Tree-Decompositions with Connected Clusters for Solving Constraint Networks*

Philippe Jégou and Cyril Terrioux

Aix-Marseille Université, LSIS UMR 7296
13397 Marseille (France)
{philippe.jegou, cyril.terrioux}@lsis.org

Abstract. From a theoretical viewpoint, the (tree-)decomposition methods offer a good approach for solving Constraint Satisfaction Problems (CSPs) when their (tree)-width is small. In this case, they have often shown their practical interest. So, the literature (coming from Mathematics or AI) has concentrated its efforts on the minimization of a single parameter, the tree-width. Nevertheless, experimental studies have shown that this parameter is not always the most relevant to consider for solving CSPs. In this paper, we experimentally show that the decomposition algorithms of the state of the art produce clusters (a tree-decomposition is a tree of clusters) having several connected components. Then we highlight that such clusters create a real problem for the efficiency of solving methods. To avoid this kind of problem, we consider here a new kind of graph decomposition called *Bag-Connected Tree-Decomposition*, which considers only tree-decompositions such that each cluster is connected. We propose a first polynomial time algorithm to find such decompositions. Finally, we show experimentally that using these bag-connected tree-decompositions improves significantly the solving of CSPs by decomposition methods.

1 Introduction

Constraint Satisfaction Problems (CSPs, see [1] for a state of the art) provide an efficient way of formulating problems in computer science, especially in Artificial Intelligence.

Formally, a *constraint satisfaction problem* is a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of n variables, $D = (D_{x_1}, \ldots, D_{x_n})$ is a list of finite domains of values, one per variable, and $C = \{C_1, \ldots, C_e\}$ is a finite set of e constraints. Each constraint C_i is a pair $(S(C_i), R(C_i))$, where $S(C_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ is the *scope* of C_i, and $R(C_i) \subseteq D_{x_{i_1}} \times \cdots \times D_{x_{i_k}}$ is its *compatibility relation*. The *arity* of C_i is $|S(C_i)|$. A CSP is called *binary* if all constraints are of arity 2. The structure of a constraint network is represented by a hypergraph (which is a graph in the binary case), called the constraint (hyper)graph, whose vertices correspond to variables and edges to the constraint scopes. In this paper, for sake of simplicity, we only deal with the case of binary CSPs but this work can easily be extended to non-binary CSP by exploiting the 2-section [2] of the constraint hypergraph (also called primal graph), as it will be

* This work was supported by the French National Research Agency under grant TUPLES (ANR-2010-BLAN-0210).

done for our experiments since we will consider binary and non-binary CSPs. Moreover, without loss of generality, we assume that the network is connected. To simplify the notations, in the sequel, we denote the graph $(X, \{S(C_1), \ldots S(C_e)\})$ by (X, C). An assignment on a subset of X is said to be *consistent* if it does not violate any constraint. Determining whether a CSP has a *solution* (i.e. a consistent assignment on all the variables) is known to be NP-complete. So, many works have been realized to make the solving of instances more efficient in practice, by using optimized backtracking algorithms which may exploit heuristics, constraint learning, non-chronological backtracking, filtering techniques based on constraint propagation, etc. The time complexity of these backtracking methods is naturally exponential, at least in $O(e.d^n)$ with n the number of variables, d the maximum size of domains and e the number of constraints.

Another way is related to the study of tractable classes defined by properties of constraint networks. E.g., it has been shown that if the structure of this network is acyclic, it can be solved in linear time [3]. Using and generalizing these theoretical results, some methods to solve CSPs have been defined, such as Tree-Clustering [4]. This kind of methods is based on the notion of tree-decomposition of graphs [5]. Their advantage is related to their theoretical complexity, that is d^{w+1} where w is the tree-width of the constraint graph. When this graph has nice topological properties and thus when w is small, these methods allow to solve large instances, e.g. radio link frequency assignment problems [6]. Note that in practice, the time complexity is more related to $d^{w^+ + 1}$ where $w^+ \geq w$ is actually an approximation of the tree-width because computing an optimal tree-decomposition (of width w) is an NP-hard problem.

However, the practical implementation of such methods, even though it often shows its interest, has proved that the minimization of the parameter w^+ is not necessarily the most appropriate. Besides the difficulty of computing the optimal value of w^+, i.e. w, it sometimes leads to handle optimal decompositions, but whose properties are not always adapted to a solving that would be the most efficient. This has led to propose graph decomposition methods that make the solving of CSPs more efficient in practice, but for which the value of w^+ can even be really greater than w [7].

In this paper, we show that a reason to this lack of efficiency for solving CSPs using decomposition can be found in the nature of the decompositions for which w^+ is close to w. Indeed, minimizing w^+ can lead to decompositions such that some clusters have several connected components. Unfortunately, this lack of connectedness may lead the solving method to spend large amount of efforts to solve the subproblems related to these disconnected clusters, by passing many times from a connected component to another. To avoid this problem, we consider here a new kind of graph decomposition called *Bag-Connected Tree-Decomposition*[1] and its associated parameter called *Bag-Connected Tree-Width* [8]. This parameter is equal to the minimal width over all the tree-decompositions for which each cluster has a single connected component. So, the Bag-Connected Tree-Width will be the minimum width for all Connected Tree-Decompositions. The notion of Bag-Connected Tree-Width has been introduced very recently and to date, only studied in [8] from a mathematical viewpoint. Here we analyze this concept in terms of its algorithmic properties. So, we firstly prove that its com-

[1] We use the term "bag" rather than "cluster" because it is more compatible with the terminology of Graph Theory.

