# A new Evaluation of Forward Checking and its Consequences on Efficiency of Tools for Decomposition of CSPs

Philippe Jégou Samba Ndojh Ndiaye
Cyril Terrioux
LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, samba-ndojh.ndiaye, cyril.terrioux}@univ-cezanne.fr

## Abstract

*In this paper, a new evaluation of the complexity of Forward Checking for solving non-binary CSPs with finite domains is proposed. Unlike what is done usually, it does not consider the size of domains, but the size of the relations associated to the constraints. It may lead sometimes to define better complexity bounds. By using this first result, we show that the tractability hierarchy proposed in [6] which compares different methods based on a decomposition of constraint networks can be seen from a new viewpoint.*

## 1 Introduction

It is well known that the CSP formalism and its generalizations to Valued CSPs offer interesting frameworks to express and solve various problems in numerous fields. A CSP can be considered as the problem of checking if a finite set $X$ of variables can be assigned in their domains of values given by $D$, while satisfying simultaneously a set $C$ of constraints. Such an assignment is a solution of the CSP. Then the problem is generally to find one solution. Unfortunately, checking the existence of a solution of a CSP is NP-complete. So, for solving CSPs, different classes of algorithms have been proposed, which combine backtracking and filtering. From a practical viewpoint, these algorithms can be frequently efficient. Precisely, it is the case when an adapted level for filtering is considered to help backtracking. Generally, this trade-off uses arc-consistency to filter the domain of unassigned variables during the search. The good level of filtering is generally situated between FC (Forward Checking) [7], which is the most restricted form of arc-consistency (one pass and limited form of filtering with arc-consistency) and MAC [13] which maintains arc-consistency on the whole resulting problem. While these algorithms can be really efficient from a practical viewpoint, their time complexity is $O(S.m^n)$ where $S$ is the size of the considered CSP, $n$ the number of variables and $m$ the maximum size of domains of variables. This evaluation is then clearly driven by the size of domains.

Different approaches have been proposed to improve these bounds, for example by exploiting structural properties that exist frequently in real life problems. The interest for the exploitation of structural properties was observed in numerous domains as in CSP [4], in constraint optimization (VCSPs) [14, 2], in SAT, in relational databases, or in Bayesian or probabilistic networks. Formally, complexity results based on topological features of problems have been proposed. Generally, they rely on the properties of a tree-decomposition [12] or a hypertree-decomposition [6] of the constraint network which formalizes the structure and consequently allows to express topological properties.

Given a tree-decomposition of width $w$, the time complexity of the best structural approaches is $O(S.m^{w+1})$, with the guarantee to have $w < n$, and in many cases, $w \ll n$. Given a hypertree-decomposition of width $h$, the time complexity is then $O(S.r^h)$, with $r$ the maximum size of relations (tables) associated to constraints. [6] has shown that hypertree-decomposition is better than tree-decomposition, since $h \leq w$.

The practical interest of such approaches has been proved in some recent works around (V)CSPs [9, 10, 11, 2]. These empirical observations seem to contradict the theoretical results, since they rely on tree-decomposition while, in our knowledge, no approach based on hypertree-decomposition has shown a practical interest yet. From a first analysis, we can think that it is due to the fact that the complexity bounds based on hypertree-decomposition are often reached to the detriment of the practical efficiency.

For example, they assume that relations associated to constraints are expressed by tables, what is sometimes unrealistic from a practical viewpoint. Moreover, to ensure complexity bounds as $O(S.r^h)$, the method performs joins of relations. For solving CSPs, such an approach is generally also unrealistic due to the size of the generated relations. On the other hand, the methods that have shown their feasibility and their practical interest are based on assignments of variables, exploiting the practical efficiency of backtracking-based algorithms (as FC or MAC), while they ensure the complexity bounds as $O(S.m^{w+1})$.

In this paper, we introduce a theoretical justification to these experimental observations. First, we investigate the complexity of algorithms as FC, with a new viewpoint. Indeed, we present here another evaluation of their complexity which is related to the size of relations associated to constraints: $O(S.r^k)$, where $k$ is a structural parameter of the CSP, independently of any decomposition method used. This result is then exploited to show that for structural methods using tree-decomposition, we can now express their complexity by $O(S.r^h)$. The result shows the relations between decomposition methods in a different light and introduces a new viewpoint for the constraint tractability hierarchy introduced in [6]. Note that the proof of the result presented in [6] remains true. Here, we modify its consequence because we change the basic hypothesis on the complexity of the basic algorithms used to solve clusters in tree-decomposition based approaches.

