Combined Strategies for Decomposition-based Methods for solving CSPs

Philippe Jégou[1], Samba Ndojh Ndiaye[2], Cyril Terrioux[1]
[1] LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen, 13397 Marseille Cedex 20, France
{philippe.jegou, cyril.terrioux}@univ-cezanne.fr
[2] LIRIS - UMR CNRS 5205
Université Claude Bernard (Lyon 1)
43 boulevard du 11 novembre 1918 (Nautibus), 69622 Villeurbanne Cedex, France
samba-ndojh.ndiaye@liris.cnrs.fr

Abstract

In this paper, we consider theoretical and practical methods based on decompositions of constraint networks. We exploit the fact that decomposition-based methods can be used considering two steps. The first step is related to the (hyper)graphical decomposition (e.g. Tree-Decomposition [16] or Hypertree-Decomposition [7]) while the second step exploits the decomposition to solve the CSPs. Thanks to this approach, we define then hybrid methods which can be optimal from a theoretical viewpoint while being efficient in practice. The complexity analysis of these combined methods allows us to give a more detailed presentation of the Constraint Tractability Hierarchy introduced in [7]. Finally, we justify our approach with experimental results.

1 Introduction

Here, we are interested in the practical and the theoretical efficiency of methods exploiting structural features of CSPs. A CSP can be considered as the problem of checking if a finite set X of variables can be assigned in their finite 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 more generally to find one solution, or to enumerate the set of solutions. 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 as FC (Forward Checking) or MAC. While these algorithms can be really efficient from a practical viewpoint, their complexity is classically evaluated by $O(S.m^n)$ where S is the size of the considered CSP, n the number of variables and m the maximum size of domains of variables.

Different approaches have been proposed to improve these bounds, for example by exploiting structural properties that exist frequently in real problems. Generally, they rely on the properties of a tree-decomposition (TD) [16] or a hypertree-decomposition (HD) [7] of the constraint network which formalizes the structure and consequently allows to express topological properties. If we consider a TD 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 practical cases, $w \ll n$. If we consider a HD of width h, the time complexity is then $O(S.r^h)$, with r the maximum size of relations (tables) associated to constraints. [7] has shown that hypertree-decomposition is better than tree-decomposition, since $h^* \leq w^*$ (h^* and $w^*$are optimal values for h and w). Moreover, the authors have introduced a formal tool called "Constraint Tractability Hierarchy" for comparing several decomposition-based methods. This theoretical tool considers classes of instances which can be solved in polynomial time. It appears, for example, that hypertree-decomposition method (MHD) [7] is more powerful than Tree-Clustering (TC) [5] since the class of tractable instances for MHD includes strictly the one of TC. This hierarchy offers an important theoretical tool for comparing decomposition-based methods. However, it is well known that it can exist a gap between theoretical and practical performances of a same method. Thus, in this paper, we will integrate to our study some aspects related to the practical behavior and use of these methods. Generally, decomposition-based methods are not used as assumed in the hierarchy which only considers optimal decompositions. Yet, computing optimal decompositions can be really expensive : it can require more time than the solving of the problem without decomposition. So, in practice, we prefer to use heuristic algorithms (approximating optimal values w^* or h^*) to compute a decomposition. Then the next step consists in solving the decomposed CSP. In the hierarchy,

there is rigid relation between the solving method and the decomposition used. Here, we decide to break this relation in order to create a separation between decompositions approaches and the solving methods based on them. We introduce and study new combined approaches, assuming that a particular graphical decomposition (e.g. HD) can be exploited by a solving method which was initially defined to use another decomposition (e.g. TC). Then, we show that this kind of hybrid methods has a real interest from a theoretical viewpoint. Indeed, we obtain original complexity results allowing to present the hierarchy of [7] more precisely. For example, we exploit a recent result on the time complexity of FC [12] which allows us to express the time complexity of TC w.r.t. the induced hypertree-width h while generally, the evaluation was limited to induced tree-width w. Moreover, this approach allows us to propose operational methods based on the hypertree-decomposition, which can be now efficiently implemented. Finally, we show empirically that our approach seems well adapted to compare decomposition-based methods with respect to their ability to solve CSPs. So, our contribution can be summarized as following. *(1) We propose a framework to define and to analyze hybrid methods for solving CSPs using decompositions. (2) We present more precisely the hierarchy of [7]. (3) We define a solving method based on hypertree-decomposition which can be efficient from a practical viewpoint. (4) We propose criteria to optimize in order to get more efficient decompositions. (5) We define extensions of TC and BTD with the same time complexity bounds than MHD ones.*

Section 2 recalls notations, results on complexity of enumerative algorithms and decomposition methods. Section 3 introduces new methods based on combined versions of decompositions and studies them from a theoretical viewpoint, proposing new complexity bounds. Section 4 presents experiments while section 5 concludes.

2 Preliminaries

2.1 Notations

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 (hyper)graph*. In this paper (as in [7]), 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 $\mathcal{A} \bowtie r_i = \{t|t \; is \; a \; tuple \; on \; Y \cup c_i \; and \; t[Y] = \mathcal{A} \; and \; t[c_i] \in r_i\}$.