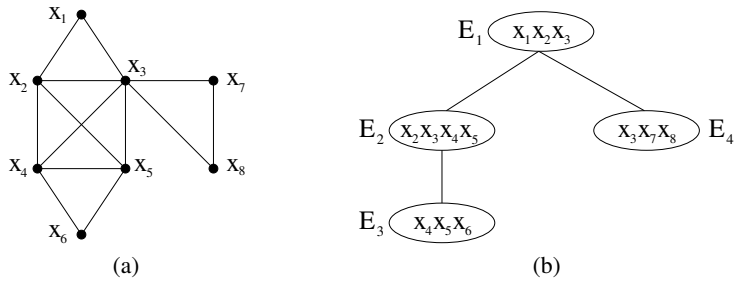

Fig. 1. A constraint graph for 8 variables (a) and an optimal tree-decomposition (b).

putation is NP-hard. So, we propose a first polynomial time algorithm (in $O(n(n+e))$) in order to approximate this parameter, and the associated decompositions. The experiments we present show the relevance of this parameter, since it allows to significantly improve the solving of CSPs by decomposition.

Note that the present work is applied to tree-decompositions, but it can also be adapted to most decompositions (e.g. [9, 10]). Indeed, in most CSP solving methods based on a decomposition approach, the decompositions are computed by algorithms which aim to approximate at best a graphical parameter (width) without taking into account the connectedness of produced clusters, neither the solving step. So, the problems observed here for tree-decomposition can also occur for other decompositions.

Section 2 introduces notations and the principles of tree-decomposition methods for solving CSPs. Section 3 points to some problems due to the computing of "good" tree-decompositions while section 4 presents the notion of bag-connected tree-decomposition, proposing a first algorithm to achieve one. Before concluding, we empirically show the interest of the use of this graph parameter for the practical solving of CSPs in section 5.

2 Solving CSPs using Graph Decomposition

Tree-Clustering (denoted TC [4]) is the reference method for solving binary CSPs by exploiting the structure of their constraint graph. It is based on the notion of tree-decomposition of graphs [5].

Definition 1 *Given a graph $G = (X, C)$, a tree-decomposition of G is a pair (E, T) with $T = (I, F)$ a tree and $E = \{E_i : i \in I\}$ a family of subsets of X, such that each subset (called cluster or bag in Graph Theory) E_i is a node of T and satisfies (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 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(b) presents a tree whose nodes correspond to the maximal cliques of the graph depicted in Figure 1(a). It is a possible tree-decomposition for this graph. So, we get $E_1 = \{x_1, x_2, x_3\}$, $E_2 = \{x_2, x_3, x_4, x_5\}$, $E_3 = \{x_4, x_5, x_6\}$, and $E_4 = \{x_3, x_7, x_8\}$. The tree-width of this graph is 3 as the maximum size of clusters is 4.

The first version of TC [4], begins by computing a tree-decomposition (using the algorithm MCS [11]). In the second step, the clusters are solved independently, considering each cluster as a subproblem, and then, enumerating all its solutions. After this, a global solution of the CSP, if one exists, can be found efficiently exploiting the tree structure of the decomposition. Time and space complexities of this first version is $O(n.d^{w^+ +1})$ where $w^+ +1$ is the size of the largest cluster ($w+1 \leq w^+ +1 \leq n$). Note that this first approach has been improved to reach a space complexity in $O(n.s.d^s)$ [12, 13] where s is the size of the largest intersection (*separator*) between two clusters ($s \leq w^+$). Unfortunately, this kind of approach which solves completely each cluster is not efficient in practice. So, later, the Backtracking on Tree-Decomposition method (denoted BTD) [14] has been proposed and shown to be really more efficient from a practical viewpoint and appears in the state of the art as a reference method for this type of approach [15]. In contrast to TC, BTD does not need to solve completely each cluster to find a solution. A backtrack search is realized, exploiting a variable ordering induced by a depth first traversal of the tree-decomposition. While this approach has shown its practical interest, from a theoretical viewpoint, in the worst case, it has the same complexities as the improved version of TC, that is $O(n.d^{w^+ +1})$ for time complexity, and $O(n.s.d^s)$ for space complexity. So, to make structural methods efficient, we must a priori minimize the values of w^+ and s when computing the tree-decomposition. Unfortunately, computing an optimal tree-decomposition (i.e. a tree-decomposition of width w) is NP-hard [16]. So, many works deal with this problem. They often exploit an algorithmic approach related to *triangulated* graphs (see [17] for an introduction to triangulated graphs). We can distinguish different classes of approaches. On the one hand, the methods looking for optimal decompositions or their approximations have not shown their practical interest, due to a too expensive runtime w.r.t. the weak improvement of the value w^+. On the other hand, the methods with no guarantee of optimality (like ones based on *heuristic triangulations*) are commonly used. They run in polynomial time (between $O(n + e)$ and $O(n^3)$), are easy to implement and their advantage seems justified. Indeed, these heuristics appear to obtain triangulations reasonably close to the optimum [18]. In practice, the most used methods to find tree-decompositions are based on MCS [11] and Min-Fill [19] which give good approximations of w^+. Moreover, in [7], experiments have shown that the efficiency for solving CSPs is not only related to the value of w^+, but also to the value of s. Nevertheless, to our knowledge, these studies were only focused on the values of w^+ and s, not on the structure of clusters which seems to be a more relevant parameter. This question is studied in the next section, showing that topological properties of clusters constitute also a crucial parameter for solving CSPs.

Before that, we recall how compute a tree-decomposition with the Min-Fill heuristic. The first step, which corresponds to Min-Fill, is to calculate a triangulation of the graph. For a given graph $G = (X, C)$, a set of edges C' will be added so that the resulting graph $G' = (X, C \cup C')$ is triangulated. Min-Fill will order the vertices from 1 to n. At each step, a vertex is numbered by choosing a unnumbered vertex x that minimizes the number of edges to be added in G' to make a clique with the set of unnumbered neighboring vertices of x. Once a vertex is numbered, it will be eliminated. After this processing, the vertices have been numbered from 1 to n, and it is ensured

that for a given vertex x with number i, its neighboring vertices in G' with a higher number $j > i$, form a clique. The order defined by these numbers is called a *perfect elimination order*. The cost of this first step is $O(n^3)$. The second step is to compute the maximal cliques of G'. Since G' is triangulated and we have a perfect elimination order, it can be achieved in linear time, i.e. in $O(n + e')$ where $e' = |C \cup C'|$ [20, 17]. Each maximal clique corresponds to a cluster of the associated tree decomposition. The third step computes the tree structure of the decomposition. Several approaches exist. A simple way consists in computing a maximum spanning tree (the constraint graph is assumed to be connected) of a graph whose vertices correspond to the maximal cliques (i.e. clusters E_i), and edges link two maximal cliques sharing at least one vertex and are labeled with the size of these intersections. This treatment can be achieved in $O(n^3)$ (e.g. by Prim's algorithm). Overall, the cumulative cost of these three steps is in $O(n^3)$.

