A New Method for Computing Suitable Tree-decompositions with respect to Structured CSP Solving

Cédric Pinto
LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
cedric.pinto@etu.univ-cezanne.fr

Cyril Terrioux
LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
cyril.terrioux@univ-cezanne.fr

Abstract

The tree-decomposition notion [15] plays a central role in the frame of the structured CSP solving. On the one hand, it is exploited in many methods like Tree-Clustering [5], BTD [11] or Cyclic-Clustering (CC [9]). It then leads to theoretical time complexity bounds among the best ones. On the other hand, it is often used as a preliminary step for computing a hypertree-decomposition. Unfortunately, finding the best tree-decomposition is a NP-hard problem. So heuristic methods are classically used for computing tree-decompositions. They mostly rely on triangulations of graphs. Sometimes, this approach by triangulation can lead to a rough identification of the structure. Such a drawback can be avoided by considering a cutset such that the remaining problem corresponds to a set of tree-decompositions. In this article, from a cutset and the corresponding set of tree-decompositions, we propose a new method for computing a suitable tree-decomposition w.r.t. CSP solving. Thanks to this approach, we can exploit BTD on the resulting tree-decomposition instead of CC on the cutset and the corresponding set of tree-decompositions. Then, unlike CC, it allows to fully exploit the informations recorded during the search, what leads to avoid some redundancies in the search space. In practice, the first empirical results are very promising since BTD with a tree-decomposition computed with the proposed method outperforms BTD with a tree-decomposition based on triangulation and some other classical algorithms.

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 (e.g. graph coloring, planning, frequency assignment problems, ...). A *finite constraint satisfaction problem* (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. A constraint $c_i \in C$ on an ordered subset of variables, $c_i = (x_{i_1}, \dots x_{i_{a_i}})$ 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. Let $Y = \{x_1, \dots x_k\}$ be a subset of X. An *assignment* $\mathcal{A}$ is a tuple $(v_1, \dots, v_k)$. An assignment $\mathcal{A}$ on Y satisfies a constraint $c \in C$ s.t. $c \subseteq Y$ if $\mathcal{A}[c] \in r_c$ with $\mathcal{A}[c]$ the restriction of $\mathcal{A}$ to the variables involved in c. $\mathcal{A}$ is said *consistent* if it satisfies each constraint $c \subseteq Y$. A solution is an assignment of each variable which satisfies all the constraints. Determining if a solution exists is a NP-complete problem. In the following, for sake of simplicity, we only consider binary CSPs (i.e. CSPs whose each constraint involves exactly two variables). Of course, this work can be extended to non-binary CSPs (see [14] for more details).

The usual methods for solving CSPs (e.g. Forward Checking [8] or MAC [16]) are based on backtracking search. This approach, often efficient in practice, has an exponential theoretical time complexity in $O(m.d^n)$ (denoted $O(exp(n))$) for an instance having n variables and m constraints and whose largest domain has d values. Several works have been developed to improve this theoretical complexity bound thanks to particular features of the instance. Generally, they exploit some structural properties of the CSP. The structure of a CSP (X, D, C, R) can be represented by the graph (X, C), called the *constraint graph*. In this context, the tree-decomposition notion [15] plays a central role. A *tree-decomposition* of a graph $G = (X, C)$ 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 $\{x, y\} \in C$, there exists $i \in I$ with $\{x, y\} \subseteq E_i$, and (iii) for all $i, j, k \in I$, if k is in a path from i to j in T, then $E_i \cap E_j \subseteq E_k$. 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. On the one hand, the tree-width leads to one of the best known theoretical time complexity bounds, namely $O(exp(w^* + 1))$ with w^* the tree-width. Different methods (e.g. [5, 11]) have been proposed to reach this bound. They aim to cluster variables s.t. the cluster arrangement is a tree. On the other hand, tree-decompositions are also exploited in other structural methods. For instance, methods relying on a hypertree-decomposition of the constraint graph [6] often require the computation of a tree-decomposition in order to compute a hypertree-decomposition. Likewise, the Cyclic-Clustering method [9] exploits a tree-decomposition. More precisely, it assigns a subset of variables (called a cutset) s.t. the constraint graph of the problem reduced to the unassigned variables is a set of clique trees (which so corresponds to a tree-decomposition).

