Connected Tree-Width: A New Parameter for Graph Decomposition

Philippe Jégou and **Cyril Terrioux**
LSIS - UMR CNRS 7296
Aix-Marseille Université
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, cyril.terrioux}@lsis.org

Abstract

For solving constraints networks (CSPs), (tree-)decomposition methods have shown their practical interest. But for the problem of computing tree-decompositions, the literature (coming from AI or from Mathematics) has concentrated the work on a single parameter, the tree-width. Nevertheless, experimental studies have shown that when a decomposition is used to solve a CSP, other parameters must also be considered. For example, it has been observed that in some cases, bad decompositions w.r.t. their width can be more efficient for the resolution of the underlying CSP.

One of the explanations of this phenomenon is related to the structure of the clusters appearing in decompositions. For example, in this paper, we show experimentally that some clusters can have several connected components when we try to minimize the width to achieve "good" decompositions. Unfortunately, this lack of connectedness may lead the solving method to spend much effort to solve the subproblems related to these non-connected clusters, by passing many times from a connected component to another. Clearly, this can be a real drawback for globally solving a CSP in terms of time efficiency and memory space. To avoid this kind of problem, we introduce here a new graph parameter called *Connected Tree-Width* which considers tree decompositions for which each cluster is connected. We show that computing such optimal decompositions is an NP-hard problem. So, we propose a polynomial time algorithm to find such decompositions, but obviously, without guaranteeing optimality.

Introduction

Constraint Satisfaction Problems (CSPs, see (Rossi, van Beek, and Walsh 2006) 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 (Berge 1973) of the constraint hypergraph (also known as the primal graph in the CSP community). 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. Testing 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, heuristics, constraint learning, non-chronological backtracking, filtering techniques based on constraint propagation, etc. The time complexity for these approaches is naturally exponential, at least in $O(n.d^n)$ where n is the number of variables and d the maximum size of domains.

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 (Freuder 1982). Using these theoretical results, some methods to solve CSPs have been defined, such as Tree-Clustering (Dechter and Pearl 1989). This kind of methods is based on the notion of tree-decomposition of graphs (Robertson and Seymour 1986). 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 that w is small, these methods allow to solve large instances, e.g. radio link frequency assignment problems (Cabon et al. 1999). Note that in practice, the time complexity is more related to $d^{w^+ +1}$ where 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 computation of the optimal value of w^+, that is w, it sometimes leads to handle optimal decompositions, but whose properties are not always adapted to a resolution that would be the most efficient. This has led to propose graph decomposition methods that make the solving of CSPs more efficient (from a practical viewpoint), but for which the value of w^+ can even be really greater than w (Jégou, Ndiaye, and Terrioux 2005).

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 many efforts to solve the subproblems related to these non-connected clusters, by passing many times from a connected component to another.

To avoid this problem, we introduce here a new graph invariant, a parameter called *Connected Tree-Width*. This parameter is equal to the minimal width over all the tree-decompositions for which each cluster has a single connected component. So, the Connected Tree-Width will be the minimum width for all Connected Tree-Decompositions. Here we prove that its computation is NP-hard. So, we propose a first polynomial time algorithm in order to approximate this parameter, and the associated decompositions. Its time complexity is $O(n(n+e))$.

The next section introduces notations and the principles of tree-decomposition methods for solving CSPs. The third section points to some problems due to the computing of "good" tree-decompositions while the fourth section introduces the notion to connected tree-decomposition, proposing a first algorithm to achieve one. The last section presents a conclusion.

Solving CSPs using Graph Decomposition

Tree-Decomposition Methods

Tree-Clustering (denoted TC (Dechter and Pearl 1989)) is the reference method for solving binary CSPs by exploiting the structure of its constraint graph. It is based on the notion of tree-decomposition of graphs (Robertson and Seymour 1986).

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) 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 $E_i \cap E_j \subseteq E_k$.

Note that the third condition (iii) can be replaced by: if a vertex x satisfies $x \in E_i \cap E_j$, then, all the nodes E_k in T appearing on the unique path from E_i to E_j contain x. 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 maximum size of clusters is 4 and thus, the tree-width of this graph is 3.