2.2 Complexity of Enumeration

The basic approach for solving a 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. The first algorithm proposed for such a filtering is Forward Checking (FC). 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 [2]. One of these extensions called nFC2 [2] has a filtering level which seems to realize a good compromise. The complexity of nFC2 is also bounded by $O(S.m^n)$ as indicated in [2]. Recently, [12] proposes a new bound which considers r the maximum size of relations (tables) associated to constraints. It has shown that the time complexity of FC or MAC is $O(S.r^k)$, where k is the size $|C'|$ of a minimum cover C' of X. Note that C' is a minimum cover of X if C' is a cover of X (that is $C' \subset C$ and $\cup_{c_i \in C'} = X$) and there is no cover C'' such that $|C''| < |C'|$. This result can be extended to any other algorithms which maintain a filtering at least as powerful as nFC2's one. For instance, it still holds for nFCi ($i \geq 2$) and MAC.

2.3 Decomposition-based Methods

The decomposition of constraint networks was introduced in [5] with Tree-Clustering (TC). TC and other methods based on this approach (see [3]) rely on the notion of tree-decomposition of graphs. Nevertheless, given a non-binary CSP, and so a constraint hypergraph, we can exploit

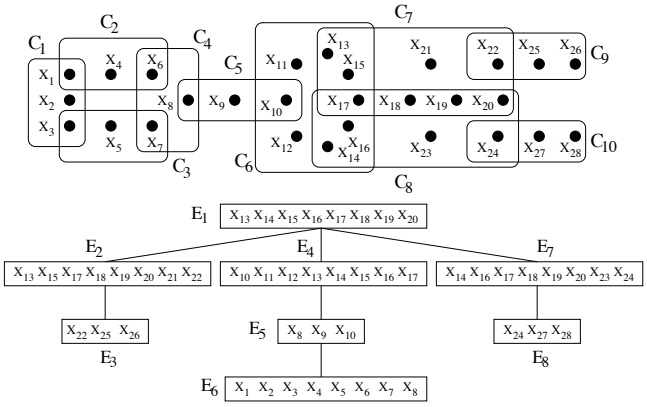

Figure 1. A constraint hypergraph and one of its optimal TD ($w^* = 7$).

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.

Definition 1 *A tree-decomposition of a graph $G = (X, A_C)$ is a pair (E, T) where $T = (I, F)$ is a tree with nodes I and edges F and $E = \{E_i : i \in I\}$ a family of subsets of X, s.t. each subset (called cluster) E_i is a node of T and verifies:*

- *(i)* $\cup_{i \in I} E_i = X$,
- *(ii) for each edge $\{x, y\} \in A_C$, there exists $i \in I$ with $\{x, y\} \subset E_i$, and*
- *(iii) for all $i, j, k \in I$, if k is in a path from i to j in T, then $E_i \cap E_j \subset E_k$.*

The width w of a tree-decomposition (E, T) is equal to $max_{i \in I}|E_i| - 1$. The tree-width w^* of G is the minimal width over all the tree-decompositions of G.

Figure 1 presents a constraint hypergraph and one of its possible TD with a minimal width ($w^* = 7$).

In [5], Tree-Clustering - denoted here TC-1989 - was initially introduced using a polynomial time algorithm (MCS [17]) for finding the TD. More precisely, the method is summarized by:

1. Compute a TD of the constraint network
   (a) Triangulation of the primal graph using MCS
   (b) Identify clusters of variables (maximal cliques)
   (c) Form the tree of clusters (join-tree)
2. Solve the subproblems defined by clusters of variables
3. Solve the tree problem in a backtrack-free manner

Note that the complexity of the step 1 is limited to $O(n^2)$. Nevertheless, we have no guarantee about the optimality for the parameter w. The complexity of the step 2 is $O(S.m^{w+1})$. So, given a constraint network, the more the width of the decomposition is small, the more the time complexity will be small. Unfortunately, the width of the TD computed thanks to MCS can be very far from the optimum. Moreover, finding an optimal TD is an NP-hard problem. Note that in [3], TC is called Join-Tree Clustering and each sub-problem is defined as the set of variables belonging to a cluster and the constraints associated to a cluster are the constraints whose scope is included in the set of variables of the cluster. The complexity of the third step is $O(|C|.w.log(m).m^{w+1})$. Finally, we can consider that the total cost is $O(S.m^{w+1})$. Note that the space complexity is related to the storage of solutions of sub-problems, $O(n.a.m^{w+1})$. In [3], Dechter suggests to limit the required space in memorizing only a part of solutions on cluster, their projection on intersection, limiting then the space complexity to $O(n.a.m^s)$ where s is the maximum size of intersection between clusters. Finally, note that TC-1989 was defined on binary CSPs but extensions have been defined for non-binary CSPs [3]. Optimizations of TC-1989 have been proposed to avoid some of its drawbacks (required memory space, total solving of sub-problems before checking for global consistency of the CSP, value of w) which make this method not very efficient in practice. For example, BTD [14] can be considered has an efficient approach to exploit tree-decomposition. This practical efficiency is due to the fact that BTD applies an enumerative algorithm guided by an ordering of variables induced by the TD. This approach generally avoids to solve completely all clusters. As for TC-1989, time complexity of BTD is $O(S.m^{w+1})$ while its space complexity is $O(n.a.m^s)$. Moreover BTD considers a given TD which can be obtained using heuristic algorithms or exact algorithms.

We can consider that TC-1989 is driven by variables since clusters are defined by set of variables. In [7], a new method has been proposed, which considers now clusters of constraints. The approach is based on the notion of hypertree-decomposition which can be seen as a generalization of tree-decomposition.