3 Disconnected Clusters and their Impact on the Efficiency of Decomposition Methods

The study of the tree-decompositions shows they can frequently possess clusters that have several connected components. For example, consider a cycle without chord (that is without edge joining two non-consecutive vertices in the cycle) of n vertices (with $n \geq 4$). Any optimal tree-decomposition has exactly $n-2$ clusters of size 3, and among them, $n - 4$ clusters have two connected components.

This phenomenon is also observed for real instances, when we consider tree-decompositions of good quality. For example, the well known RLFAP instance *Scen-06* appearing in the CSP 2008 Competition[2] is defined on 200 variables and its network admit good tree-decompositions which can be found quite easily (e.g. Min-Fill finds one with $w^+ = 20$). Unfortunately, a detailed analysis of these tree-decompositions shows that they have several disconnected clusters. More generally, it turns out that about 32% of the 7,272 instances of the CSP 2008 Competition have a tree-decomposition with at least one disconnected cluster when MCS or Min-Fill are used, what is generally the case of most tree-decomposition methods for solving CSPs. Among these instances for which MCS or Min-Fill produce tree-decompositions with disconnected clusters, we can notably find most of the RLFAP or FAPP instances which are often exploited as benchmarks for decomposition methods for both decision and optimization problems. Moreover, sometimes, the percentage of disconnected clusters in one instance may be very large up to 99% and about 35% in average. For the FAPP instances, the average is about 48% for tree-decompositions produced by Min-Fill, and a greater average using MCS. This observation will be even more striking for algorithms that find decompositions with smaller widths, as suggested by the example of the cycle without chord.

The presence of disconnected clusters in the considered tree-decomposition can have a negative impact on the practical efficiency of decomposition methods which can be penalized by a large amount of time or memory to solve the instance. Firstly, it is well known that if a constraint network is not connected, this can have important consequences on the effectiveness of its solving. For example, if one of its connected

[2] See http://www.cril.univ-artois.fr/CPAI08 for more details.

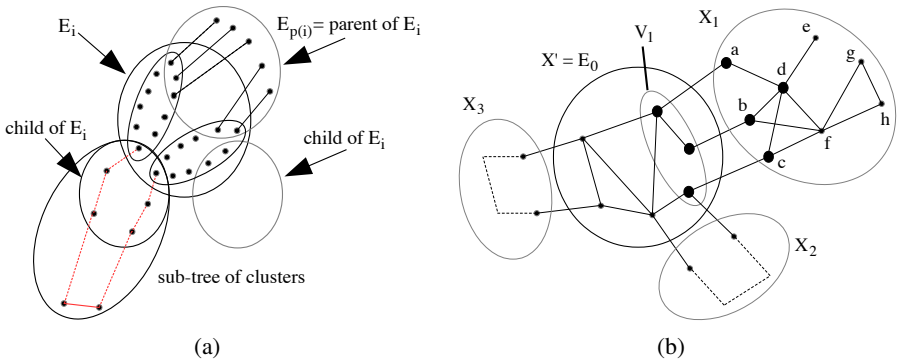

Fig. 2. (a) Disconnected cluster in a Tree-Decomposition, (b) First pass in the loop for *Bag-Connected-TD*.

components has no solution, and if the solving first addresses a connected component that has solutions, all of them should be listed before proving the inconsistency of the whole CSP. In the case of decomposition methods, the existence of disconnected clusters is perhaps even more pernicious. In the case of TC, let us consider a disconnected cluster. On the one hand, the phenomenon already encountered in the case of disconnected networks may arise. But it is also possible that this cluster has solutions. All these solutions will be calculated and stored before processing another cluster. Their number can be significant as it is the product of the number of solutions of each of its connected components. Note that for some benchmarks coming from the FAPP instances, the number of connected components in one cluster can be greater than 100. However, many local solutions of this cluster may be globally incompatible, because these connected components may be linked by some constraints which appear in other clusters. Figure 2(a) shows an example of decomposition for which two connected components of a cluster E_i are connected by a sequence of constraints that appear in the subproblem rooted in this cluster. Thus, the overall inconsistency of local solutions of E_i can only be detected when all these clusters have been solved, during the composition of global solutions produced by TC in its last step. This leads TC to a large consumption of time and memory, making this approach unrealistic in practice.

Other methods were proposed to avoid this kind of phenomenon where clusters are initially solved independently. This is notably the case of BTD which is one of the most effective approaches based on decompositions. Although BTD has shown its practical interest, unfortunately, the observed phenomenon still exists, even if it will generally be attenuated. To well understand this, we must remind that BTD solves an instance by solving successively the subproblems rooted in every cluster of the tree-decomposition. But unlike TC which first calculates all the solutions of a cluster, when accessing a cluster, BTD only computes one solution. Roughly speaking, the subproblem rooted in a cluster E_i corresponds to the subproblem involving all the variables of the descendants of E_i in the tree-decomposition (see [14] for more details). In practice, BTD starts its backtrack search by assigning consistently the variables of the root cluster before exploring a child cluster. When exploring a new cluster E_i, it only assigns the variables which appear in the cluster E_i but not in its parent cluster $E_{p(i)}$, that is all the variables

of the cluster E_i except the variables of the separator $E_i \cap E_{p(i)}$[3]. For instance, let us consider the constraint graph of Figure 1 and its associated tree-decomposition. If we assume that E_1 is the root cluster, BTD first tries to assign consistently the variables of E_1. If so, it keeps on the search with one of its child clusters (i.e. E_2 or E_4). If BTD chooses to explore first E_2, it will have to assign consistently the variables of $E_2 \backslash (E_1 \cap E_2)$ (i.e x_4 and x_5).
Now and more generally, let us consider a disconnected cluster E_i. We have two cases:

- if $G[E_i \backslash (E_i \cap E_{p(i)})]$[4] is disconnected: BTD has to consistently assign variables which are distributed in several connected components. If the subproblem rooted in E_i is trivially consistent (for instance it admits a large number of solutions), BTD will find a solution by doing at most a few backtracks and keep on the search on the next cluster. So, in such a case, the non-connectivity of E_i does not entail any problem. In contrast, if this subproblem has few solutions or none, we have a significant probability that BTD passes many times from a connected component of $G[E_i \backslash (E_i \cap E_{p(i)})]$ to another when it solves this cluster. Roughly speaking, BTD may have to explore all the consistent assignments of each connected component by interleaving eventually the variables of the different connected components. Indeed, if BTD exploits filtering techniques, the assignment of a value to a variable x of $E_i \backslash (E_i \cap E_{p(i)})$ has mainly impact on the variables of the connected component of $G[E_i \backslash (E_i \cap E_{p(i)})]$ which contains x. In contrast, the filtering does not modify or slightly the domain of any variable in another connected component. This entails that inconsistencies are often detected later and not necessarily in E_i but in one of its descendant cluster (as illustrated previously by Figure 2(a)). If so, BTD may require a large amount of time or memory (due to (no)good recording) to solve the subproblem rooted in E_i, especially if the variables have large domains. For example, this negative phenomenon has been empirically observed on some FAPP instances (e.g the fapp05-0350-10 instance) with a BTD version using MAC [21].
- if $G[E_i \backslash (E_i \cap E_{p(i)})]$ is connected: it follows that E_i is a disconnected cluster because its separator with its parent cluster is disconnected. As the variables of this separator are already assigned, the non-connectivity of E_i does not cause any problem.

This negative impact of disconnected clusters is compatible with empirical results reported in the literature. We have observed that sometimes, the percentage of disconnected clusters for Min-Fill differs significantly from one for MCS, which may explain some differences of efficiency observed by different authors. Indeed, even if the width is the same, decompositions computed by Min-Fill offer best results for solving than the ones obtained by MCS [7] and is considered as the best heuristic of the state of the art now. Moreover, the analysis of tree-decompositions shows also that the connection between connected components of some clusters is frequently observed in the leaves (clusters) of the decomposition, further increasing more the negative effects observed. To avoid this kind of phenomenon, we study classes of tree-decompositions for which all the clusters are connected.

[3] We assume that $E_i \cap E_{p(i)} = \emptyset$ if E_i is the root cluster.
[4] For any $Y \subseteq X$, the subgraph $G[Y]$ of $G = (X, C)$ induced by Y is the graph (Y, C_Y) where $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

4 A New Parameter for Graph Decomposition of CSPS

4.1 Bag-Connected Tree-Decomposition

The facts presented above lead us naturally to consider only tree-decompositions for which all the clusters are connected. This concept has been recently introduced in the context of Graph Theory [8]. It has been studied for some of its combinatorial properties. However, the algorithmic issues related to its computation have not been studied yet, neither in terms of complexity, nor to propose algorithms to find them. [8] provides a central theorem indicating an upper bound of Bag-Connected Tree-Width[5] as a function of the tree-width. We present now the notion of Bag-Connected Tree-Decomposition, which corresponds to tree-decomposition for which each cluster E_i is connected (i.e. the subgraph $G[E_i]$ of G induced by E_i is a connected graph).

Definition 2 *Given a graph $G = (X, C)$, a **Bag-Connected Tree-Decomposition** of G is a tree-decomposition (E, T) of G such that for all $E_i \in E$, the subgraph $G[E_i]$ is a connected graph. The width of a Bag-Connected Tree-Decomposition (E, T) is equal to $max_{i \in I}|E_i| - 1$. The **Bag-Connected Tree-Width** w_c is the minimal width over all the bag-connected tree-decompositions of G.*

Given a graph $G = (X, C)$ of tree-width w, necessarily $w \leq w_c$. The central theorem of [8] provides an upper bound of the Bag-Connected Tree-Width as a function of the tree-width and k which is the maximum length of its geodesic cycles[6]. More precisely, we have $w_c \leq w + \binom{w+1}{2}.(k.w - 1)$ ($k = 1$ if G has no cycle). Note that $w_c = \lceil \frac{n}{2} \rceil$ for graphs defined by cycles of length n and without chord. Nevertheless, if G is a triangulated graph, $w = w_c$.

Furthermore, the fact that $w \leq w_c$, independently of the complexity of achieving a Bag-Connected Tree-Decomposition, indicates that the decomposition methods based on it, necessarily appear below Tree-Decomposition methods in the hierarchy introduced in [9]. But this remark has no real interest here because our contribution mainly concerns practical efficiency of such methods. Nevertheless, the difference between w and w_c can naturally have incidences on the efficiency of solving in practice. Indeed, if we consider the example of the cycle of length n given in section 3 (a geodesic cycle), optimal decompositions give $w = 2$ and $w_c = \lceil \frac{n}{2} \rceil$. But, in such a case, even if the bag-connected tree-width is arbitrarily greater than the tree-width, applying BTD based on MAC is always as effective since as soon as the first variable is assigned, BTD detects the inconsistency or directly finds a solution, due to the arc-consistency propagation which will be realized along the connected paths in the clusters.

The natural question now is related to the computation of optimal Bag-Connected Tree-Decompositions, that is Bag-Connected Tree-Decompositions of width w_c. We show that this problem, as for Tree-Decompositions, is NP-hard.

[5] Note that we use the term of Bag-Connected Tree-Width rather than one of Connected Tree-Width exploited in [8] because the term of Connected Tree-Width has been introduced before in [22] but corresponds to a quite different concept.