The first version of TC (Dechter and Pearl 1989), begins by computing a tree-decomposition (using the algorithm MCS (Tarjan and Yannakakis 1984)). 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 it has been later shown (Dechter and Fattah 2001; Dechter 2003) that this first approach can be improved for space complexity in $O(n.s.d^s)$ where s is the size of the largest minimal *separator*, i.e. the size of the largest intersection 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) has been proposed and shown to be really more efficient from a practical viewpoint (Jégou and Terrioux 2003; Jégou, Ndiaye, and Terrioux 2005; 2007) and appears in the state of the art as a reference method for this type of approach (Karakashian 2013). 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, its complexities are the same as ones of 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.

Computing Tree-Decomposition

Computing an optimal tree-decomposition (i.e. a tree-decomposition of width w) is NP-hard (Arnborg, Corneil, and Proskuroswki 1987). So, many works deal with this problem. They often exploit an algorithmic approach related to *triangulated* graphs (see (Golumbic 1980) for an introduction to triangulated graphs). A graph is said triangulated if it has a *perfect elimination order* (p.e.o.), i.e. a vertex order $\sigma = (x_1, \dots, x_n)$ such that, for any vertex x_i, the vertices in the neighborhood of x_i which follow x_i in σ form a clique. The link between triangulated graphs and tree-decompositions is obvious. Indeed, given a triangulated graph, the set of maximal cliques $E = \{E_1, E_2, \dots, E_k\}$ of (X, C) corresponds to the family of clusters associated with a tree-decomposition. The graph in Figure 1(a) is already triangulated, a possible p.e.o. being $(x_1, x_2, x_8, x_7, x_3, x_4, x_5, x_6)$. As any graph G is not necessarily triangulated, we can obtain a tree-decomposition by triangulating G. We call *triangulation* the addition to G of a set C' of edges such that $G' = (X, C \cup C')$ is triangulated. The width of G' is equal to the maximal size of cliques mi-

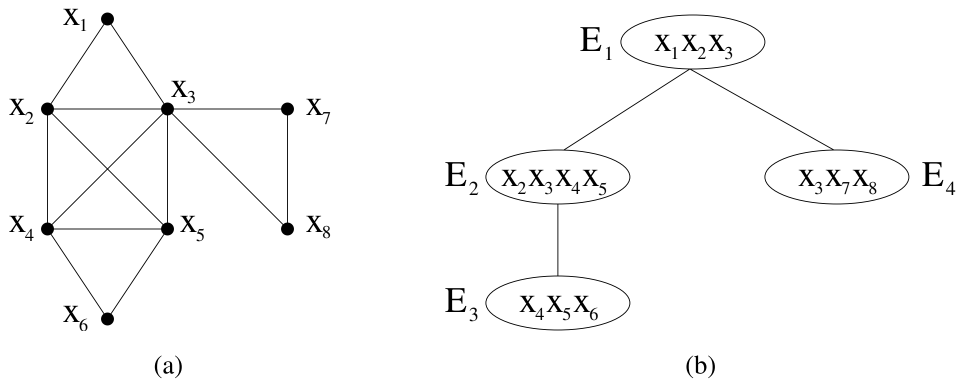

Figure 1: A constraint graph for 8 variables (a) and an optimal tree-decomposition (b).

nus one in graph G'. The tree-width of G is then equal to the minimal width over all triangulations.