From a theoretical viewpoint, reach the best theoretical complexity bound requires to compute an optimal tree-decomposition (i.e. a tree-decomposition with a minimum width), which is a NP-hard problem [1]. In practice, it is clear that it is not reasonable to solve a NP-hard problem as a preliminary step of the solving of a NP-complete problem. So heuristic methods are generally used. In most cases, they rely on triangulated graphs. A graph is triangulated if it has no cycle of length greater than 3 without a chord (i.e. an edge between two non consecutive vertices in a cycle). Indeed, from a triangulated graph, we can compute in linear time an optimal tree-decomposition of this graph (each maximal clique of a triangulated graph corresponds to a cluster of an associated optimal tree-decomposition). As constraint graphs are generally not triangulated, a triangulated graph is computed from the initial constraint graph by adding edges s.t. the resulting graph is triangulated. This operation is called a triangulation. As finding the best triangulation (w.r.t. the number of added edge) is NP-Hard, methods based on heuristic triangulation are generally used and so we have no guarantee that they add the minimal number of edges. So, even if some studies have been performed from a graphical viewpoint or from a CSP solving viewpoint [10], in some cases, a heuristic triangulation may add a lot of edges and, by so doing, not succeed in identifying precisely the problem structure.

An alternative way to exploit the structure of a graph relies on the cutset notion. A subset V of vertices is a *cutset* of a graph (X, C) if the graph $(X - V, \{\{x, y\} \in C \; s.t. \; x, y \in X - V\})$ induced by V has a particular topological property. For instance, among all the possible kinds of cutset, cycle-cutsets are well known. They are defined s.t. the graph induced by V is acyclic. In our study, we focus on cutsets s.t. the induced graph is triangulated (i.e. it corresponds to a set of tree-decompositions). For instance, figure 1 presents a graph having 20 vertices. The set $\{y_1, y_2\}$ forms a cutset of this graph s.t. the induced graph involving the vertices $x_1, \ldots, x_{18}$ is triangulated. This alternative solution requires the use of a method like Cyclic-Clustering. Such an approach allows to stay closer to the initial graph than one based on a triangulation and so to better exploit the structure of the constraint graph. Unfortunately, the Cyclic-Clustering approach has some limits. For instance, informations recorded during the search are not fully exploited to avoid redundant parts of the search space.

In this paper, we propose a new method for computing a tree-decomposition. Like in the Cyclic-Clustering approach, it relies on a triangulated subgraph and a cutset. From the triangulated subgraph, we can compute a set of tree-decompositions (a tree-decomposition by connected component of the triangulated subgraph). These tree-decompositions are then merged with the cutset in order to form a single tree-decomposition. The first main advantage of this approach is that a triangulation step is not required and so we can hope to better exploit the structure of the initial constraint graph. Moreover, this method allows then to solve the problem thanks to the BTD method [11] (which is one of the most efficient structural methods). Finally, by so doing, we record and exploit more informations during the solving and so more redundancies can be avoided in practice.

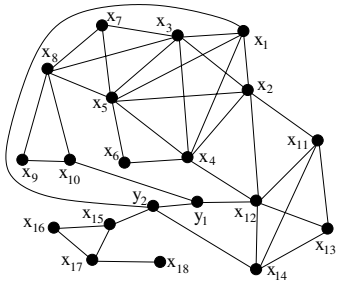


Figure 1. A graph with 20 vertices.

2 A new approach to compute tree-decompositions: the Merging

The BTD algorithm [11] records and exploits many informations in order to speed up the CSP solving. The Cyclic-Clustering method (and more precisely CC-BTD$_i$ [12]) executes potentially many calls to BTD. However, it does not reuse the recorded informations during these calls.

So, we propose a new method able to exploit such informations: the Merging method.

In a similar way of the first step of CC, we compute a TIS from which we construct an associated set of tree-decompositions. The vertices which do not belong to these tree-decompositions form the associated cutset. So, this structure is well-adapted to the Cyclic-Clustering algorithm. Nonetheless, instead of running CC, we propose to integrate the cutset in the set of tree-decompositions in order to obtain a unique tree-decomposition. Then, after these operations, we can run the BTD algorithm.

From a cutset and a set of tree-decompositions, definition 1 describes the construction of a single tree-decomposition.