[6] A cycle is said *geodesic* if for any pair of vertices x and y belonging to the cycle, the distance between x and y in the graph is equal to the length of the shortest path between x and y in the cycle.

Theorem 1 *Computing an optimal Bag-Connected Tree-Decomposition is NP-hard.*

Proof: We propose a polynomial reduction from the problem of computing an optimal tree-decomposition to this one. Consider a graph $G = (X, C)$ of tree-width w, the associated tree-decomposition of G being (E, T). Now, consider the graph G' obtained by adding to G an universal vertex x, that is a vertex which is connected to all the vertices in G. Note that from (E, T), we can obtain a tree-decomposition for G' by adding in each cluster $E_i \in E$ the vertex x. It is a bag-connected tree-decomposition since each cluster is necessarily connected (by paths containing x) and its width is $w + 1$. To show that this addition defines a reduction, it is sufficient to show that w is the tree-width of G iff the bag-connected tree-width w_c of G' is $w + 1$.

($\Rightarrow$) We know that at most, the width of the considered tree-decomposition of G' is $w + 1$ since this tree-decomposition is connected and its width is $w + 1$. Thus, $w_c \leq w + 1$. Assume that $w_c \leq w$. So, there is a bag-connected tree-decomposition of G' of width at most w. Using this tree-decomposition of G', we can define the same tree, but deleting the vertex x, to obtain a tree-decomposition of G of width $w - 1$, which contradicts the hypothesis.

($\Leftarrow$) With the same kind of argument as before, we know that the tree-width w of G is at most $w_c - 1$. And by construction, it cannot be strictly less than $w_c - 1$. So, it is exactly $w_c - 1$.
Moreover, achieving G' is possible in linear time. □

We have seen that for solving CSPs, it is not necessary to find an optimal tree-decomposition, and this is even often desirable. Also, we now propose an algorithm which computes a bag-connected tree-decomposition in polynomial time, of course without any guarantee about its optimality. The algorithm *Bag-Connected-TD* described below finds a bag-connected tree-decomposition of a given graph $G = (X, C)$.

4.2 Computing a Bag-Connected Tree-Decomposition

The first step of Algorithm 1 finds a first cluster, denoted E_0, which is a subset of vertices which are connected. X' is the set of already treated vertices. It is initialized to E_0. This first step can be done easily, using an heuristic. Then, let $X_1, X_2, \ldots X_k$ be the connected components of the subgraph $G[X \backslash E_0]$ induced by the deletion of the vertices of E_0 in G. Each one of these sets is inserted in a queue F. For each element X_i removed from the queue F, let $V_i \subseteq X$ be the set of vertices in X' which are adjacent to at least one vertex in X_i. Note that V_i (which can be connected or not) is a separator of the graph G since the deletion of V_i in G makes G disconnected (X_i being disconnected from the rest of G). A new cluster E_i is then initialized by this set V_i. So, we consider the subgraph of G induced by V_i and X_i, that is $G[V_i \cup X_i]$. We choose a first vertex $x \in X_i$ that is connected to at least one vertex of E_i (so one vertex of V_i). This vertex is added to E_i. If $G[E_i]$ is connected, we stop the process because we are sure that E_i will be a new connected cluster. Otherwise, we continue, taking another vertex of X_i.

Figure 2(b) shows the computation of E_1, the second cluster (after E_0), at the first pass in the loop. After the addition of vertices a, b and c, the subgraph $G[V_1 \cup \{a, b, c\}]$ is not connected. If the next reached vertex is d, it is added to E_1, and thus, $E_1 = V_1 \cup \{a, b, c, d\}$ is a new connected cluster, breaking the search in $G[V_1 \cup X_1]$.

Algorithm 1: $Bag\text{-}Connected\text{-}TD$

Input: A graph $G = (X, C)$
Output: A set of clusters $E_0, \dots E_m$ of a bag-connected tree-decomposition of G

1 Choose a first connected cluster E_0 in G
2 $X' \leftarrow E_0$
3 Let $X_1, \dots X_k$ be the connected components of $G[X \backslash E_0]$
4 $F \leftarrow \{X_1, \dots X_k\}$
5 **while** $F \neq \emptyset$ **do** `/* find a new cluster` E_i `*/`
6 Remove X_i from F
7 Let $V_i \subseteq X'$ be the neighborhood of X_i in G
8 $E_i \leftarrow V_i$
9 Search in $G[V_i \cup X_i]$ starting from $V_i \cup \{x\}$ with $x \in X_i$. Each time a new vertex x is found, it is added to E_i. The process stops once the subgraph $G[E_i]$ is connected
10 **if** V_i *belongs to the set of clusters already found* **then** Delete the cluster V_i (because $V_i \subsetneq E_i$)
11 $X' \leftarrow X' \cup E_i$
12 Let $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ be the connected components of $G[X_i \backslash E_i]$
13 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

When this process is finished, we add the vertices of E_i to X' and we compute $X_{i_1}, \dots X_{i_{k_i}}$ the connected components of the subgraph $G[X_i \backslash E_i]$. Each one is then inserted in the queue F. In the example of Figure 2(b), two connected components will be computed, $\{e\}$ and $\{f, g, h\}$. This process continues while the queue is not empty. In the example, in the right part of the graph, the algorithm will compute 3 connected clusters: $\{d, e\}$, $\{b, c, d, f\}$ and $\{f, g, h\}$.

Note that the line 10 is only useful when the set V_i computed at line 7 is a previously built cluster. In such a case, the cluster V_i can be removed. Indeed, as $V_i \subsetneq E_i$, V_i becomes useless in the tree-decomposition.

We now establish the validity of the algorithm and we evaluate its time complexity.