Section 2 presents the analysis of the complexity of enumerative algorithms like FC based on the size of relations. Section 3 describes how this analysis can be exploited to propose new complexity bounds for tree-decomposition approaches and indicates a new comparison with hypertree-decomposition ones.

## 2 Complexity of FC Revisited

### 2.1 Complexity expressed by the size of domains

A *finite constraint satisfaction problem* or *finite constraint network* $(X, D, C, R)$ is defined as a set of variables $X = \{x_1, \dots x_n\}$, a set of domains $D = \{d_1, \dots d_n\}$ (the domain $d_i$ contains all the possible values for the variable $x_i$), and a set $C$ of constraints among variables. A constraint $c_i \in C$ on an ordered subset of variables, $c_i = (x_{i_1}, \dots x_{i_{a_i}})$ ($a_i$ is called the *arity* of the constraint $c_i$), is defined by an associated relation $r_i \in R$ of allowed combinations of values for the variables in $c_i$. Note that we take the same notation for the constraint $c_i$ and its scope. We denote $a$ the maximal arity of the constraints in $C$. Without loss of generality, we assume that each variable is involved in at least one constraint. A solution of $(X, D, C, R)$ is an assignment of each variable which satisfies all the constraints. The CSP structure can be represented by the hypergraph $(X, C)$, called the *constraint hypergraph*. If each constraint of a CSP is binary (i.e. involves exactly two variables), then $(X, C)$ is a graph called the *constraint graph*.

In this paper (as in [6]), we assume that the relations are not empty and can be represented by tables as in relational database theory. Then, we denote by $S$ the size of a CSP (which verifies $S \leq n.m + a.r.|C|$ where $r = max\{|r_i| : r_i \in R\}$). Let $Y = \{x_1, \dots x_k\}$ be a subset of $X$ and $\mathcal{A}$ an assignment of $Y$. $\mathcal{A}$ can be considered as a tuple $\mathcal{A} = (v_1, \dots v_k)$. The *projection* of $\mathcal{A}$ on a subset $Y'$ of $Y$, denoted $\mathcal{A}[Y']$, is the restriction of $\mathcal{A}$ to the variables of $Y'$. The projection of the relation $r_i$ on the subset $Y'$ of $c_i$ is the set of tuples $r_i[Y'] = \{t[Y']|t \in r_i\}$. The join of relations will be denoted $\bowtie$, and the join of $\mathcal{A}$ with a relation $r_i$ is $\mathcal{A} \bowtie r_i = \{t|t \; is \; a \; tuple \; on \; Y \cup c_i \; and \; t[Y] = \mathcal{A} \; and \; t[c_i] \in r_i\}$.

The basic approach for solving CSP is based on the classical procedure called *Backtracking (BT)*. The time complexity of this basic algorithm is $O(a.r.|C|.m^n)$ since the number of potential nodes developed during the search is $m^n$ and assuming that a constraint check $\mathcal{A}[c_i] \in r_i$ is computable in $O(a.r)$. To simplify the notations, it can be expressed by $O(S.m^n)$. Generally, this algorithm is never used because it is clearly inefficient in practice.

The most classical approach to improve BT is based on filtering. After any assignment $\mathcal{A}$ on $Y$ with $x_k$ the last assigned variable, a filtering will be achieved in the domains of future variables. This filtering removes the values which are not compatible with the assignment of $x_k$. The first algorithm proposed for such a filtering is Forward Checking (FC [7]). It was initially defined on binary CSPs. Numerous extensions and generalizations of FC have been proposed in order to solve non-binary CSPs or to exploit more powerful filters [7, 13, 1]. In this paper, we consider one of these extensions called nFC2 [1] whose level of filtering seems to realize a good trade-off according to the presented empirical results obtained on non-binary CSPs.

