prove $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} \supset C$.
Let $c \in C$. According to definition 5, there exists at less a cluster $\mathcal{C}_i$ such that $c \subseteq \mathcal{C}_i$. In particular, we necessarily have $c \subseteq \mathcal{C}_k$ where $k = \min\{i | c \subseteq \mathcal{C}_i\}$ and $c \not\subseteq \mathcal{C}_{p(k)}$. Therefore, $c \in E_{\mathcal{P},\mathcal{C}_k}$. So we obtain $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} \supset C$ and then $\bigcup_{\mathcal{C}_i \subseteq X} E_{\mathcal{P},\mathcal{C}_i} = C$

Now we have to prove that $\forall \mathcal{C}_i, \mathcal{C}_j,\ E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j} = \emptyset$.
Assume that there exists two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ such that $E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j} \neq \emptyset$. Let $c \in E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j}$. According to definition 8, we have $c \subseteq \mathcal{C}_i \cap \mathcal{C}_j$.
Then, according to definition 5, there exists a path between $\mathcal{C}_i$ and $\mathcal{C}_j$ such that if $\mathcal{C}_k$ belongs to this path, $\mathcal{C}_i \cap \mathcal{C}_j \subseteq \mathcal{C}_k$. The parent cluster of $\mathcal{C}_i$ or $\mathcal{C}_j$'s one clearly belongs to this path. Therefore $c \subseteq \mathcal{C}_{p(i)}$ or $c \subseteq \mathcal{C}_{p(j)}$. So we obtain a contradiction since $c \in E_{\mathcal{P},\mathcal{C}_i}$ and $c \in E_{\mathcal{P},\mathcal{C}_j}$. So $\forall i, j,\ E_{\mathcal{P},\mathcal{C}_i} \cap E_{\mathcal{P},\mathcal{C}_j} = \emptyset$

Hence, the sets $(E_{\mathcal{P},\mathcal{C}_i})_i$ form a partition of C. □

Note that this property becomes fundamental when $\oplus$ isn't idempotent. Indeed, in such a case, we must be careful not to take into account a constraint several times. Exploiting the sets $E_{\mathcal{P},\mathcal{C}_i}$ prevents such a problem from occurring and so ensures that BTD_{val} safely computes the valuation of assignments. Then, we can define the notion of induced VCSP:

Definition 9 *Let $\mathcal{C}_i$ and $\mathcal{C}_j$ be two clusters with $\mathcal{C}_j$ a son of $\mathcal{C}_i$. Let $\mathcal{A}$ be an assignment on $\mathcal{C}_i \cap \mathcal{C}_j$. $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j} = (X_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, D_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, C_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, R_{\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}}, S, \phi)$ is the* ***VCSP induced by*** *$\mathcal{A}$ on the descent of $\mathcal{C}_i$ rooted in $\mathcal{C}_j$ (i.e. on $\mathcal{C}_j$ and its descendants) with:*

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

The induced VCSP $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$ corresponds to the VCSP $\mathcal{P}$ restricted to the subproblem rooted in $\mathcal{C}_j$ such that the domain of each variable x in $\mathcal{C}_i \cap \mathcal{C}_j$ is reduced to the value assigned to x in $\mathcal{A}$. That is, we consider the subproblem whose variables are ones of $\mathcal{C}_j$ and its descendants. As for the constraint set of $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$, it only contains the constraints which exclusively appear in $\mathcal{C}_j$ and its descendants. For instance, given the assignment $\mathcal{A} = (B \leftarrow 2, C \leftarrow 2)$ on $\mathcal{C}_1 \cap \mathcal{C}_3$, let us consider $\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}$ the VCSP induced by $\mathcal{A}$ on the descent of $\mathcal{C}_1$ rooted in $\mathcal{C}_3$. We have $X_{\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}} = \{B, C, F, G, H, I\}$, $d_B = d_C = \{2\}$, $d_F = d_G = d_H = d_I = \{1, 2, 3\}$ and $C_{\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}} = \{c_{BF}, c_{CF}, c_{BG}, c_{BH}, c_{GH}, c_{FI}\}$. Note that the constraint c_{BC} doesn't belong to the constraint set of $\mathcal{P}_{\mathcal{A},\mathcal{C}_1/\mathcal{C}_3}$ because it isn't a proper constraint of $\mathcal{C}_3$ ($c_{BC} \subseteq \mathcal{C}_1$ and $\mathcal{C}_1 = \mathcal{C}_{p(3)}$). Now, from the sets $E_{\mathcal{P},\mathcal{C}_i}$, we can introduce the notion of local valuation for a cluster:

Definition 10 *Given a cluster $\mathcal{C}_i$ and an assignment $\mathcal{A}$ on $Y \subset X$ with $Y \cap \mathcal{C}_i \neq \emptyset$. The **local valuation for the cluster** $\mathcal{C}_i$ of the assignment $\mathcal{A}$ with respect to $\mathcal{P}$ (noted $v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A})$) is the local valuation of $\mathcal{A}$ restricted to the constraints of $E_{\mathcal{P},\mathcal{C}_i}$, that is to say $v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) = \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \\ and\ \mathcal{A}\ violates\ c}} \phi(c)$*

In other words, the valuation local for a cluster $\mathcal{C}_i$ only takes into account the constraints proper to $\mathcal{C}_i$. Remark that the local valuation for a cluster can be computed incrementally. This valuation presents many interesting properties. First, its computation only depends on the variables of the considered cluster.

Property 3 *Let $\mathcal{C}_i$ be a cluster and $\mathcal{A}$ an assignment on $Y \subseteq X$ such that $\mathcal{C}_i \subseteq Y$. $v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}[\mathcal{C}_i])$*

Proof: $v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) = \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \\ and\ \mathcal{A}\ violates\ c}} \phi(c) = \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \cap \mathcal{C}_i \\ and\ \mathcal{A}\ violates\ c}} \phi(c) = \bigoplus_{\substack{c \in E_{\mathcal{P},\mathcal{C}_i} | c \subseteq Y \cap \mathcal{C}_i \\ and\ \mathcal{A}[\mathcal{C}_i]\ violates\ c}} \phi(c) = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}[\mathcal{C}_i])$ □

Then, the aggregation of local valuations for a cluster allows us to compute the valuation of a complete assignment.

Property 4 *Let $\mathcal{A}$ be an assignment on X.*

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

Proof: Since the sets $(E_{\mathcal{P},\mathcal{C}_i})_i$ form a partition of C (property 2), each constraint of C is taken into account only once. So, $\mathcal{V}_{\mathcal{P}}(\mathcal{A}) = \bigoplus_{\mathcal{C}_i \subseteq X} v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A})$. □

It follows from these two properties that we can compute the valuation of a complete assignment $\mathcal{A}$ by exploiting only the local valuation for each cluster $\mathcal{C}_i$ of the assignment $\mathcal{A}[\mathcal{C}_i]$. Finally, the next property ensures that the local valuation for a cluster $\mathcal{C}_j$ of an assignment $\mathcal{B}$ with respect to $\mathcal{P}$ is preserved if we considered an induced subproblem which contains $\mathcal{C}_j$.

Property 5 *Let $\mathcal{C}_i$ and $\mathcal{C}_j$ two clusters with $\mathcal{C}_j$ a descendant of $\mathcal{C}_i$. Let $\mathcal{A}$ be an assignment on $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$ and $\mathcal{P}' = \mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$. If $\mathcal{B}$ is an assignment on $\mathcal{C}_j$ such that $\mathcal{B}[\mathcal{C}_j \cap \mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}[\mathcal{C}_j \cap \mathcal{C}_i \cap \mathcal{C}_{p(i)}]$, $v_{\mathcal{P},\mathcal{C}_j}(\mathcal{B}) = v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B})$.*

Proof: as $E_{\mathcal{P},\mathcal{C}_j} = E_{\mathcal{P}',\mathcal{C}_j}$ $v_{\mathcal{P},\mathcal{C}_j}(\mathcal{B}) = v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B})$. □

Now, we are able to define the notion of structural valued good.