Theorem 2 *The algorithm Bag-Connected-TD computes the clusters of a bag-connected tree-decomposition of a graph G.*

Proof: We need only to prove the lines 5-13 of the algorithm. We first prove the termination of the algorithm. At each pass through the loop, at least one vertex will be added to the set X' and this vertex will not appear later in a new element of the queue because they are defined by the connected components of $G[X_i \backslash E_i]$, a subgraph that contains strictly fewer vertices than was contained in X_i. So, after a finite number of steps, the set $X_i \backslash E_i$ will be an empty set, and therefore no new addition in F will be possible.

We now show that the set of clusters $E_0, E_1, \dots E_m$ induces a bag-connected tree-decomposition. By construction each new cluster is connected. So, we have only to prove that they induce a tree-decomposition. We prove this by induction on the added clusters, showing that all these added clusters will induce a tree-decomposition of the graph $G(X')$.

Initially, the first cluster E_0 induces a tree-decomposition of the graph $G[E_0] = G[X']$.

For the induction, our hypothesis is that the set of already added clusters $E_0, E_1, \dots E_{i-1}$ induces a tree-decomposition of the graph $G[E_0 \cup E_1 \cup \dots \cup E_{i-1}]$. Consider now the addition of E_i. We show that by construction, $E_0, E_1, \dots E_{i-1}$ and E_i induces

a tree-decomposition of the graph $G[X']$ by showing that the three conditions (i), (ii) and (iii) of the definition of tree-decompositions are satisfied.

(i) Each new vertex added in X' belongs to E_i
(ii) Each new edge in $G[X']$ is inside the cluster E_i.
(iii) We can consider two different cases for a vertex $x \in E_i$, knowing that for other vertices, the property is already satisfied by the induction hypothesis:
  (a) $x \in E_i \backslash V_i$: in this case, x does not appear in another cluster than E_i and then, the property holds.
  (b) $x \in V_i$: in this case, by the induction hypothesis, the property was already verified.

Finally, it is easy to see that if the line 10 is applied, we obtain a tree-decomposition of the graph $G[X']$. □

Theorem 3 *The time complexity of the algorithm Bag-Connected-TD is* $O(n(n+e))$.

Proof: The lines 1-4 are feasible in linear time, that is $\mathrm{O}(n+e)$, since the cost of computing the connected components of $G[X \backslash E_0]$ is bounded by $\mathrm{O}(n+e)$. Nevertheless, we can note that the line 1 can be done by a more expensive heuristic to get a more relevant first cluster, but at most in $\mathrm{O}(n(n+e))$ in order not to exceed the time complexity of the most expensive step of the algorithm. We analyze now the cost of the loop (line 5). Firstly, note that there is less than n insertions in the queue F. Now, we analyze the cost of each treatment associated to the addition of a new cluster, and we give for each one, its global complexity.

- Line 6: obtaining the first element X_i of F is bounded by $\mathrm{O}(n)$, thus globally $\mathrm{O}(n^2)$.
- Line 7: obtaining the neighborhood $V_i \subseteq X'$ of X_i in G is bounded by $\mathrm{O}(n+e)$, thus globally by $\mathrm{O}(n(n+e))$.
- Line 8: this step is feasible in $\mathrm{O}(n)$, thus globally $\mathrm{O}(n^2)$.
- Line 9: the cost of the search in $G[V_i \cup X_i]$ starting with vertices of V_i and $x \in X_i$ is bounded by $\mathrm{O}(n+e)$. Since the while loop runs at most n times, the global cost of the search in these subgraphs is bounded by $\mathrm{O}(n(n+e))$. Moreover, for each new added vertex x, the connectivity of $G[E_i]$ is tested with an additional cost bounded by $\mathrm{O}(n+e)$. Note since such a vertex is added at most one time, globally, the cost of this test is bounded by $\mathrm{O}(n(n+e))$. So, the cost of the line 9 is globally bounded by $\mathrm{O}(n(n+e))$.
- Line 10: using an efficient data structure, this step can be realized in $\mathrm{O}(n)$, thus globally $\mathrm{O}(n^2)$.
- Line 11: the cost of finding the connected components of $G[X_i \backslash E_i]$ is bounded by $\mathrm{O}(n+e)$. So, globally, the cost of this step is $\mathrm{O}(n(n+e))$.
- Line 12: the insertion of a set X_{i_j} in F is feasible in $\mathrm{O}(n)$, thus globally $\mathrm{O}(n^2)$ since there is less than n insertions in F.

Finally, the time complexity of the algorithm *Bag-Connected-TD* is $\mathrm{O}(n(n+e))$. □

From a practical viewpoint, it can be assumed that the choice of the first cluster E_0 can be crucial for the quality of the decomposition which is being computed. Similarly, the choice of vertex x, selected in line 9 may be of considerable importance. For these two choices, heuristics can of course be used. This is discussed in the next section. However, a particular choice of these heuristics makes it possible, without any change of the complexity, to compute optimal tree-decompositions for the case of triangulated graphs. Assume that the first cluster E_0 is a maximal clique. This can be done efficiently using a greedy approach. Now, for the choice of the vertex x in line 9, we consider the vertex which has the maximum number of neighbors in the set V_i. As in a triangulated graph, all the clusters of an optimal tree-decomposition are cliques, necessarily, V_i being a clique, x will be connected to all the vertices of V_i and thus, E_i will be a clique. Progressively, each maximal clique will be found and the tree-decomposition will be optimal. Line 10 will be used for the case of maximal cliques including more than one vertex x of a new connected component. In any case, the practical interest of this type of decomposition is based on both the efficiency of its computation, but also on the significance which it may have for solving CSPs. This is discussed in section 5.

5 Experiments

In this section, we mainly compare the solving efficiency and the structural parameters for BTD using tree-decompositions produced by Min-Fill with ones for BTD exploiting bag-connected tree-decompositions. These latter are computed thanks to the *Bag-Connected-TD* algorithm. Regarding the choice of the first cluster in *Bag-Connected-TD*, it consists in computing greedily a maximal clique of the constraint network[7]. To choose the next vertex, we have considered 6 heuristics. We present here the best ones:

- NV1: the next vertex is a vertex in the neighborhood of previously chosen vertices,
- NV2: the vertices are processed in the decreasing degree order,
- NV3: the vertices are processed according to the order they are visited by a breadth-first traversal of the graph from the vertices of V_i,
- NV4: we choose as next vertex the vertex which has the maximum number of neighbors in the set V_i.