Several approaches and algorithms have been proposed for triangulations. We can distinguish four classes of approaches. First, computing an *optimal triangulation* is NP-hard. So no polynomial algorithm is known yet and the proposed algorithms have an exponential time complexity. Secondly, we can exploit *approximation algorithms* which approximate the optimum by a constant factor and whose complexity is often polynomial in the tree-width (Amir 2001). Unfortunately, implementing these two first approaches do not have much interest from a practical viewpoint (e.g. the latter is time expensive while obtaining results of poor quality). On the other hand, we can exploit *minimal triangulations*. A minimal triangulation computes a set C' s.t. $(X, C \cup C')$ is triangulated and, for every subset $C'' \subsetneq C'$, $(X, C \cup C'')$ is not triangulated. Note that a minimal triangulation is not necessarily optimal. The main interest of this approach is related to the existence of polynomial algorithms (e.g. LEX-M (Rose, Tarjan, and Lueker 1976) and LB (Berry 1999) whose time complexity is $O(ne')$ with e' the number of edges in the triangulated graph). Finally, the fourth approach, namely *heuristic triangulations*, generally add some edges to the initial graph until the graph is triangulated. They often achieve this work in polynomial time (between $O(n + e')$ and $O(n(n + e')))$ but they do not provide any minimality warranty. Nonetheless, in practice, they can be easily implemented and their interest seems justified. Indeed, these heuristics appear to obtain triangulations reasonably close to the optimum (Kjaerulff 1990). The most popular heuristics are MCS and Min-Fill. MCS (as for TC) relies on the algorithm of (Tarjan and Yannakakis 1984) which recognizes the triangulated graphs. Min-Fill (Rose 1973) orders the vertices from 1 to n by choosing as next vertex one which leads to add a minimum of edges when completing the subgraph induced by its unnumbered neighbors.

The two first approaches do not appear very interesting as a first step of a CSP solving method due to a too expensive runtime w.r.t. the weak improvement of the value w^+. So, in practice, the most used methods to find tree-decomposition are based on MCS and Min-Fill which run faster than LEX-M or LB with similar approximations of w^+. Moreover, in (Jégou, Ndiaye, and Terrioux 2005), 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. It has also been experimentally shown that growing the value of w^+ while minimizing the value of s, seems to offer a good approach to improve the search for solving the underlying CSP. To solve CSPs more efficiently, it was proposed in this study, given a tree-decomposition, to merge some clusters sharing numerous variables, to obtain a new tree-decomposition with a larger size of clusters, but with separators of smaller size. This kind of approach was also justified by the use of heuristics in the search of local solutions, which seems to be more efficient with larger clusters. A theoretical study analyzing bounds for time complexity of such an approach is given in (Jégou, Ndiaye, and Terrioux 2007). Nevertheless, these studies were only focused on the values of the parameters 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 seems to be a crucial parameter for solving CSPs.

Impact of Non-Connected Clusters on the Efficiency of BTD-like methods

In this section, we first observe that, from a CSP instance with a connected constraint (hyper)graph $G = (X, C)$, classical tree-decomposition methods like MCS or Min-Fill may produce tree-decompositions for which some clusters have several connected components. In the following, such clusters are called non-connected. Then we study the consequences which may happen when a method like BTD solves an instance with such a tree-decomposition.

Up to now, the tree-decompositions exploited by BTD-like methods are generally computed by using MCS or Min-Fill. Unfortunately, MCS and Min-Fill may produce tree-decompositions with non-connected clusters. For instance, it turns out that about 32 % of the 7272 instances of the CSP 2008 Competition[1] have a tree-decomposition with at least one non-connected cluster when MCS or Min-Fill are used

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

to compute tree-decompositions. Among the instances for which MCS or Min-Fill produce tree-decompositions with non-connected clusters, we can notably find most of the RLFAP or FAPP instances which are often exploited as benchmarks for BTD-like methods for both decision and optimization problems. A priori, this observation will be even more striking for algorithms that find decompositions with smaller widths.

Regarding the solving with BTD, the presence of non-connected clusters in the considered tree-decomposition does not necessarily entail a negative impact on the practical efficiency of the method. However, if BTD is penalized by the presence of such clusters, it may require a large amount of time or memory to solve the instance. To well understand this phenomenon, we must remind that BTD solves an instance by solving successively the subproblems rooted in every cluster of the tree-decomposition. 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 (Jégou and Terrioux 2003) 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 appears 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 $E_i \cap E_{p(i)}$[2]. 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, let us consider a non-connected cluster E_i. We have several cases:

- if $G[E_i \backslash (E_i \cap E_{p(i)})]$[3] 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. 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 instance, this negative phenomenon has been empirically observed on some FAPP instances (e.g the normalized-fapp05-0350-10 instance) with a BTD version based on MAC (Sabin and Freuder 1994).
- if $G[E_i \backslash (E_i \cap E_{p(i)})]$ is connected: it follows that E_i is a non-connected 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 entail any problem.