Definition 2 *Given a hypergraph $H = (X, C)$, a hypertree for the hypergraph H is a triple (T, χ, λ) where $T = (N, F)$ is a rooted tree, and χ and λ are labelling functions which associate to each vertex $p \in N$ two sets $\chi(p) \subset X$ and $\lambda(p) \subset C$. We denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.*
A hypertree-decomposition of H is a hypertree $HD = (T, \chi, \lambda)$ for H which satisfies all the following four conditions:

- *(i) for each edge $c \in C, \exists p \in vertices(T)$ s.t. $c \subset \chi(p)$,*

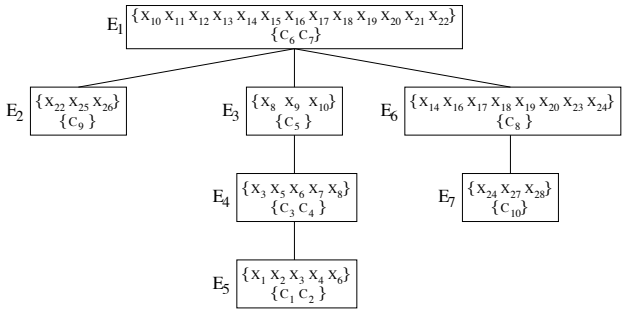

Figure 2. An optimal HD ($h^* = 2$).

- *(ii) for each vertex $x \in X$, the set $\{p \in vertices(T) : x \in \chi(p)\}$ induces a (connected) subtree of T,*
- *(iii) for each $p \in vertices(T), \chi(p) \subset \cup_{c \in \lambda(p)} c$,*
- *(iv) for each $p \in vertices(T), \cup_{c \in \lambda(p)} c \cap \chi(T_p) \subset \chi(p)$.*

An edge $c \in C$ is strongly covered in HD if there exists $p \in vertices(T)$ such that $c \subset \chi(p)$ and $c \in \lambda(p)$. A hypertree-decomposition HD is a complete decomposition of H if every edge of H is strongly covered in HD.
The width h of a hypertree-decomposition $HD = (T, \chi, \lambda)$ is $max_{p \in vertices(T)} |\lambda(p)|$. The hypertree-width h^* of H is the minimum width over all its hypertree-decompositions.

Remark that acyclic hypergraphs are precisely the hypergraphs having a hypertree-width equal to one. For solving CSP, we only consider complete HD. Figure 2 presents a complete HD of the hypergraph in figure 1. Based on this notion of hypertree-decomposition, the method, denoted here MHD-1999, has been proposed in IJCAI 1999 ([7]) is defined by three steps.

1. Compute a HD of the CSP
2. Solve each cluster using a join of relations
3. Solve the tree problem in a backtrack-free manner

[7] presents an evaluation of the complexity, assuming that the step 1 can be realized in $O(|C|^{2h^*}.n^2)$ thanks to the algorithm *opt-k-decomp* [7]. For that, the considered hypergraph must have a bounded hypertree-width. The *opt-k-decomp* algorithm is able to compute an optimal HD in a polynomial time for all hypergraphs whose hypertree-width is bounded. Note that the cost of solving each cluster in step 2 is bounded by $O(S.r^h)$. The space complexity is related to the size of an associated relation, that is $O(r^h)$. The practical interest of MHD-1999 has not been clearly shown yet. This is probably due to the lack of efficient algorithms to compute HD, and to the space ($O(n.r^h)$) required to compute the step 2. Nevertheless, from a theoretical viewpoint, this method is clearly relevant as indicated by the "Constraint Tractability Hierarchy" introduced in [7] (see figure 3). This hierarchy provides a theoretical comparison of the well known structural methods for solving CSP and a hierarchy on these methods. In this hierarchy, a decomposition-based method D is considered as running in four steps.

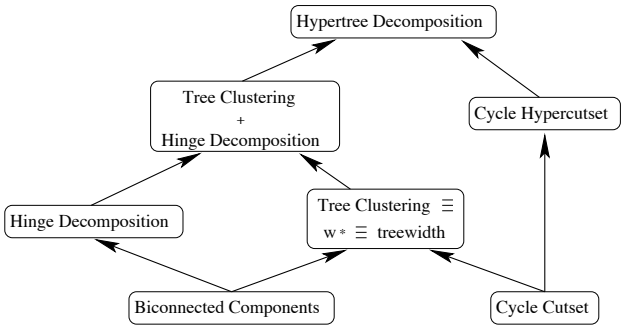

Figure 3. The Constraint Tractability Hierarchy

1. Recognize that a CSP P is tractable (by verifying if the width of the decomposition used by D, denoted $D\text{-}width$, is bounded by a constant).
2. Compute, in polynomial time, a CSP decomposition for P whose width is less than the constant.
3. Transform P, in polynomial time, into an equivalent CSP P'.
4. Solve P' in polynomial time (in $S^{D-width}$ since $D\text{-}width$ is bounded by a constant).

So, if we consider MHD-1999, $D\text{-}width = h^*$. Each CSP whose hypertree-width is bounded is tractable (can be solved in polynomial time) by MHD-1999. Formally, let D_1 and D_2 be two decomposition-based methods. We consider that D_2 *strongly generalizes* D_1 (represented by an arc (D_1,D_2) in the figure 3) if D_2 generalizes D_1 (each problem tractable using D_1 is also tractable using D_2) and D_2 beats D_1 (there is a class of problems tractable according to D_2 but not according to D_1). While MHD-1999 is at the top of the hierarchy, note that formally, TC-1989 should not be included since this method does not guarantee to use a TD whose width is less than the constant of the first step. Actually, the width of the TD computed thanks to MCS can be far from the optimal and so significantly greater than the constant. There are other decomposition-based methods which outperform the HD: generalized hypertree-decomposition [8] and fractional hypertree-decomposition [10] but the methods based on these decompositions cannot be considered in the tractable hierarchy. Thus, the hypertree-decomposition is at the top of this hierarchy. Finally, note that for efficient implementations of HD, Gottlob's group has recently proposed interesting algorithmic tools to find efficiently good approximations of h^* [6].