The solving is achieved by BTD based on MAC, by using the dom/wdeg variable heuristic [23]. We choose as root cluster the cluster E_i which maximizes the ratio $\frac{e_i}{|E_i|-1}$ with e_i the number of constraints of the cluster E_i. This choice provides better results than ones of [24]. The decomposition runtimes for Min-Fill and *Bag-Connected-TD* are similar and are included in the runtime of BTD. All the implementations are written in C++. The experimentations are performed on a linux-based PC with an Intel Pentium IV 3.2 GHz and 1 GB of memory.

5.1 Instances for which Min-Fill produces some disconnected clusters

In this subsection, we compare the bag-connected tree-decompositions with disconnected ones from the viewpoint of the solving efficiency. With this aim in view, we

[7] Remind that we use the 2-section for non-binary instances.

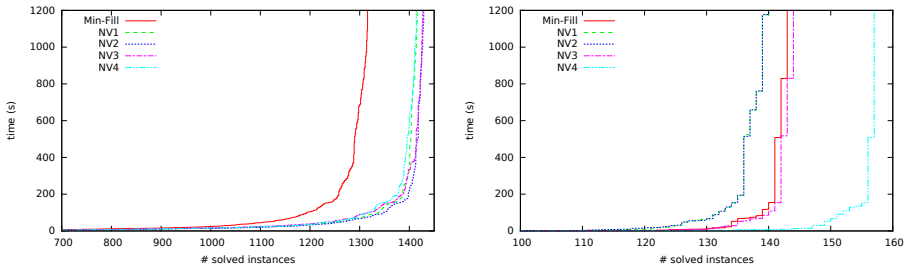

Fig. 3. The cumulative number of solved instances for each considered tree-decomposition for instances for which Min-Fill produces some disconnected clusters (a), for instances for which Min-Fill produces a bag-connected tree-decomposition (b).

consider 1,597 instances (of arbitrary arity) among the 2,310 instances of the CSP 2008 Competition for which Min-Fill produces a tree-decomposition with at least one disconnected cluster. The excluded instances are instances which cannot be solved without exceeding the time limit (namely 1,200 s) or which contain global constraints (not implemented yet in our solver). Among the considered instances, we can notably find instances from families rlfap, fapp, modifiedRenault, graphColoring, bqwh or travellingSalesman.

Figure 3(a) presents the cumulative number of solved instances for each considered tree-decomposition. First, we can observe that, by using any bag-connected tree-decomposition, BTD solves more instances than by using the disconnected tree-decompositions produced by Min-Fill. Note that this observation remains true if we use any connected decomposition based on the non presented heuristics. The best number of solved instances is reached thanks to the tree-decomposition based on the heuristic NV2 and NV3. These decomposition allow us to solve respectively 114 and 111 additional instances w.r.t. Min-Fill. Those based on NV1 and NV4 are close to each other. Moreover, for any decomposition, most instances are solved in less than 60 s.

Now, in order to fairly compare the runtimes, we only consider the instances which are solved by BTD for all the considered tree-decompositions, including Min-Fill. The runtime for solving the 1,230 instances by using the decompositions based on Min-Fill is 50,669 s while by using the connected decompositions based on NV1, it requires only 32,372. The connected decomposition based on NV2 is relatively close to NV1 (namely 33,202 s). Those based on NV3 and NV4 are slightly slower (respectively 36,420 s and 36,087 s). Note that the two other heuristics (not presented here) also outperform the Min-Fill decomposition.

If we focus on the 329 instances having a suitable structure (i.e. instances having a ratio n/w^+ greater than 2), again, we observe the same trend, namely that BTD with bag-connected tree-decomposition performs better than BTD with Min-Fill. The best cumulative runtime is achieved by BTD using NV1 in 5,698 s while the worst one is obtained by BTD using Min-Fill in 13,641 s. BTD using NV2 and NV4 are close to each other with respectively 6,137 s and 6,010 s while BTD with NV3 needs 8,483 s.

Finally, if we compare these results with ones obtained by a classical enumerative algorithm like MAC, we can note that some instances solved by BTD with some NV_i

are not solved by MAC and conversely. We also observe that MAC performs sometimes better, sometimes worse than BTD using connected decompositions. However, globally, they have similar results. This is explained by the fact that most of the 1,597 instances we consider are far from having a suitable structure. In contrast, when the structure has interesting features, BTD outperforms MAC. For instance, BTD with decomposition based on NV3 requires 856 s for solving 10 instances over the 12 instances of rlfap-Scens11 family while MAC only solves 8 instances in 1,595 s. Moreover, for solving these 8 instances, BTD requires only 63 s, that is more than 25 times faster than MAC.

5.2 Instances for which Min-Fill produces a bag-connected tree-decomposition

This subsection briefly deals with the behavior of BTD when solving instances for which Min-Fill produces a bag-connected tree-decomposition. Of course, for such instances, Min-Fill and our *Bag-Connected-TD* algorithm do not necessarily produce the same tree-decompositions. By lack of place, we focus directly our study on the more relevant instances (i.e. the 191 instances having a suitable structure for a decomposition approach).

As shown in Figure 3(b), BTD using Min-Fill succeeds in solving more instances than BTD using NV1 or NV2 (143 instances against 140) but less than BTD using NV3 and NV4 which solve respectively 144 and 157 instances. If we focus our study on the 132 instances which are solved by BTD for all the considered tree-decompositions, including Min-Fill, BTD using NV3, NV4 or Min-Fill obtain the best cumulative runtime (respectively in 1,283 s, 1,298 s and 1,280 s) while BTD using NV1 or NV2 are slower (respectively 2,226 s and 2,265 s).