Definition 11 *Let $\mathcal{C}_i$ and $\mathcal{C}_j$ two clusters with $\mathcal{C}_j$ a son of $\mathcal{C}_i$. A **structural valued good** of $\mathcal{C}_i$ with respect to $\mathcal{C}_j$ is a pair $(\mathcal{A}, v)$ with $\mathcal{A}$ an assignment on $\mathcal{C}_i \cap \mathcal{C}_j$ and v the optimal valuation of the VCSP $\mathcal{P}_{\mathcal{A},\mathcal{C}_i/\mathcal{C}_j}$.*

For instance, if we consider the assignment $\mathcal{A} = (B \leftarrow 2, C \leftarrow 2)$ on $\mathcal{C}_1 \cap \mathcal{C}_3$, we obtain the good $(\mathcal{A}, 2)$ since the best assignment on $Desc(\mathcal{C}_3)$ is $(B \leftarrow 2, C \leftarrow 2, F \leftarrow 3, G \leftarrow 3, H \leftarrow 3, I \leftarrow 3)$. Note that this assignment violates the constraints c_{BC}, c_{GH} and c_{FI}, but c_{BC} is discarded (since $c_{BC} \notin E_{\mathcal{P},\mathcal{C}_3}$).

Given an assignment $\mathcal{A}$ on $\mathcal{C}_i$, the following theorem expresses that we can compute the valuation of the best assignment $\mathcal{B}$ on $Desc(\mathcal{C}_i)$ with $\mathcal{B}[\mathcal{C}_i] = \mathcal{A}$ by exploiting the optimal valuation of each subproblem rooted in a son $\mathcal{C}_f$ of $\mathcal{C}_i$ and induced by $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_f]$. Note that the optimal valuation of each subproblem is provided either by solving the considered subproblem or by exploiting a structural valued good. Finally, remark that this optimal valuation can be computed independently of ones of other subproblems.

Theorem 1 *Let $\mathcal{C}_i$ be a cluster, $\mathcal{A}$ an assignment on $\mathcal{C}_i$ and $\mathcal{P}' = \mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}],\mathcal{C}_{p(i)}/\mathcal{C}_i}$.*

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

The proof of this theorem requires the following lemma:

Lemma 1 *Let $\mathcal{C}_i$ be a cluster and $\mathcal{A}$ an assignment on $\mathcal{C}_i$. Let $\mathcal{P}' = \mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}],\mathcal{C}_{p(i)}/\mathcal{C}_i}$.*

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

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

We have $\lambda = \lambda'$.

Proof (lemma 1):

For each $\mathcal{C}_j$ son of $\mathcal{C}_i$, we note $\lambda_{\mathcal{C}_j} = \min_{\substack{\mathcal{B}|X_{\mathcal{B}} = Desc(\mathcal{C}_j) \cup \mathcal{C}_i \\ and\ \ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right]$. We then have $\lambda' = \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \lambda_{\mathcal{C}_j}$.

For each $\mathcal{C}_j$ son of $\mathcal{C}_i$, there exists an assignment $\mathcal{B}_{\mathcal{C}_j}$ on $Desc(\mathcal{C}_j) \cup \mathcal{C}_i$ such that $\mathcal{B}_{\mathcal{C}_j}[\mathcal{C}_i] = \mathcal{A}$ and $\lambda_{\mathcal{C}_j} = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_{\mathcal{C}_j})$. Likewise, there is an assignment

$\mathcal{B}_\lambda$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{B}_\lambda[\mathcal{C}_i] = \mathcal{A}$ and $\lambda = \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda) \right]$.