3 New Methods to Solve Decomposed CSPs

3.1 Solving VS Graphical Decompositions

In practice, decomposition-based methods are not generally used as assumed in the hierarchy. The steps 1 and 2 are generally merged, the computing of a decomposition whose width is bounded allowed to conclude that the problem width is bounded. Moreover, this decomposition is computed thanks to heuristic algorithms likewise in TC-1989 instead of exact ones which are too expensive. Note that from a practical viewpoint, optimal values w^* or h^* are not required and can be not desirable [13]. Moreover, steps 3 and 4 can be separated as for TC or MHD, or limited to one step as for BTD. So, in this paper, we consider that a decomposition-based method denoted DM_{DEC} has as input a CSP P and a graphical decomposition (denoted *DEC*) of P. Then the decomposition-based method *DM* solves P using the considered graphical decomposition *DEC*. For example, TC_{TD} denotes the applying of TC to a CSP P considering an optimal TD while $TC_{MCS(TD)}$ considers a TD found by MCS and solves it using TC. Thus $TC_{MCS(TD)}$ corresponds to TC-1989 while TC_{TD} is then TC referenced in the hierarchy considering an optimal TD. Finally, MHD_{HD} corresponds to MHD-1999. In this paper we show the interest to consider different steps in introducing combined versions of methods $DM \in \{TC, MHD, BTD\}$ and graphical decompositions as TD or HD, with optimal or non-optimal decompositions. We must show now how methods such as TC or BTD, which have been defined to run on tree-decompositions can be extended using hypertree-decompositions.

3.2 From HD to TD and from TD to (G)HD

Let $\mathcal{P} = (X, D, C, R)$ be a CSP and $HD = (T, \chi, \lambda)$ a hypertree-decomposition of $H = (X, C)$ whose width is h. (T, χ) verifies all the conditions required to be a tree-decomposition except that there can exist a cluster $\chi(p)$ contained in another $\chi(p')$, with $p, p' \in vertices(T)$. To compute a tree-decomposition from (T, χ), it is sufficient to merge each cluster $\chi(p)$ in one $\chi(p')$ containing it. The edges in T joining $\chi(p)$ to other nodes will join them to $\chi(p')$. Thereby, we obtain a tree-decomposition $TD(HD)$.

Finally, since HD defines a cover of the sets $\chi(p), \forall p \in vertices(T)$ whose size is at most h, a minimum cover of each cluster in $TD(HD)$ is also at most h. In Figure 4, we have a TD which is not optimal ($w = 12$) induced by the optimal HD in Figure 2.

Conversely, given a tree-decomposition of $TD = (E, T)$ of $H = (X, C)$ we could try to compute a hypertree-decomposition. While compute a hypertree-decomposition is not immediate, we can easily obtain a generalized hypertree-decomposition (GHD [8]).

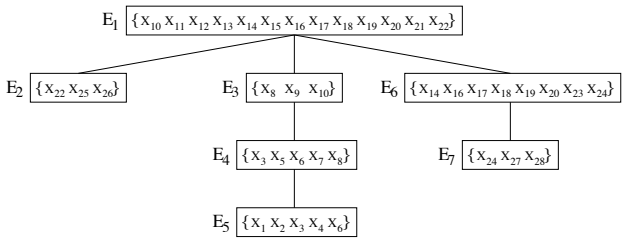

Figure 4. Induced tree-decomposition.

Definition 3 *Given a hypergraph $H = (X, C)$, a generalized hypertree-decomposition of H is a hypertree $GHD = (T, \chi, \lambda)$ for H which satisfies the following conditions:*

- *(i) for each edge $c \in C, \exists p \in vertices(T)$ s.t. $c \subset \chi(p)$,*
- *(ii) for each vertex $x \in X$, the set $\{p \in vertices(T) : x \in \chi(p)\}$ induces a (connected) subtree of T,*
- *(iii) for each $p \in vertices(T), \chi(p) \subset \cup_{c \in \lambda(p)} c$. An edge $c \in C$ is strongly covered in GHD if there exists $p \in vertices(T)$ such that $c \subset \chi(p)$ and $c \in \lambda(p)$.*

A generalized hypertree-decomposition GHD is a complete decomposition of H if every edge of H is strongly covered in GHD. The width gh of a generalized hypertree-decomposition $GHD = (T, \chi, \lambda)$ is $max_{p \in vertices(T)} |\lambda(p)|$. The generalized hypertree-width gh^* of H is the minimum width over all its generalized hypertree-decompositions.