5.3 Comparisons of the structural parameters

Table 1 presents the value of the structural parameters for some instances. Not surprisingly, Min-Fill produces tree-decompositions with smaller widths and larger numbers of clusters than ones produced by *Bag-Connected-TD*. However, if in some cases, the width obtained by *Bag-Connected-TD* is significantly larger than one provided by Min-Fill (e.g. the width produced by NV3 for instance squares-23-23), in other cases, it remains relatively close (even sometimes equal) to one obtained by Min-Fill. This notably occurs for instance renault-mod-33_ext but also for instances for which Min-Fill produces a bag-connected tree-decomposition (see part (b) of Table 1). We also observe that the quality of the width obtained thanks to *Bag-Connected-TD* may significantly vary depending on the considered instances. If NV1 often presents the best width among ones computed by *Bag-Connected-TD* algorithm, it is sometimes outperformed by NV3 or NV4 (e.g. for instance mps-red-qnet1).

Regarding the parameter s, the observed trends are similar to ones for the width.

6 Conclusion

In this paper, we have introduced the concept of Bag-Connected Tree-Decomposition in the field of constraint network decomposition. After having shown the interest of this

Table 1. Value of the structural parameters for some instances for which Min-Fill produces some disconnected clusters (a), for which Min-Fill produces a bag-connected tree-decomposition (b).

	Instances	n	e	Min-Fill		NV1		NV2		NV3		NV4	
				w^+	s	w_c^+	s	w_c^+	s	w_c^+	s	w_c^+	s
(a)	2-insertions-4-3	149	541	38	34	66	54	95	14	101	66	58	57
	ewddr2-10-by-5-9	50	265	16	15	22	17	21	20	26	23	45	37
	renault-mod-33_ext	111	133	11	11	12	11	14	11	17	15	16	13
	scen7	400	2,865	33	29	90	48	319	9	116	94	81	34
	squares-23-23	1,058	1,268	45	4	45	5	45	5	235	88	45	26
	fapp06-0500-1	500	3,478	221	210	286	284	286	284	314	314	313	248
	js-taillard-15-100-4	225	1785	86	70	114	102	121	97	129	102	210	197
(b)	mps-red-qnet1	5,380	621	970	773	1,272	1,265	1,272	1,265	978	954	998	974
	anna-9	138	493	12	12	14	14	14	14	16	15	14	13
	haystacks-10	100	459	9	1	9	1	9	1	9	1	9	1
	renault-mod-8_ext	111	126	11	11	11	11	12	11	13	12	11	11
	qwh-15-106-9_ext	225	2324	99	99	102	102	102	102	103	103	173	168

new parameter and proposed a first polynomial time algorithm which computes Bag-Connected Tree-Decompositions, we have experimentally demonstrated the relevance of this approach since it allows to significantly improve the solving of CSP using decomposition methods. Indeed, by using bag-connected tree-decompositions, BTD succeeds in solving many more instances. Moreover, the improvement of the runtime is approximately 63% w.r.t. BTD using tree-decompositions with disconnected clusters produced by Min-Fill. This benefit mainly comes from the connectedness of clusters since when Min-Fill produces bag-connected tree-decompositions, the results of the different versions of BTD are close. Finally, thanks to these bag-connected decompositions, BTD also can significantly outperform MAC for well structured instances (e.g. for the rlfapScens11 family, BTD can be 25 times faster than MAC).

The first extension of this work is related to the study of bag-connected tree-decompositions in the more general field of Graphical Models in AI. This concerns the study of this notion for other classes of methods as Hypertree-Decomposition, And/Or Search, Bucket Elimination, etc. This approach is particularly justified by the fact that, even if some of these approaches are based on other parameters (e.g. Hypertree-Decomposition), their efficient implementations use generally algorithms coming from Tree-Decompositions (e.g. Min-Fill for Hypertree-Decomposition [25]). Another promising study is related to the use of bag-connected tree-decompositions in the field of optimization and counting problems. The second extension of this work is related to a theoretical study of this new graph parameter from a mathematical viewpoint. For example, what are the fundamental properties of this parameter? For what classes of graphs, this parameter is easy to compute, or is close to the tree-width? Or, are there problems which are hard when the tree-width is bounded by a constant, and which are easy when the bag-connected tree-width is bounded by a constant?

References

1. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.
2. C. Berge. *Graphs and Hypergraphs*. Elsevier, 1973.
3. E. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1):24–32, 1982.
4. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
5. N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.
6. C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4:79–89, 1999.
7. P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of CP*, pages 777–781, 2005.
8. M. Müller. Connected tree-width. *CoRR*, abs/1211.7353, 2012.
9. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
10. M. Gyssens, P. Jeavons, and D. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66:57–89, 1994.
11. 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.
12. R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.
13. R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.
14. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.
15. J. Petke. *On the bridge between Constraint Satisfaction and Boolean Satisfiability*. PhD thesis, University of Oxford, 2012.
16. S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics*, 8:277–284, 1987.
17. M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
18. U. Kjaerulff. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical report, Judex R.R. Aalborg., Denmark, 1990.
19. D. J. Rose. A graph theoretic study of the numerical solution of sparse positive denite systems of linear equations. In *Graph Theory and Computing*, pages 183–217. R.C. Read (ed.), Academic Press, New York, 1973.
20. F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1 (2):180–187, 1972.
21. D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI*, pages 125–129, 1994.
22. P. Fraigniaud and N. Nisse. Connected treewidth and connected graph searching. In *Proceedings of LATIN*, pages 479–490, 2006.
23. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI*, pages 146–150, 2004.
24. P. Jégou, S. N. Ndiaye, and C. Terrioux. An extension of complexity bounds and dynamic heuristics for tree-decompositions of CSP. In *Proceedings of CP*, pages 741–745, 2006.
25. A. Dermaku, T. Ganzow, G. Gottlob, B. J. McMahan, N. Musliu, and M. Samer. Heuristic methods for hypertree decomposition. In *Proceedings of MICAI*, pages 1–11, 2008.