We want to prove that for each son $\mathcal{C}_j$ of $\mathcal{C}_i$, we have $\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) = \lambda_{\mathcal{C}_j}$.
Assume there exists a son $\mathcal{C}_s$ of $\mathcal{C}_i$ such that $\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i]) \neq \lambda_{\mathcal{C}_s}$.
By definition of $\lambda_{\mathcal{C}_s}$, $\lambda_{\mathcal{C}_s} \prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i]) = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda)$

Let $\mathcal{B}'$ be an assignment on $Desc(\mathcal{C}_i)$ such that $\mathcal{B}'[\mathcal{C}_i] = \mathcal{A}$ and $\forall \mathcal{C}_j \in Sons(\mathcal{C}_i)$, $\mathcal{B}'[Desc(\mathcal{C}_j) \cup \mathcal{C}_i] = \mathcal{B}_{\mathcal{C}_j}$. Such an assignment exists since $\forall \mathcal{C}_j, \mathcal{C}_{j'} \in Sons(\mathcal{C}_i)$, $Desc(\mathcal{C}_j) \cap Desc(\mathcal{C}_{j'}) \subseteq \mathcal{C}_i$.
Furthermore, we have $\lambda_{\mathcal{C}_s} = \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}'[Desc(\mathcal{C}_s) \cup \mathcal{C}_i])$.

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

$$\begin{aligned}
\lambda' &= \lambda_{\mathcal{C}_s} \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \lambda_{\mathcal{C}_j} \\
&\prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_s) \cup \mathcal{C}_i]) \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) \right] \\
&\prec \bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_s)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda) \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i) \setminus \{\mathcal{C}_s\}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda) \right] = \lambda
\end{aligned}$$

Hence, we obtain a contradiction with the definition of λ. So, for each son $\mathcal{C}_j$ of $\mathcal{C}_i$, $\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}_\lambda[Desc(\mathcal{C}_j) \cup \mathcal{C}_i]) = \lambda_{\mathcal{C}_j}$. It ensues that $\lambda = \lambda'$. □

Proof (theorem 1):
We note $M = \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} v_{\mathcal{P}'}(\mathcal{B})$.

$$\begin{aligned}
M &\underset{property\ 1}{=} \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \mathcal{V}_{\mathcal{P}'}(\mathcal{B}). \\
&\underset{property\ 4}{=} \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left(\bigoplus_{\mathcal{C}_j \subseteq Desc(\mathcal{C}_i)} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right) \\
&= \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left(v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{B}) \oplus \bigoplus_{\substack{\mathcal{C}_j | j \neq i, \\ \mathcal{C}_j \subseteq Desc(\mathcal{C}_i)}} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right) \\
&\underset{property\ 3}{=} \min_{\substack{\mathcal{B} | X_{\mathcal{B}} = Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i] = \mathcal{A}}} \left(v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{B}[\mathcal{C}_i]) \oplus \bigoplus_{\substack{\mathcal{C}_j | j \neq i, \\ \mathcal{C}_j \subseteq Desc(\mathcal{C}_i)}} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right)
\end{aligned}$$

For every assignment $\mathcal{B}$ such that $X_{\mathcal{B}} = Desc(\mathcal{C}_i)$ and $\mathcal{B}[\mathcal{C}_i] = \mathcal{A}$, we have $v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{B}[\mathcal{C}_i]) = v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A})$. As $v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A})$ is a constant, we have:

$$
\begin{aligned}
M &= v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \min_{\substack{\mathcal{B}[X_{\mathcal{B}}=Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left(\bigoplus_{\substack{\mathcal{C}_j | j\neq i, \\ \mathcal{C}_j \subseteq Desc(\mathcal{C}_i)}} v_{\mathcal{P}',\mathcal{C}_j}(\mathcal{B}) \right) \\
&= v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \min_{\substack{\mathcal{B}[X_{\mathcal{B}}=Desc(\mathcal{C}_i) \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left(\bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right) \\
&\underset{lemma\ 1}{=} v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}[X_{\mathcal{B}}=Desc(\mathcal{C}_j)\cup\mathcal{C}_i \\ and\ \mathcal{B}[\mathcal{C}_i]=\mathcal{A}}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right) \\
M &= v_{\mathcal{P}',\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}[X_{\mathcal{B}}=Desc(\mathcal{C}_j)\ and \\ \mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_j]=\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j]}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P}',\mathcal{C}_k}(\mathcal{B}) \right] \right) \\
&\underset{property\ 5}{=} v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}[X_{\mathcal{B}}=Desc(\mathcal{C}_j)\ and \\ \mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_j]=\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j]}} \left[\bigoplus_{\mathcal{C}_k \subseteq Desc(\mathcal{C}_j)} v_{\mathcal{P},\mathcal{C}_k}(\mathcal{B}) \right] \right) \\
&\underset{property\ 4}{=} v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j \in Sons(\mathcal{C}_i)} \left(\min_{\substack{\mathcal{B}[X_{\mathcal{B}}=Desc(\mathcal{C}_j) \\ and\ \mathcal{B}[\mathcal{C}_i\cap\mathcal{C}_j]=\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j]}} \mathcal{V}_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j],\mathcal{C}_i/\mathcal{C}_j}}(\mathcal{B}) \right) \\
&= v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \oplus \bigoplus_{\mathcal{C}_j\ son\ of\ \mathcal{C}_i} \alpha^*_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i\cap\mathcal{C}_j],\mathcal{C}_i/\mathcal{C}_j}} \ \square
\end{aligned}
$$

From theorem 1, we deduce the following corollary. This corollary establishes the link between the optimal valuation of a subproblem rooted in $\mathcal{C}_i$ and the optimal valuation of each subproblem rooted in a son $\mathcal{C}_j$ of $\mathcal{C}_i$.

Corollary 1 *Let $\mathcal{C}_i$ be a cluster and $\mathcal{A}$ an assignment on $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$.*

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

4.3 The BTD$_{val}$ algorithm

The BTD$_{val}$ method is based on the BB algorithm (note that we can also base it on vFC). It explores the search space by exploiting a compatible order, which begins with the variables of the root cluster $\mathcal{C}_1$. Inside a cluster $\mathcal{C}_i$, it proceeds classically like BB by assigning a value to a variable, by maintaining and comparing upper and lower bounds and by backtracking if a lower bound is greater than (or equal to) the corresponding upper bound. However, unlike BB, BTD$_{val}$ uses two kinds of bounds: local bounds and global ones. The local bounds only take into account the subproblem rooted in $\mathcal{C}_i$ (namely the induced VCSP $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$ with $\mathcal{A}$ the current assignment on $\mathcal{C}_i \cap \mathcal{C}_{p(i)}$). The local lower bound corresponds to the valuation of the current assignment on $Desc(\mathcal{C}_i)$, that is to say, the local valuation of the current assignment with respect to $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$. The local upper bound is then defined by the valuation of the best known assignment $\mathcal{B}$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{A}$. In other words, it's the valuation of the best known solution for $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$. The global bounds are similar to BB's ones,

that is to say the local valuation of the current assignment for the lower bound and the valuation of the best known solution for the upper one.

When every variable in $\mathcal{C}_i$ is assigned, if each lower bound is less than the corresponding upper bound, BTD$_{val}$ keeps on the search with the first son of $\mathcal{C}_i$ (if there is one). More generally, let us consider a son $\mathcal{C}_j$ of $\mathcal{C}_i$. Given the current assignment $\mathcal{A}$ on $\mathcal{C}_i$, BTD$_{val}$ checks whether the assignment $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$ corresponds to a valued structural good:

- if so, BTD$_{val}$ aggregates the valuation associated to this valued good with each lower bound.
- else, it extends $\mathcal{A}$ on $Desc(\mathcal{C}_j)$ in order to compute the valuation v of the best assignment $\mathcal{B}$ such that $\mathcal{B}[\mathcal{C}_i \cap \mathcal{C}_j] = \mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j]$. Then, it aggregates v with each lower bound and it records the valued good $(\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_j], v)$.