To compute a GHD from the given TD, it is sufficient to cover all the clusters. This is not the case for the hypertree-decomposition since there is an additional condition (the fourth) that must be verified. This condition is in the definition of hypertree-decomposition only to make sure the existence of an algorithm that can compute in a polynomial time a HD whose hypertree-width is bounded by a constant if any. In this context, this aim has no interest. Since the TD is given, the best induced HD is the one whose cluster coverings are minimum. Thus, we will not take in account this condition and will compute a GHD. We associate a minimum cover $\lambda(p)$ to each cluster $E_p \in E$ with $p \in vertices(T)$. Thus, we obtain a generalized hypertree-decomposition $GHD(TD)$. Nevertheless, this GHD is not necessarily complete. To make it complete, we need to add for each hyperedge c which is not strongly covered a child to a node p such that $c \subset E_p$. Let $GHD_c(TD)$ be the obtained complete generalized hypertree-decomposition. $GHD(TD)$ and $GHD_c(TD)$ have the same width. The GHD induced by the TD in Figure 1 is easily obtained, and its width is

4, because the maximum size of $\lambda(p)$ is 4 with the node associated to E_6 ($\{C_1, C_2, C_3, C_4\}$ is this minimum cover). Moreover, if we consider gh^*, the width of a GHD of a given hypergraph, we know that $gh^* \leq h^* \leq 3.gh^* + 1$ ([1]). From this result, there is a hypertree-decomposition $HD_c(TD)$ of width at most $3.k + 1$.

3.3 Combined methods

Combined methods. It is now possible to exploit hypertree-decomposition with methods such as TC or BTD which have been designed to run on tree-decompositions. Before, we will define a new extension of TC (denoted *TC-2009*) more adapted to CSP defined with non-binary constraints. Given a CSP and one tree-decomposition $TD = (E, T)$, as in TC-1989, the subproblem associated to a cluster E_i is defined by the same set of variables E_i. But now, the set of constraints 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 $R_{E_i} = \{r_j[c_j \cap E_i] : c_j \in C_{E_i}\}$. By the same way, we can define *BTD-2009* too. The motivation to consider partial constraints can easily be understood by a simple example. Consider a CSP with three constraints defined on $3(n + 1)$ variables: $C_1 = \{x_0, x_1, \ldots x_n, y_1, \ldots y_n\}, C_2 = \{y_0, y_1, \ldots y_n, z_1, \ldots z_n\}$ and $C_3 = \{z_0, z_1, \ldots z_n, x_1, \ldots x_n\}$. The applying of TC induces 4 clusters, one per constraint, plus the "central" cluster $\{x_1, \ldots x_n, y_1, \ldots y_n, z_1, \ldots z_n\}$. The basic definition of TC does not consider partial constraints for the "central" cluster. So, the cost for solving it will be related to its number of variables. It will be m^{3n} while if we consider partial constraints, this cost will be limited to r^2. As for TC-1989, we have 3 steps. The first one computes a TD of the constraint network, using a decomposition algorithm for (hyper)graphs while the other steps (steps 2 and 3 presented in section 2) are not modified. So, this step is now parametrized by any graphical decomposition *DEC*. If we consider an optimal hypertree-decomposition HD, we define then *TC-2009*$_{HD}$, that is TC running on a tree-decomposition $TD(HD)$ induced by HD and considering as subproblems, the clusters of variables and the constraints whose scope intersects clusters. Likewise, we can define a large and non-exhaustive collection of methods:

- *TC*$_{MCS(TD)}$ =*TC-1999*: TC with MCS to find TD
- *BTD*$_{MCS(TD)}$: BTD with MCS to find TD
- *TC*$_{HMIN(TD)}$: TC with heuristic HMIN to find TD (minimization of w)
- *BTD*$_{HMIN(TD)}$: BTD with heuristic HMIN to find TD (minimization of w)
- *TC*$_{TD}$: TC with optimal TD (optimility: $w = w^*$)
- *BTD*$_{TD}$: BTD with optimal TD (optimility: $w = w^*$)
- *TC-2009*$_{HD}$: *TC-2009* with TD induced by an optimal HD (optimility: $h = h^*$)

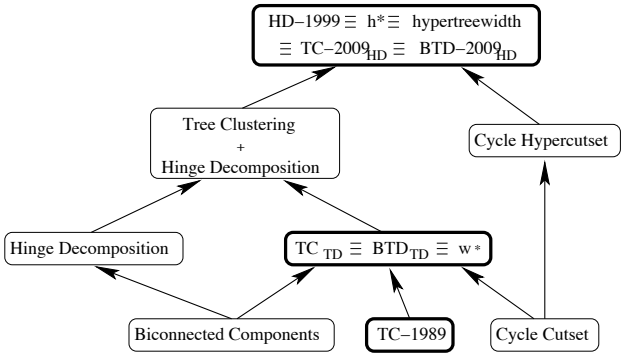

Figure 5. The Constraint Tractability Hierarchy revisited

- *BTD-2009*$_{HD}$: *BTD-2009* with TD induced by an optimal HD (optimility: $h = h^*$)
- *TC-2009*$_{HMIN(HD)}$: *TC-2009* with TD induced by a HD computed by a heuristic (minimization of h)
- *BTD-2009*$_{HMIN(HD)}$: *BTD-2009* with TD induced by a HD computed by a heuristic (minimization of h)

Moreover, we can consider more complex heuristics which consider simultaneously several criteria. For example, we can define heuristics with a minimization of w and then a maximization of constraints per clusters. Then, we define heuristics such as $HMIN(TD)+HMAX(HD)$ allowing to define methods as *BTD*$_{HMIN(TD)+HMAX(HD)}$, that is BTD with TD computed using a heuristic to minimize the number of variables per clusters, and then, to maximize the number of constraints in clusters.