Let us consider a current assignment $\mathcal{A}$ which satisfies all the constraints included in $Y$. After the assignment of the last variable $x_k$, nFC2 applies arc-consistency in one pass (see [1] for more details) on each constraint of $C_{c,f}$ where $C_{c,f}$ is the set of constraints involving the current variable and at least one future variable (formally, $C_{c,f} = \{c_j \in C|x_k \in c_j \; and \; c_j \not\subset Y\}$). Consequently, the obtained filtered domains depend on the order according to which the constraints are processed. In order to handle easily the filtered domains, we define nFC2$_m$ in such a way that its filtering can be seen as the minimal filtering of nFC2 among all the possible constraint orders. Actually, in nFC2$_m$, the domain of each future variable is filtered independently of ones of the other future variables. The filtered domain of a future variable $x_i$ is the set of

| $c_1$ | | | $c_2$ | | | $c_3$ | | |
|---|---|---|---|---|---|---|---|---|
| x | y | z | u | v | w | x | y | w |
| a | a | a | a | a | a | a | a | a |
| a | b | c | a | b | b | a | b | c |
| a | c | b | c | c | c | | | |

| Assignment | Algorithm | Action |
|---|---|---|
| (x, a) | nFC2 | AC($\{c_3\}$) then AC($\{c_1\}$) |
| | nFC2$_m$ | AC($\{c_3\}$) and AC($\{c_1\}$) |
| (u, a) | nFC2 | AC($\{c_2\}$) |
| | nFC2$_m$ | AC($\{c_2\}$) |

| $(x, a)$ | nFC2 | nFC2$_m$ |
|---|---|---|
| $d_x^{\mathcal{A}}$ | a | a |
| $d_y^{\mathcal{A}}$ | a, b | a, b |
| $d_z^{\mathcal{A}}$ | a, c | a, b, c |
| $d_u^{\mathcal{A}}$ | a, c | a, c |
| $d_v^{\mathcal{A}}$ | a, b, c | a, b, c |
| $d_w^{\mathcal{A}}$ | a, c | a, c |

| $(u, a)$ | nFC2 | nFC2$_m$ |
|---|---|---|
| $d_x^{\mathcal{A}}$ | a | a |
| $d_y^{\mathcal{A}}$ | a, b | a, b |
| $d_z^{\mathcal{A}}$ | a, c | a, b, c |
| $d_u^{\mathcal{A}}$ | a | a |
| $d_v^{\mathcal{A}}$ | a | a |
| $d_w^{\mathcal{A}}$ | a | a |

**Figure 1. Filtering caused by nFC2 and nFC2$_m$ after assignments $(x, a)$ and $(u, a)$.**

values $d_i^{\mathcal{A}} = \{v_i \in d_i^{\mathcal{A}[Y\setminus\{x_k\}]} | \forall c_j \in C_{c,f}\ s.t.\ x_i \in c_j, \exists t \in r_j, t \in \prod_{x_l \in c_j} d_l^{\mathcal{A}[Y\setminus\{x_k\}]}\ and\ t[x_i] = v_i\}$ and one of an assigned variable $x_j$ is $d_j^{\mathcal{A}} = \{\mathcal{A}[x_j]\}$. Initially, $d_i^{\emptyset} = \{v_i \in d_i | \forall c_j \in C, x_i \in c_j, \exists t \in r_j, t[x_i] = v_i\}$ (i.e. we only consider the value $v_i$ which appears in each relation involving $x_i$). By so doing, given an assignment $\mathcal{A}$, the filtering caused by nFC2$_m$ (i.e. the set of pairs $(x_i, v_i)$ s.t. the value $v_i$ has been removed from the domain of $x_i$) is clearly included in one caused by nFC2 (for any constraint order). Like nFC2, if the domain of a future variable becomes empty, then nFC2$_m$ backtracks; otherwise, it keeps on the search with a future variable. One can easily prove that nFC2$_m$ is sound and complete and terminates (the proof is similar to one of nFC2 proposed in [1]). Regarding the time complexity, the size of the search space is bounded by $O(m^n)$. The analysis reported in [1] indicates that the local cost at any node, i.e. the number of checks performed, is $O(|C_{c,f}|.(a-1).m^{a-1})$. Note that it is easy to express this complexity by taking into account the maximal size $r$ of the relations $r_i$, to get finally $O(|C_{c,f}|.a.r)$. Necessarily $|C_{c,f}|.a.r \in O(S)$, with $S$ the size of the CSP, and so the cost of nFC2 and nFC2$_m$ can be bounded now by $O(S.m^n)$.