If, after having proceeded the son $\mathcal{C}_j$, the two lower bounds don't exceed their respective upper bound, BTD$_{val}$ keeps on the search with the next son of $\mathcal{C}_i$. Remark that by exploiting the structural valued goods, BTD$_{val}$ doesn't solve again some subproblems. So the variables of these subproblems aren't assigned again. Hence we call such a phenomenon a *forward-jump* (by analogy with backjump). For instance, suppose that we use the alphabetical order as variable order and that, after assigning the variable F in $\mathcal{C}_3$, we exploit a good on $\mathcal{C}_3 \cap \mathcal{C}_4$. Then, we try to assign I without exploring again $Desc(\mathcal{C}_4)$. If every son has been proceeded and each lower bound doesn't exceed its corresponding upper bound, then a better solution for $\mathcal{P}_{\mathcal{A},\mathcal{C}_{p(i)}/\mathcal{C}_i}$ has been found. Finally, if a failure occurs, BTD$_{val}$ tries to modify the current assignment on $\mathcal{C}_i$.

In fact, due to the structural valued good definition, the global lower bound is defined by the valuation of the best extension of $\mathcal{A}$ on every cluster which precedes the current cluster in the used compatible enumeration. It's the same for the local lower bound, but we only consider the clusters belonging to the descent of the current cluster. Remark that we consider an extension of $\mathcal{A}$, and not $\mathcal{A}$, because $\mathcal{A}$ only contains the variables belonging to a cluster located on the path between the root cluster and the current cluster. Finally note that the global upper bound is the same as BB's one, unlike the global lower bound which is better than BB's one.

Figure 2 describes the BTD$_{val}$ algorithm. Given an assignment $\mathcal{A}$ and a cluster $\mathcal{C}_i$, BTD$_{val}$ looks for the best assignment $\mathcal{B}$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{A}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}] = \mathcal{B}[\mathcal{C}_i \backslash V_{\mathcal{C}_i}]$ and $v_{\mathcal{P}_{\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}], \mathcal{C}_{p(i)}/\mathcal{C}_i}}(\mathcal{B}) \prec \alpha_{\mathcal{C}_i}$, where:

- $V_{\mathcal{C}_i}$ is the set of unassigned variables in $\mathcal{C}_i$,
- $\alpha_{\mathcal{C}_1}$ is the valuation of the best known solution,
- l_{tot} is the valuation of the best extension $\mathcal{A}'$ of $\mathcal{A}$ on all the clusters which precede $\mathcal{C}_i$ according to the compatible numbering ($l_{tot} = v_{\mathcal{P}}(\mathcal{A}') \prec \alpha_{\mathcal{C}_1}$),
- $\alpha_{\mathcal{C}_i}$ is the valuation of the best known assignment $\mathcal{B}'$ on $Desc(\mathcal{C}_i)$ such that $\mathcal{A}[\mathcal{C}_i \cap \mathcal{C}_{p(i)}] = \mathcal{B}'[\mathcal{C}_i \cap \mathcal{C}_{p(i)}]$
- $l_{\mathcal{C}_i} = v_{\mathcal{P},\mathcal{C}_i}(\mathcal{A}) \prec \alpha_{\mathcal{C}_i}$.

If BTD$_{val}$ finds such an assignment, it returns its valuation, otherwise it returns a valuation greater than (or equal to) $\alpha_{\mathcal{C}_i}$. The initial call is BTD$_{val}(\emptyset, \mathcal{C}_1, \mathcal{C}_1, \bot, \top, \bot, \top)$.