For the analysis of the complexity of *TC-2009*$_{HD}$, assume that the width of HD is h^*. Each sub-problem (cluster) is solved independently (step 2) using an algorithm as nFC2. Thanks to the result presented in [12] recalled in section 2, 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 $k = max\{k_i : i \in I\}$. Moreover, we have $k \leq h$. So, we can now establish that:

Theorem 1 *The time cost of TC-2009*$_{HD}$ *is* $O(S.r^{h^*})$.

Note that this result holds for BTD-2009$_{HD}$ too. This result allows us to give a more precise presentation of the Constraint Tractability (Figure 5). Hierarchy since TC-2009$_{HD}$ and BTD-2009$_{HD}$ are at the same (top) level of this hierarchy. It proves that TC-2009$_{HD}$ performs at least as well as MHD-1999. Precisely, the time complexity of TC-2009$_{HD}$ and then of BTD-2009$_{HD}$ are the same as for MHD-1999.

	TC-1999		TC		BTD		MHD-1999	
MCS	m^{w+1}	m^{w+1}	m^{w+1}	m^{w+1}	m^{w+1}	m^s	UD	
HMIN(TD)	UD		m^{w+1}	m^{w+1}	m^{w+1}	m^s	UD	
HMIN(HD)	UD		$\min(m^{w+1}, r^h)$	m^{w+1}	$\min(m^{w+1}, r^h)$	m^s	UD	
TD	UD		m^{w^*+1}	m^{w^*+1}	m^{w^*+1}	m^s	UD	
HD	UD		$\min(m^{w+1}, r^{h^*})$	m^{w+1}	$\min(m^{w+1}, r^{h^*})$	m^s	r^{h^*}	r^{h^*}

	TC-2009		BTD-2009	
MCS	$\min(m^{w+1}, r^h)$	m^{w+1}	$\min(m^{w+1}, r^h)$	m^s
HMIN(TD)	$\min(m^{w+1}, r^h)$	$\min(m^{w+1}, r^h)$	$\min(m^{w+1}, r^h)$	$\min(m^s, r^h)$
HMIN(HD)	$\min(m^{w+1}, r^h)$	$\min(m^{w+1}, r^h)$	$\min(m^{w+1}, r^h)$	$\min(m^s, r^h)$
TD	$\min(m^{w^*+1}, r^h)$	$\min(m^{w^*+1}, r^h)$	$\min(m^{w^*+1}, r^h)$	$\min(m^s, r^h)$
HD	$\min(m^{w+1}, r^{h^*})$	$\min(m^{w+1}, r^{h^*})$	$\min(m^{w+1}, r^{h^*})$	$\min(m^s, r^{h^*})$

Table 1. Time and space complexities for hybrid methods.

The second consequence is to have now an implementation of MHD with BTD-2009$_{HD}$ which inherits of the same time complexity as MHD-1999 while limiting drastically the space complexity and then allowing a potential practical efficiency. From a practical viewpoint, this fact can be really significant to get efficient implementations of decomposition-based methods to solve real instances of CSPs.

Finally, Table 1 presents the time complexity and the space complexity of several combined methods. In this table, s is the maximum size of the intersections between clusters. In this table, lines represent the considered (hyper)graphical decompositions while columns are related to the solving method. E.g. in the square corresponding the line TD and the column BTD, we have m^{w^*} and m^s which are respectively the time complexity and the space complexity of BTD running on an optimal TD, that is the complexities of BTD$_{TD}$. Note that if a method (e.g. TC-1989) has not been defined to run on a particular decomposition (TD for TC-1989), this will be denoted UD for undefined.

4 Experiments

In this section, we run experiments to evaluate the practical interest of the methods defined previously to check consistency of instances.

The available implementations of TC and MHD do not succeed in solving these instances because of a huge amount of memory and time they require to solve separately the subproblems in a decomposition. Then, we use BTD which has already shown its effectiveness on structured CSPs.

Moreover, computing an optimal (hyper)tree-decomposition is an NP-hard problem. The runtime of exact techniques is too large. Yet, there is no guarantee on the practical efficiency using these decompositions. So, we prefer heuristics with better time complexity to compute our decompositions. Thus, we will not consider BTD$_{HD}$ and BTD$_{TD}$. In [9], the *Bucket Elimination for Hypertree* ([6]) is evaluated as the best technique for computing hypertree-decompositions within a reasonable amount of time. Its implementation is available at www.dbai.tuwien.ac.at/proj/hypertree/downloads.html. So, we will consider the method *BTD-2009*$_{BE(HD)}$. However, in each cluster, this method takes in account all the constraints intersecting it. This approach is different from the one in MHD where, for solving a cluster, only the constraints given by the HD are considered. So, we decide to define an extension of *BTD*$_{BE(HD)}$, *BTD-HD*$_{BE(HD)}$ which exploits totally the HD likewise in MHD. It guarantees the same time complexity bounds than MHD ones. The Minfill triangulation algorithm (MF) and the MCS one are known to give very good tree-decompositions w.r.t. the practical solving of structured problems. Thus, we will also consider the methods *BTD-2009*$_{MF(TD)}$ and *BTD-2009*$_{MCS(TD)}$. For a complete comparison, we evaluate an enumerative solving technique (namely *FC*) and an efficient method which combines several solvers and has won the CP'2008 competition (*Hydra_k_10*).