For example, Figure 1 illustrates the differences between nFC2 and nFC2$_m$ with the n-ary CSP used in [1] (figure 1, p 210). The CSP has 6 variables $\{x, y, z, u, v, w\}$ with the same domain $\{a, b, c\}$, and 3 constraints $c_1(x, y, z)$, $c_2(u, v, w)$ and $c_3(x, y, w)$. The initial domains are reduced to those values in the relations. nFC2 and nFC2$_m$ enforce arc-consistency ("in one pass" [1]) on the same sets of constraints. Since nFC2 takes into account the filterings done to enforce arc-consistency on previous constraints, the constraint order is important. In contrast, nFC2$_m$ behaves like if it performs arc-consistency on all constraints at the same time. Thus, after the assignment $(x, a)$, if nFC2 enforces AC($\{c_1\}$) before AC($\{c_3\}$), the value $b$ for the variable $z$ will not be filtered unlike in the example.

## 2.2 Complexity expressed by the size of relations

Now, we analyze the complexity of nFC2$_m$ w.r.t. the size of relations. Theorem 1 states that any assignment considered by nFC2$_m$ satisfies both the constraints whose variables are totally assigned and ones whose a part of variables is assigned. By so doing, we can ensure that nFC2$_m$ enumerates the assignments in the limit of the joins of the corresponding relations, what allows us to refine its complexity. To simplify the formalism, we denote by $Y_k$ the ordered set $Y = (x_1, \ldots, x_k) \subsetneq X$, $C_Y = \{c_j \in C | c_j \cap Y \neq \emptyset\}$ and $C_{Y,f} = \{c_j \in C_Y | c_j \not\subset Y\}$.

**Theorem 1** *If* $\mathcal{A} = (v_1, \ldots, v_k) \in \bowtie_{c_i \in C_{Y_k}} r_i[c_i \cap Y_k]$ *and* $\forall j, k+1 \leq j \leq n, d_j^{\mathcal{A}} \neq \emptyset$, *then* $\forall v_{k+1} \in d_{k+1}^{\mathcal{A}}, (v_1, \ldots, v_k, v_{k+1}) \in \bowtie_{c_i \in C_{Y_{k+1}}} r_i[c_i \cap Y_{k+1}]$.

**Proof:** Let $\mathcal{A}' = (v_1, \ldots, v_k, v_{k+1})$ be the extension of the assignment $\mathcal{A}$ obtained by assigning $x_{k+1}$ with the value $v_{k+1} \in d_{k+1}^{\mathcal{A}}$.
We want to prove that $\mathcal{A}' \in \bowtie_{c_i \in C_{Y_{k+1}}} r_i[c_i \cap Y_{k+1}]$.
First, we consider a partition of the constraint set $C_{Y_{k+1}}$ as follows:

$$\begin{aligned} C_{Y_{k+1}} &= \{c_i \in C | c_i \subset Y_{k+1}\} \cup C_{Y_{k+1},f} \quad (1) \\ &= \{c_i \in C | c_i \subset Y_k\} \quad \cup \\ &\quad \{c_i \in C | c_i \backslash Y_k = \{x_{k+1}\}\} \quad \cup \\ &\quad \{c_i \in C | c_i \in C_{Y_{k+1},f}\ and\ x_{k+1} \in c_i\} \quad \cup \\ &\quad \{c_i \in C | c_i \in C_{Y_{k+1},f}\ and\ x_{k+1} \notin c_i\} \quad (2) \end{aligned}$$

So, we have: $\bowtie_{c_i \in C_{Y_{k+1}}} r_i[c_i \cap Y_{k+1}]$

$$\begin{aligned} &\overset{(1)}{=} (\bowtie_{c_i \subset Y_{k+1}} r_i) \bowtie (\bowtie_{c_i \in C_{Y_{k+1},f}} r_i[c_i \cap Y_{k+1}]) \quad (1) \\ &\overset{(2)}{=} (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \backslash Y_k = \{x_{k+1}\}} r_i) \bowtie \\ &\quad (\bowtie_{c_i \in C_{Y_{k+1},f}\ and\ x_{k+1} \in c_i} r_i[c_i \cap Y_{k+1}]) \bowtie \\ &\quad (\bowtie_{c_i \in C_{Y_{k+1},f}\ and\ x_{k+1} \notin c_i} r_i[c_i \cap Y_k]) \quad (3) \end{aligned}$$