Theorem 2 *BTD_{val} is sound, complete and terminates.*

Finally, we provide the time and space complexities of BTD_{val}. We suppose that a tree-decomposition (or an approximation) has been computed. Therefore the parameters used in the next theorem are related to this decomposition. BTD_{val} obtains complexities similar to Tree-Clustering's ones:

Theorem 3 *BTD_{val} has a time complexity in $O(n.s^2.m.\log(d).d^{w^+ +1})$ and a space complexity in $O(n.s.d^s)$ with $w^+ + 1$ the size of the biggest $\mathcal{C}_k$ and s the size of the biggest intersection $\mathcal{C}_i \cap \mathcal{C}_j$ where $\mathcal{C}_j$ is a son of $\mathcal{C}_i$.*

5 Related works

BTD_{val} is mostly based on tree-decomposition. So, works like Tree-Clustering and its improvements [11, 12] or the dynamic programming approach of Koster [10] are close to our approach. BTD_{val} can be considered as an hybrid approach

```
    BTD_val(A, C_i, V_{C_i}, l_tot, α_{C_1}, l_{C_i}, α_{C_i})
1.  If V_{C_i} = ∅
2.  Then
3.     If Sons(C_i) = ∅ Then Return l_{C_i}
4.     Else
5.        F ← Sons(C_i)
6.        α ← ⊥
7.        While F ≠ ∅ and α ⊕ l_tot ≺ α_{C_1} and α ⊕ l_{C_i} ≺ α_{C_i} Do
8.           Choose C_j in F
9.           F ← F\{C_j}
10.          If (A[C_j ∩ C_i], v) is a good of C_i/C_j in G Then α ← α ⊕ v
11.          Else
12.             v ← BTD_val(A, C_j, C_j\(C_j ∩ C_i), l_tot ⊕ α, α_{C_1}, ⊥, α_{C_i})
13.             α ← α ⊕ v
14.             Record the good (A[C_j ∩ C_i], v) of C_i/C_j in G
15.          EndIf
16.       EndWhile
17.       Return α ⊕ l_{C_i}
18.    EndIf
19. Else
20.    Choose x ∈ V_{C_i}
21.    d ← d_x
22.    While d ≠ ∅ and l_tot ≺ α_{C_1} and l_{C_i} ≺ α_{C_i} Do
23.       Choose a in d
24.       d ← d\{a}
25.       L ← {c = {x, y} ∈ E_{P,C_i} | y ∉ V_{C_i}}
26.       l_a ← ⊥
27.       While L ≠ ∅ and l_tot ⊕ l_a ≺ α_{C_1} and l_{C_i} ⊕ l_a ≺ α_{C_i} Do
28.          Choose c in L
29.          L ← L\{c}
30.          If c violates A ∪ {x ← a} Then l_a ← l_a ⊕ φ(c)
31.       EndWhile
32.       If l_tot ⊕ l_a ≺ α_{C_1} and l_{C_i} ⊕ l_a ≺ α_{C_i}
33.       Then α_{C_i} ← min(α_{C_i}, BTD_val(A ∪ {x ← a}, C_i, V_{C_i}\{x}, l_tot ⊕ l_a, α_{C_1}, l_{C_i} ⊕ l_a, α_{C_i}))
34.       EndIf
35.    EndWhile
36.    Return α_{C_i}
37. EndIf
```

Fig. 2. The BTD_{val} algorithm.

realizing a tradeoff between practical time and space complexity. In [12], Dechter and El Fattah present a time-space tradeoff scheme. This scheme allows them to propose a spectrum of algorithms such that tree-clustering and cycle-cutset conditioning (linear for space complexity) are two extremes in this spectrum. Another interesting idea in their work is the possibility to modify the size of separators to minimize space. This idea can also be exploited in BTD$_{val}$.