Definition 1 *Let $TD = \{TD_1, ..., TD_p\}$ be a set of tree-decompositions generated from a triangulated subgraph where each $TD_j = (E_j, \mathcal{T}_j)$ with $\mathcal{T}_j = (I_j, A_j)$ and V a cutset. $TD^M = (E^M, \mathcal{T}^M)$ with $\mathcal{T}^M = (I^M, A^M)$ is the decomposition obtained after Merging operations among each TD_j and V defined as follows:*

- *Let E_0^M be the root of TD^M, $E_0^M = \{y \in V\}$.*
- *$\forall TD_j \in TD$, $\forall E_{j_i} \in E_j$, we construct E_k^M s.t. $E_k^M = E_{j_i} \cup \{y \in V | \exists \{x, y\} \in C$ with $x \in Desc(E_{j_i})\}$.*
- *We connect E_0^M with each root E_{j_0} of TD.*

Figure 2 provides an example of merging. In figure 2(a), we can see a cutset and a TIS of the graph in figure 1. In figure 2(b), we add the cutset's variables in the tree-decompositions associated to the TIS. In figure 2(c), we connect the cluster containing the cutset with every root of the initial tree-decompositions.

Theorem 1 proves that a decomposition constructed in definition 1 is a valid tree-decomposition (the proofs of the following theorems are available in [14]).

Theorem 1 *TD^M is a valid tree-decomposition.*

The Merging algorithm is presented in algorithms 1-3. The first step (lines 1-3 of algorithm 1) adds each variable x_i of the cutset V into a cluster E_{j_i} of the tree-decomposition TD_j if there exists at least one constraint between x_i and a variable belonging to E_{j_i}. The function $AddVariable$ consists in adding this variable x_i into each cluster located on the branch between the root E_{j_0} of TD_j and E_{j_i}. The second step (line 4 of algorithm 1) constructs a new cluster E_0^M which contains the whole cutset. The final step (lines 5-6 of algorithm 1) defines this cluster E_0^M as root of the built tree-decomposition. We can observe that the second step is necessary to respect the third condition of the tree-decomposition definition.

We will denote n the number of variables, m the number of constraints, d the size of the largest domain, k the size of cutset, $w + 1$ the size of the largest cluster of the set of tree-decompositions, s the size of the largest separator and p the maximal height of the set of tree-decompositions, i.e. the maximum number of clusters between a root and a leaf in a tree-decomposition.

Algorithm 1: Merging()

```
1 forall x ∈ cutset do
2     forall {x, y} ∈ C with y ∈ E_{j_i} and E_{j_i} belongs to a
      tree-decomposition TD_j associated to TIS do
3         AddVariable(x, *E_{j_i})
4 E_0^M ← {x ∈ cutset}
5 forall Tree-decomposition TD_j do
6     ChooseRoot(*TD_j, *E_0^M)
```

Algorithm 2: AddVariable($x, *E_{j_i}$)

```
1 if x ∈ E_{j_i} then return
2 else if E_{j_i} = E_{j_0} then return
3 else
4     E_{j_i} ← E_{j_i} ∪ {x}
5     AddVariable(x, *Father(E_{j_i}))
6 return
```

Algorithm 3: ChooseRoot($*TD_j, *E_0^M$)

```
1 if E_{j_0} ∩ E_0^M = ∅ then return
2 if E_0^M ⊂ E_{j_0} then E_0^M ← E_{j_0}
3 else Father(E_{j_0}) ← E_0^M
```

Theorem 2 *The time complexities of $AddVariable$ and $ChooseRoot$ are respectively $O(p)$ and $O(w + k + 1)$.*

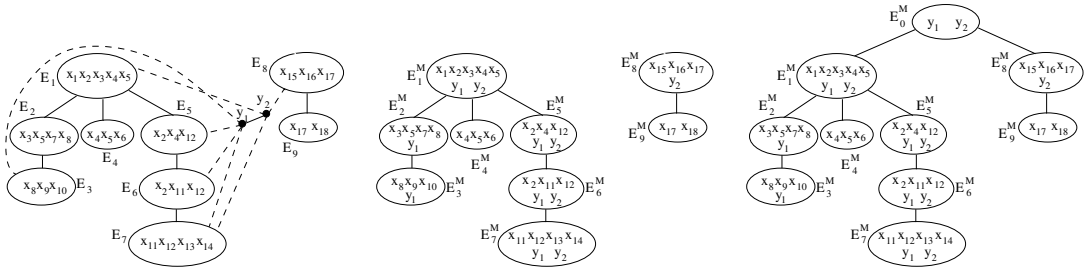
Theorem 3 *The time complexity of the Merging algorithm is $O(p(k + m_{kd}) + k + n(w + k + 1))$ with m_{kd} the number of constraints between the cutset and the set of tree-decompositions associated to TIS.*