Then, by construction of the non-empty domain $d_{k+1}^{\mathcal{A}}$, for each constraint $c_i \in C$ s.t. $x_{k+1} \in c_i$ and $c_i \cap Y \neq \emptyset$, there exists a tuple $t \in r_i$ s.t. $t[Y_k \cap c_i] = \mathcal{A}[Y_k \cap c_i]$ and $t[x_{k+1}] = v_{k+1}$, i.e. s.t. $t[c_i \cap Y_{k+1}] = \mathcal{A}'[c_i \cap Y_{k+1}]$.

$$\text{So,}\quad \mathcal{A}' \left[ \bigcup_{c_i | x_{k+1} \in c_i} (c_i \cap Y_{k+1}) \right] \in (\bowtie_{c_i \backslash Y_k = \{x_{k+1}\}} r_i) \bowtie (\bowtie_{c_i \in C_{Y_{k+1},f}\ and\ x_{k+1} \in c_i} r_i[c_i \cap Y_{k+1}]).$$

Moreover, $\mathcal{A} \in \bowtie_{c_i \in C_{Y_k}} r_i[c_i \cap Y_k]$

$$\begin{aligned} &= (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \in C_{Y_k,f}} r_i[c_i \cap Y_k]) \\ &= (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \in C_{Y_k,f}\ and\ x_{k+1} \notin c_i} r_i[c_i \cap Y_k]) \bowtie \\ &\quad (\bowtie_{c_i \in C_{Y_k,f}\ and\ x_{k+1} \in c_i} r_i[c_i \cap Y_k]) \end{aligned}$$

$$\text{So,}\quad \mathcal{A}'[Y_k] = \mathcal{A} \in (\bowtie_{c_i \subset Y_k} r_i) \bowtie (\bowtie_{c_i \in C_{Y_k,f}\ and\ x_{k+1} \notin c_i} r_i[c_i \cap Y_k]).$$

Hence, from (3), $\mathcal{A}' \in \bowtie_{c_i \cap Y_{k+1}} r_i[c_i \cap Y_{k+1}]$. □

**Theorem 2** *The time complexity of nFC2$_m$ for solving a CSP $(X, D, C, R)$ is $O(S.r^{|C|})$.*

**Proof:** For any assignment $\mathcal{A}$ on $Y$ considered by nFC2$_m$ (the variable order is assumed static and induced by $Y$), we have $\mathcal{A} \in \bowtie_{c_i \in C_Y} r_i[c_i \cap Y]$ (theorem 1). So, as the number of nodes is equal to the number of assignments, it can be bounded, for a given depth $|Y|$, by $\prod_{c_i \in C_Y} |r_i|$. Moreover, $C_Y \subset C$. Hence, $\prod_{c_i \in C_Y} |r_i| \leq \prod_{c_i \in C} |r_i| \leq r^{[C|}$. So, as the height of the search tree is bounded by $n$, the number of nodes is bounded by $n.r^{|C|}$. As the cost in each node is $O(S)$, nFC2$_m$ has a complexity in $O(S.r^{|C|})$. □

This complexity can be refined again by taking into account the notion of minimum cover of the constraint set:

**Definition 1** *Given a set $X$ and $C$, a family of subset of $X$, a cover of $X$ is a subset $C' \subset C$ such that $\cup_{c_i \in C'} = X$. $C'$ is a minimum cover of $X$ if there is no cover $C''$ such that $|C''| < |C'|$. The value $|C'|$ will be denoted $k_{(X,C)}$.*

**Theorem 3** *The time complexity of nFC2$_m$ for solving a CSP $(X, D, C, R)$ is $O(S.r^{k_{(X,C)}})$.*

**Proof:** For any assignment $\mathcal{A}$ on $Y$ considered by nFC2$_m$ (the variable order is assumed static and induced by $Y$), we have $\mathcal{A} \in \bowtie_{c_i \in C_Y} r_i[c_i \cap Y]$ (theorem 1). Moreover, at each level of the search tree, we have $\{c_i \in C' | c_i \cap Y \neq \emptyset\} \subset C_Y$ with $C'$ a cover of $X$. Hence, $\mathcal{A} \in \bowtie_{c_i \in C' | c_i \cap Y \neq \emptyset} r_i[c_i \cap Y]$. So, at each level, the number of nodes is bounded by $\prod_{c_i \in C' | c_i \cap Y \neq \emptyset} |r_i| \leq \prod_{c_i \in C'} |r_i| \leq r^{|C'|}$. Hence, the total number of nodes is bounded by $n.r^{|C'|}$. Therefore, as the local cost for each node is $O(S)$, nFC2$_m$ has a complexity in $O(S.r^{|C'|})$. If, now we consider a minimum cover, this complexity becomes $O(S.r^{k_{(X,C)}})$. □