The experiments are run on a Linux based PC with Pentium IV 3.2GHz and 1GB of RAM. We first consider the random structured CSP presented in [11]. Each CSP class has several parameters (n, d, w, t, s, ns, p). An instance is defined by n variables whose domain size is d. Its constraint graph is a clique-tree with ns nodes whose size is at most $w + 1$ and whose separator size is bounded by s. t is the number of forbidden tuples for each constraint. p is the percentage of constraints removed from the instance built by this way. The results presented in the Table 2 for each class is the average on 30 instances. For each instance the time out (TO) is fixed to 900s. MO means that the method runs out of memory for one instance (the memory needed to solve the problem exceeds 1GB). The poorest results are obtained by FC which does not take profit of the structural properties to enhance the solving. *Hydra_k_10* improves slightly these performances. *BTD-HD*$_{BE(HD)}$ runs out of memory in the solving of several instance classes.

CSP (n, d, w, t, s, ns, p)	FC	$Hydra$	*BTD-09*$_{MCS(TD)}$		*BTD-09*$_{MF(TD)}$		*BTD-HD*$_{BE(HD)}$			*BTD-09*$_{BE(HD)}$		
	time	time	time	w	time	w	time	w	h	time	w	h
(150, 25, 15, 215, 5, 15, 10)	275.64	122.46	4.21	13	2.56	12.5	1.13	13	7	1.85	13	7
(150, 25, 15, 237, 5, 15, 20)	398.66	92.56	1.64	12.7	2.62	11.7	21.11	12.1	6.7	4.99	12.1	6.7
(150, 25, 15, 257, 5, 15, 30)	219.97	30.76	0.95	12	1.64	10.8	MO	10.5	6	9.72	10.5	6
(150, 25, 15, 285, 5, 15, 40)	259.71	12.46	1.19	11.5	2.49	10	MO	9.6	5.9	33.46	9.6	5.9
(250, 20, 20, 107, 5, 20, 10)	719.28	568.42	12.13	18	49.91	17	222.95	17	9	155.20	17	9
(250, 20, 20, 117, 5, 20, 20)	642.08	394.50	5.56	17.3	24.09	15.9	MO	15.7	8.8	73.49	15.7	8.8
(250, 20, 20, 129, 5, 20, 30)	548.50	195.94	8.24	16.7	16.30	14.9	MO	13.9	8	75.47	13.9	8
(250, 20, 20, 146, 5, 20, 40)	590.84	197.70	15.20	16	45.76	13.9	MO	12.6	7.8	139.60	12.6	7.8
(250, 25, 15, 211, 5, 25, 10)	429.86	393.17	8.32	13	19.74	12.6	5.16	13	7	6.66	13	7
(250, 25, 15, 230, 5, 25, 20)	608.56	351.28	6.13	12.7	21.06	11.9	7.60	12.7	6.9	7.22	12.7	6.9
(250, 25, 15, 253, 5, 25, 30)	484.56	217.85	7.75	12.1	49.41	11	211.93	11	6.1	131.01	11	6.1
(250, 25, 15, 280, 5, 25, 40)	498.94	80.82	5.26	11.7	43.44	10.1	MO	9.9	5.9	61.06	9.9	5.9
(250, 20, 20, 99, 10, 25, 10)	529.61	584.41	105.51	17.9	147.49	17	MO	15.2	9	205.32	15.2	9
(500, 20, 15, 123, 5, 50, 10)	613.11	418.22	7.69	13	43.69	12.9	4.07	13	7	6.54	13	7
(500, 20, 15, 136, 5, 50, 20)	583.69	394.27	6.33	12.8	122.65	12	4.44	12.9	6.96	5.30	12.9	6.96

Table 2. Runtimes (in s) and decomposition parameters on random structured CSPs (*BTD-09* stands for *BTD-2009*).

	$BTD-09_{MCS(TD)}$		$BTD-09_{MF(TD)}$		$BTD-HD_{BE(HD)}$			$BTD-09_{BE(HD)}$		
bench	time	w	time	w	time	w	h	time	w	h
ren-3	11.15	12	10.67	10	42.34	12	3	20.56	12	3
ren-6	3.75	13	3.71	10	1279.55	13	3	2.70	13	3
ren-12	11.85	12	11.64	10	134.24	10	3	10.49	10	3
ren-16	10.36	12	6.04	10	3.65	13	3	6.43	13	3
ren-17	5.42	12	9.59	10	145.06	11	3	3.41	11	3
ren-18	12.09	11	12.23	11	39.34	12	3	10.46	12	3
ren-19	12.07	11	7.74	9	49.25	11	3	16.36	11	3
ren-23	2.81	11	12.87	9	28.82	12	4	3.79	12	4
ren-24	8.05	14	7.97	11	35.01	12	4	7.63	12	4
ren-30	9.81	13	3.80	10	202.05	11	3	8.25	11	3
ren-35	12.28	12	7.32	10	57.33	11	3	13.19	11	3
ren-36	1.78	13	3.80	11	225.76	13	4	4.88	13	4
ren-37	17.01	15	13.68	12	13.64	14	4	21.16	14	4
ren-39	35.55	16	13.45	12	746.24	12	5	1.79	12	5
ren-40	8.60	14	5.86	11	65.55	12	4	9.01	12	4
ren-42	3.44	15	2.48	12	TO	12	3	2.50	12	3
ren-47	21.31	14	53.71	12	324.20	12	5	80.25	12	5

Table 3. Method runtimes (in s) and decomposition parameters on CSP instances modifiedRenault/normalized-renault-mod-* from benchmarks of CP'2008 competition.