Theorem 4 *Let TD^M be a tree-decomposition constructed with Merging algorithm. The time complexity of BTD executed on TD^M is $O(exp(w + k + 1))$ and its space complexity is $O(exp(s + k))$.*

3 Experimental results

In this section, we assess the quality of the tree-decompositions computed thanks to Merging method w.r.t. the CSP solving and we compare the obtained results with other classical solving algorithms.

The tests are performed on structured problems. More exactly, our generator of binary CSPs constructs a triangulated constraint graph. Then, it constructs an other graph



(a) A TIS and a cutset (b) Integration of the cutset (c) Cutset as root

Figure 2. Example of Merging

Classes $(\mathbf{n}, \mathbf{d}, \mathbf{r}, \mathbf{t}, \mathbf{s}, \mathbf{k}, \mathbf{e_1}, \mathbf{e_2})$	$\mathbf{S}_{max}$	CC-BTD		FC-BTD		
		CC-BTD$_1$	CC-BTD$_2$	Triang Triang	Merging without structure	Merging with structure
(50, 15, 15, 65, 5, 15, 80, 50)	best	$>_{15}$ 654.78	$>_5$ 292.63	10.14	1.34	**0.74**
	∞			16.94	Mem	**2.53**
(50, 15, 15, 65, 5, 20, 130, 50)	best	176.85	132.19	4.87	**0.95**	1.38
	∞			Mem	Mem	**2.54**
(50, 15, 15, 68, 5, 15, 80, 30)	best	$>_{17}$ 1010.61	$>_1$ 43.55	$>_1$ 98.03	$>_1$ 41.78	**0.80**
	∞			Mem	Mem	**1.45**
(50, 15, 20, 54, 5, 10, 35, 20)	best	$>_{25}$ 986.02	$>_9$ 412.72	$>_3$ 174.26	$>_{\mathbf{2}}$ **74.39**	$>_2$ 75.25
	∞			Mem	Mem	**1.45**

Table 1. Runtime (in seconds) for the four considered classes

which represents the cutset. Finally, it adds constraints between both graphs. We need 8 parameters to generate this kind of problems: n the number of variables of the triangulated graph, d the size of the largest domain, r the size of the largest clique of triangulated graph, t the number of forbidden tuples by the constraints, s the size of the largest separator, k the size of the cutset, e_1 the number of constraints into the cutset and e_2 the number of constraints between the cutset and the triangulated graph. So, a class of problems is defined by these 8 parameters: $(n, d, r, t, s, k, e_1, e_2)$. For each considered class, the number of consistent problems is approximately equal to the number of inconsistent problems. In our experiments, BTD exploits FC to solve the clusters. For ordering variables in FC or inside a cluster, we use the well-known *dom/deg* heuristic which first chooses the variable x_i which minimizes the ratio $\frac{|d_{x_i}|}{|\Gamma_{x_i}|}$ with d_{x_i} the current domain of x_i and Γ_{x_i} the set of the variables sharing a constraint with x_i. We compare BTD based on triangulation to compute the tree-decomposition and BTD using Merging. We also compare the obtained results with ones of CC-BTD$_1$, CC-BTD$_2$, and FC. However, FC turns to be unable to solve efficiently most of these problems, within the time limit (namely 30 minutes). We know that for efficiency reasons, CC-BTD and Merging method need a cutset with few solutions. Unfortunately, we do not have any method to recognize such a structure. So, as we aim to assess the interest of Merging method for structured problems and not the quality of the computed cutset, we provide cutset to Merging and CC-BTD. Moreover, as we want to measure the impact of the knowledge of the structure on the efficiency of the resolution, we experiment a second variant of Merging which uses the Balas and Yu's algorithm [2] to compute a MTIS, and so the rest of variables forms a cutset. The experimentations are performed on a linux-based PC with an Intel Pentium IV 3.2 GHz and 1 Gb of memory and the running times are expressed in seconds. For each class, we solve 50 instances and the presented results are then the averages of results obtained for each instance. The notation $>_i$ indicates that i instances are unsolved by the corresponding algorithm. In this case, as we do not know the real runtime, we add penalty of 30 minutes, for each unsolved instance. Finally, the notation Mem means that the memory space (1 Gb) is insufficient for at least one instance.