We can note that nFC2$_m$ exploits naturally a minimum cover of $C$ without having to achieve an expensive computation of this cover (finding a minimum cover is NP-Hard [5]). This result can be extended to any other algorithm which maintains a filtering at least as powerful as nFC2$_m$'s one. For instance, it still holds for nFC$i$ ($i \geq 2$) and MAC.

Let us consider again the example in Figure 1. A minimum cover of this problem is the set $\{c_1, c_2\}$. Let us consider the join of $r_1$ and $r_2$ and the search tree of nFC2$_m$ according to the static variable order $(x, y, z, u, v, w)$. One can observe that each tuple computed by nFC2$_m$ is in the join of $r_1$ and $r_2$. In other words, the cost of nFC2$_m$ is less than one of the join $r_1 \bowtie r_2$.

Now, if we generalize this example by adding any number of variables in $c_1$ and $c_2$, the classical complexity is $O(S.m^n)$ which is exponential while it is $O(S.r^2)$ (polynomial) using nFC2$_m$ with the new evaluation. Moreover, the theoretical result given by the theorem 3 surprisingly contradicts empirical evaluations. Indeed, generally more a problem is constrained, more it is easy to solve (excepted for under-constrained problems). So, the bounds given by theorems 2 and 3 with $S.r^{k_{(X,C)}} \leq S.r^{|C|}$ indicate that less the problem is constrained, less its complexity is high. But, observing the example, it is more efficient to check also constraint $c_3$ during the search on $c_1 \cup c_2$ (the whole problem), than to check only $c_1$ and $c_2$ because the dead-ends will be found earlier.

# 3 Complexity of Tree-Decomposition Methods Revisited

For lack of place, we do not give here neither details, nor proofs for results presented in this section. So interested readers must see [8]. The decomposition of constraint networks was introduced in [4] with Tree-Clustering (TC). TC and other methods based on this approach [3] rely on the notion of tree-decomposition of graphs [12]. Nevertheless, given a non-binary CSP, and so a constraint hypergraph, we can exploit it by considering its *primal graph*. Let $H = (X, C)$ be a hypergraph, the primal graph of $H$ is the graph $G = (X, A_C)$ where $A_C = \{\{x, y\} \subset X : \exists c_i \in C \; s.t. \; \{x, y\} \subset c_i\}$. So, given a CSP, we consider its primal graph to define an associated tree-decomposition of the CSP.

Assume that we have a tree-decomposition of width $w$ for the constraint network. Initially, TC was defined on binary CSPs. Nevertheless, extensions have been defined for non-binary CSPs (see [3]). Here we give a particular generalization of TC. Given a tree-decomposition $(E, T)$ associated to a CSP where $E = \{E_1, E_2, \ldots E_N\}$ is the set of clusters, the subproblem associated to a cluster $E_i$ is defined by the same set of variables $E_i$ but considers now included constraints and constraints which intersect $E_i$. Formally, the set of constraint for a cluster $E_i$ is $C_{E_i} = \{c_j \in C : c_j \cap E_i \neq \emptyset\}$. The relations associated to these constraints are $R_{E_i} = \{r_j[c_j \cap E_i] : c_j \in C_{E_i}\}$. Each subproblem (cluster) is solved independently and after this step, TC solves the whole CSP as an acyclic CSP. The cost of the first step is bounded by the cost of finding all the solutions of subproblems. This complexity is generally defined by $O(S.m^{w+1})$, since the maximal size of clusters is $w + 1$ assuming that the cost for enumerating solutions on clusters is $O(S.m^{w+1})$. Moreover, the maximal number of solutions in each cluster is also bounded by $O(m^{w+1})$, and then, the cost of the last step is also bounded by $O(S.m^{w+1})$.