This large amount of memory space required is due to the important number of solutions in the clusters which leads to the recording of numerous (no)goods. Although the computed HD has a good tree-width, the reduced number of constraints in the clusters makes them very difficult to solve with a large number of solutions. If we consider *BTD-2009*$_{BE(HD)}$, the performances are far better. There are less recorded (no)goods since the clusters are more constrained and contain less solutions. Nevertheless, the best results are obtained by BTD based on tree-decomposition. The tree-decompositions computed by MCS have a greater width than those computed by Minfill. However, in practice, the MCS ones are more effective. They allow a better dynamic ordering of variables. We observe on these benchmarks that computing a tree-decomposition with many constraints in the clusters is more efficient in practice.

To confirm this observation on decomposition-based methods, we consider CSP instances from the CP'2008 competition. For each problem, the Time out (TO) is fixed to 1800s. The results presented in the Table 3 are on the modifiedRenault class instances. But, we do the same observations on some other classes like geom. As expected, *BTD-HD*$_{BE(HD)}$ has a very bad behavior. It fails in solving many instances (TO or MO). Its mean runtime is greater than 248s for the class modifiedRenault reported. The subproblems in a HD remain very hard to solve in practice because of the reduced number of constraints considered. This small constraint number weakens the power of filtering techniques which contribute a lot in the efficiency of enumerative methods. *BTD-2009*$_{MCS(TD)}$ and *BTD-2009*$_{MF(TD)}$ behave far better thanks to a maximum number of constraints in the clusters which are easier to

solve. Furthermore, *BTD-2009*$_{BE(HD)}$ outperforms drastically *BTD-HD*$_{BE(HD)}$ because it takes in account all possible constraints. Nevertheless, these results are still worse than those of *BTD-2009*$_{MF(TD)}$. Note that, while FC fails in solving nearly all the modifiedRenault class CSP, $Hydra\_k\_10$, as expected, outperforms the others methods on these problems.

To summarize, we note first that computing a tree-decomposition with a small width is more relevant in practice than computing a good HD. Indeed, this tree-decomposition provides better results in practice as well as good time complexity bounds w.r.t. the clusters size and its minimum cover. Furthermore, its clusters can be solved more efficiently when they contain many constraints. The fact is justified considering that the more a problem is constrained, the more it will be easy to solve (this can be easily justified by probabilistic arguments). Hence, the most promising approach consists in computing a TD with very constrained clusters.

5 Discussion and Conclusion

We have proposed new approaches based on combined decompositions of CSPs. Precisely, we have introduced two new optimal methods, TC-2009 and BTD-2009, which exploit hypertree-decomposition and TC or BTD for solving problems. This approach allows to get better complexity bounds while inheriting of practical efficiency of enumerative methods like nFC2, one of the most powerful techniques for solving CSP. We have then also enriched the Constraint tractability hierarchy by updating the top of the hierarchy using methods based on tree-decomposition as TC or BTD (TC-2009 and BTD-2009). Finally, we have obtained experimental results that show the interest of our approach.

Our results differ from the one given in [3] (Theorem 7.28, page 231). In this work, Dechter proposes to solve a tree-decomposition of a CSP in time r^{hw} where hw is obtained by the sum of the induced hypertree-width h and the number of variables in cluster which are not covered by cluster constraints. For TC-2009 and BTD-2009, complexity is now limited to r^h (where h is now the induced generalized hypertree-width). Our results also differ from the paper in [4] which demonstrates empirically that the bounding power of the tree-width is often superior in practice to the hypertree-width in probabilistic or deterministic networks. In the technical report extending this paper, it is said that the And/Or Search Graph approach can guarantee a time complexity bound depending on the width of a given hypertree-decomposition. But, this method only considers a subclass of the possible hypertrees meaning that its complexity is weaker than the MHD one. Yet, TC-2009 and BTD-2009 have a time complexity depending on the width of a generalized hypertree-decomposition. Thus, their complexity bound is at least equivalent to MHD one.

A natural continuation of this work could be related to the study of graphical decompositions which combine optimization of parameters as w (minimizing) and h (maximizing). Moreover, it seems natural to extend this analysis to COP (optimization) or probabilistic graphical models such as in [4] (and [15]).

References

[1] I. Adler, G. Gottlob, and M. Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007.

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

[3] R. Dechter. *Tractable Structures for Constraint Satisfaction Problems*, pages 209–244. Chapter in the *Handbook of Constraint Programming* F. Rossi, T. Walsh and P. van Beek, 2006.

[4] R. Dechter, L. Otten, and R. Marinescu. On the Practical Significance of Hypertree vs. Tree Width. In *ECAI*, pages 913–914, 2008.

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

[6] A. Dermaku, T. Ganzow, G. Gottlob, B. MacMahan, N. Musliu, and M. Samer. Heuristic Methods for Hypertree Decompositions. In *MICAI 2008*, pages 1–11, 2008.

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

[8] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences (JCSS)*, 66(4):775–808, 2003.

[9] G. Gottlob and M. Samer. A backtracking-based algorithm for hypertree decomposition. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[10] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[11] P. Jégou, S. Ndiaye, and C. Terrioux. Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs. In *Proc. of IJCAI*, pages 112–117, 2007.

[12] P. Jégou, S. Ndiaye, and C. Terrioux. A New Evaluation of Forward Checking and its Consequences on Efficiency of Tools for Decomposition of CSPs. In *ICTAI*, pages 486–490, 2008.

[13] P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *CP*, pages 777–781, 2005.

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

[15] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166:165–193, 2005.

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

[17] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3):566–579, 1984.