Thanks to the recorded informations during the resolution, BTD speeds up the CSP solving. Unfortunately, the number of recorded informations is exponential in the size of the largest separator. So, in order to limit the memory requirements of BTD, we transform the tree-decomposition s.t. its maximal separator be limited by a parameter S_{max}. For our tests, we try many values for S_{max}. In the results, S_{max} = *best* means that the indicated times are those obtained with the best value S_{max} for each algorithm. $S_{max} = \infty$ means that we do not limit the separator size.

First, we can note that if the parameter S_{max} is unlimited then the memory requirements of BTD is often too ex-

pensive except when the structure is known for the Merging method. As CC-BTD$_1$ and CC-BTD$_2$ know the structure of problems, it is not necessary to limit S_{max} since the largest separator of the set of tree-decompositions associated to TIS cannot be greater to the value of parameter s which is 5. We note that the main advantage of CC-BTD is the memory requirements. Unfortunately, CC-BTD gives bad runtimes and many problems are unsolved especially for CC-BTD$_1$. For all that, the additional recordings and reuses realized by CC-BTD$_2$ improve usually the runtime and the number of solved problems. As predicted, the knowledge of the structure is important for the efficiency of the solving. However, the difference takes place particularly when S_{max} is unlimited. Finally, we can observe that BTD using Merging, with or without knowledge of the structure, is better than BTD using triangulation to compute the tree-decomposition. In order to confirm this result, we consider the number of added edges by the triangulation with the number of constraints for each class. We observe that the number of added edges can sometimes be important (from 15% up to 45% of edges of the initial constraint graph). However, we note that the size of the largest cluster of the tree-decomposition obtained with triangulation and with Merging are approximately equal. This means that the difference of efficiency between Merging and triangulation mostly depends on the content of clusters.

To sum up, on the considered structured instances, the tree-decomposition computed thanks to Merging allows BTD to obtain significantly better results than ones it obtains when the tree-decomposition is based on a triangulation. This improvement concerns as well the required memory space as the runtime. Indeed, this method permits to better exploit the structure of problems.

4 Conclusions and future works

In this article, from a cutset and the corresponding set of tree-decompositions, we have proposed a new method for computing a suitable tree-decomposition w.r.t. CSP solving. Thanks to this approach, we can exploit BTD on the resulting tree-decomposition instead of CC or CC-BTD$_i$ on the cutset and the corresponding set of tree-decompositions. Then, unlike CC-BTD$_i$, we are now able to fully exploit the (no)goods recorded during the search, what leads to avoid some redundancies in the search space. In practice, the first empirical results are very promising since BTD with a tree-decomposition computed with the proposed method outperforms BTD with a tree-decomposition based on triangulation and some other classical algorithms.

This work raises the problem of finding good cutsets and/or TIS. Indeed, while a lot of works deal with triangulations, few works have been proposed about cutsets and TIS. Then, if the cutset and TIS considered here are static, it could be interesting to extend this work to dynamic cutset and/or TIS. Finally, this work can be extended to other domains exploiting tree-decompositions (e.g. in constraint optimization [17, 4, 13] or in relational databases [3, 7]).

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

References

[1] S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics*, 8:277–284, 1987.

[2] E. Balas and C. Yu. Finding a maximum clique in an arbitrary graph. *Siam J. on Computing*, 15(4):1054–1068, 1986.

[3] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30:479–513, 1983.

[4] 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.

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

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

[7] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

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

[9] P. Jégou. Cyclic-Clustering: a compromise between Tree-Clustering and the Cycle-Cutset method for improving search efficiency. In *Proc. of European Conference on Artificial Intelligence (ECAI-90)*, pages 369–371, 1990.

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

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

[12] P. Jégou and C. Terrioux. A Time-space Trade-off for Constraint Networks Decomposition. In *Proc. of ICTAI*, pages 234–239, 2004.

[13] A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, Novembre 1999.

[14] C. Pinto and C. Terrioux. A New Method for Computing Suitable Tree-decompositions with respect to Structured CSP Solving. Technical report, LSIS, 2008.

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

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

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