BTD$_{val}$ presents a better time complexity than the dynamic programming approach of Koster. Then, BTD$_{val}$ differs from this approach in computing a tree-decomposition (or an approximation of a tree-decomposition). BTD$_{val}$ exploits a triangulation of the constraint graph, while the dynamic programming approach uses a heuristic method and network flow techniques. Furthermore, Koster proposes several pretreatments. In particular, one of these pretreatments allows to reduce the size of the constraint graph, which may also reduce the time complexity. So adding such pretreatments may be useful for our approach.

BTD$_{val}$ is close to a method like the russian dolls search [9]. Indeed, in order to find the optimal valuation of a VCSP, BTD$_{val}$ solves many subproblems according a pre-established compatible order. The BTD$_{val}$'s clusters have a role similar to one of variables in RDS. Nevertheless, the two methods exploit differently the optimal valuations of subproblems. Like BTD$_{val}$, the method Tree-RDS [17] (a variant of RDS) takes advantage of the constraint graph in order to determine whether some problems are independent or not. However, if the independence of subproblems is used similarly, the Tree-RDS's subproblems differ conceptually from BTD$_{val}$'s ones. It's the same for the adaptation [18] of the algorithm Pseudo-Tree Search and its combination with a variant of RDS.

6 Conclusion

In this paper, we have defined a new method (called BTD$_{val}$) for solving valued CSPs. This method can actually be based on BB or on vFC. Thanks to the notion of structural valued goods we have introduced, BTD$_{val}$ obtains complexity bounds similar to (or better than) the best known ones. Indeed, the time complexity of BTD$_{val}$ is $O(ns^2m\log(d).d^{w^+ +1})$ with $w^+ + 1$ the size of the biggest cluster while the space complexity is $O(nsd^s)$ with s the size of the biggest intersection between two clusters. Now, an experimental study is required to assess the practical interest of our approach.

Among the possible extensions of this work, we must base our algorithm on more efficient methods like the russian dolls search or algorithms which use directional arc-consistency [19–21]. Using such methods seems natural since BTD$_{val}$ exploits a compatible enumeration order.

References

1. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.

2. E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
3. S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proc. of the 14^{th} IJCAI*, pages 624–630, 1995.
4. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proc. of the 14^{th} IJCAI*, pages 631–637, 1995.
5. P. Dago and G. Verfaillie. Nogood Recording for Valued Constraint Satisfaction Problems. In *Proc. of ICTAI 96*, pages 132–139, 1996.
6. J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.
7. T. Schiex. Une comparaison des cohérences d'arc dans les Max-CSP. In *Actes des JNPC'2002*, pages 209–223, 2002. In french.
8. J. Larrosa. On arc and node consistency. In *Proc. of AAAI*, 2002.
9. G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proc. of the 14^{th} AAAI*, pages 181–187, 1996.
10. A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, November 1999.
11. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
12. R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.
13. N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of tree-width. *Algorithms*, 7:309–322, 1986.
14. S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embedding in a k-tree. *SIAM Journal of Discrete Mathematics*, 8:277–284, 1987.
15. A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125:3–17, 2001.
16. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
17. P. Meseguer and M. Sánchez. Tree-based Russian Doll Search. In *Proc. of Workshop on soft constraint*. CP'2000, 2000.
18. J. Larrosa, P. Meseguer, and M. Sánchez. Pseudo-Tree Search with Soft Constraints. In *Proc. of the 15^{th} ECAI*, pages 131–135, 2002.
19. R. Wallace. Directed arc consistency preprocessing. In *Proc. of the ECAI-94 Workshop on Constraint Processing, LNCS 923*, pages 121–137, 1994.
20. R. Wallace. Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem. In *Proc. of AAAI*, pages 188–195, 1996.
21. J. Larrosa and P. Meseguer. Exploiting the use of DAC in Max-CSP. In *Proc. of the 2^{nd} CP*, pages 308–322, 1996.