Theorem 3 allows us to propose another bound, assuming that we use a procedure as nFC2$_m$ for solving each cluster. The cost of solving a cluster $E_i$ is now $O(S_i.r^{k_i})$, where $S_i$ is the size of the subproblem associated to $E_i$, while $k_i = k_{(E_i, C_{E_i})}$ (i.e. the parameter associated to a minimum cover of $E_i$). Note that the size of the set of solutions in $E_i$ is bounded by $O(r^{k_i})$. So the total cost for solving the whole decomposed CSP is $O(S.r^k)$ where $k = max\{k_i : i \in I\}$. So we have now:

**Theorem 4** *The time complexity of Tree-Decomposition*

*methods for solving CSPs is* $O(S.r^k)$.

[6] provides a theoretical comparison of the well known structural methods for solving CSP and a hierarchy on these methods related to their power w.r.t. the classes of problems they can solve in polynomial time. [6] states that the hypertree decomposition strongly generalizes the other methods like Tree-Clustering. This notion of hypertree decomposition can be seen as a generalization of one of tree-decomposition.

A hypertree-decomposition associates a tree-decomposition with a covering by hyperedges of each cluster. The aim is to reduce the size of the covering sets (the number of covering hyperedges for each cluster). Thus, a CSP instance can be solved in $O(S.r^h)$ with $h$ the width of a hypertree-decomposition of its constraint hypergraph (the method, proposed in [6], consists in computing an acyclic equivalent CSP by solving each cluster thanks to relation joins and then in solving classically the obtained acyclic CSP). Moreover, there exists a class of problems whose hypertreewidth is bounded while their treewidth is not. According to this, hypertree-decomposition strongly generalizes Tree-Clustering.

Using the results of section 2, we can prove that if a CSP can be solved in $O(S.r^h)$ thanks to a hypertree-decomposition $HD$ of its hypergraph, with $h$ its width, it can also be solved with the same time complexity bound using TC on one tree-decomposition induced by the hypertree-decomposition and denoted $TD_{HD}$. Then, since $HD$ defines a cover of the clusters of hyperedges whose size is at most $h$, a minimum cover of each cluster in $TD_{HD}$ is also at most $h$.

**Theorem 5** *$k \leq h$ with $k = max\{k_{(E_i,C_{E_i})} : i \in I'\}$.*

**Corollary 1** *The TC time complexity for solving $\mathcal{P}$ is $O(S.r^h)$.*

This result proves that TC performs at least as good as hypertree-decomposition.

## 4 Conclusion

Firstly, we have shown that the complexity of algorithms as nFC2 (backtracking + filtering) can be expressed by $O(S.r^k)$, with $r$ the maximum size of relations associated to the constraints, $k$ a structural parameter of the CSP and $S$ the size of the CSP. Previously, this complexity was essentially expressed by $O(S.m^n)$ with $m$ the size of domains, and $n$ the number of variables. This result shows that the complexity of structural methods using tree-decomposition for solving CSPs is finally at the same level than hypertree-decomposition. This result gives a theoretical explanation of the experimental results observed by the community. Among the different continuations of this work, one of the most interesting should be to analyze the possible modification of the hierarchy between decomposition methods introduced in [6]. Another could be to improve the practical efficiency of decomposition methods by computing better tree-decompositions of constraint networks, considering also the quality of the associated hypertree-decomposition.

**Acknowledgments** This work is supported by an ANR grant (STAL-DEC-OPT project).

## References

[1] C. Bessière, P. Meseguer, E. C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141:205–224, 2002.

[2] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of AAAI*, pages 22–27, 2006.

[3] R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

[4] R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.

[5] M. Garey and D. Johnson. Computer and Intractability. In *Freeman*, 1979.

[6] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.

[7] R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[8] P. Jégou, S. Ndiaye, and C. Terrioux. A new Evaluation of Forward Checking and its Consequences on Efficiency of Tools for Decomposition of CSPs. Technical report, LSIS, 2008.

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

[10] P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proc. of ECAI*, pages 196–200, 2004.

[11] R. Marinescu and R. Dechter. Dynamic Orderings for AND/OR Branch-and-Bound Search in Graphical Models. In *Proc. of ECAI*, pages 138–142, 2006.

[12] N. Robertson and P. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.

[13] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI*, pages 125–129, 1994.

[14] C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proc. of CP*, pages 709–723, 2003.