[2]We assume that $E_i \cap E_{p(i)} = \emptyset$ if E_i is the root cluster.

[3]For any $Y \subseteq X$, the subgraph $G[Y]$ of $G = (X, C)$ induced by Y is the graph (Y, C_Y) where $C_Y = \{\{x, y\} \in C | x, y \in Y\}$.

Note that, even if by construction and assuming that the constraint network is connected, there is always a connected cluster in the computed tree-decomposition, the root cluster, which is chosen heuristically at the start of BTD, may be a non-connected cluster. Moreover, we clearly see that the order according to which the clusters are explored by BTD plays a significant role in the occurrence of the negative phenomenon and so in the practical efficiency of BTD. Unfortunately, we have observed at many times that, for some instances, whatever the order we consider, we have at least one cluster E_i such that $G[E_i \backslash (E_i \cap E_{p(i)})]$ is disconnected. Sometimes, the percentage of non-connected clusters may be very important up to 99 % and about 35 % in average. For instance, for the FAPP instances, the average is about 48 % for tree-decompositions produced by Min-Fill. That is why it seems necessary to consider tree-decompositions which take into account the connectivity inside the clusters.

Moreover, we could be attempted to exploit non-chronological backtracking like backjumping (Rossi, van Beek, and Walsh 2006) instead of the basic chronological backtracking. However, such a technique is generally useless here. Indeed, generally, the connected components of a non-connected cluster E_i have connections between them in the clusters of the descent of E_i in the tree-decomposition. So, as most of solvers exploit a filtering algorithm at least as powerful as Arc-Consistency (Rossi, van Beek, and Walsh 2006), the exploitation of non-chronological backtracking will be too expensive w.r.t. the benefits it can bring.

Finally, these observations are compatible with our previous empirical results (Jégou, Ndiaye, and Terrioux 2005; 2007). In particular, they provide a new viewpoint to explain the results obtained about the impact of the choice of a triangulation method or of the cluster order (heuristics for choosing the root cluster or the next child cluster) on the practical efficiency of BTD. For instance, we have observed that sometimes, the percentage of non-connected clusters for Min-Fill differs significantly from one for MCS, which may explain some differences of efficiency observed in previous works. Regarding the improvement of the efficiency of BTD when merging some clusters (Jégou, Ndiaye, and Terrioux 2005), we can note that this merging reduces significantly the number of non-connected clusters. For instance, if, from the tree-decompositions produced by MCS or Min-Fill, we merge the clusters which have a separator with their parent cluster greater than 5, only 12 % of the instances of the CSP 2008 Competition still have a tree-decomposition with non-connected clusters.

A New Parameter for Graph Decomposition

Connected Tree-Decomposition

We define now the notion of Connected Tree-Decomposition, which corresponds to tree-decomposition for which each cluster E_i is connected (i.e. $G[E_i]$ is a connected graph).

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

Given a graph $G = (X, C)$ of tree-width w, necessarily $w \leq w_c$. Nevertheless, if G is a chordal graph, $w = w_c$. If not, for example for cycles of length k without chords, the connected tree-width of such graphs is $\lceil \frac{k}{2} \rceil$.

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

Theorem 1 *Computing an optimal 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 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 connected tree-width w_c of G' is $w + 1$.

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 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.
2. ($\Leftarrow$) 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 running in polynomial time that makes it possible to find a connected tree-decomposition, of course without any guarantee about the optimality of the result.

Computing a Connected Tree-Decomposition

The algorithm $Connected\text{-}TD$ described below finds a connected tree-decomposition of a given graph $G = (X, C)$.

The first step of the algorithm finds a first cluster, denoted E_0, which is a subset of vertices which are connected. X' is initialized to E_0. X' will be the set of already treated vertices. This first step can be done easily, using an heuristic. Then, let $X_1, X_2, \dots X_k$ 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 that G is not connected, 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 $x \in X_i$ which 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 satisfying the same condition, that is connected to at least one vertex of E_i. This step can be done by a breadth first search starting from the vertices of V_i.

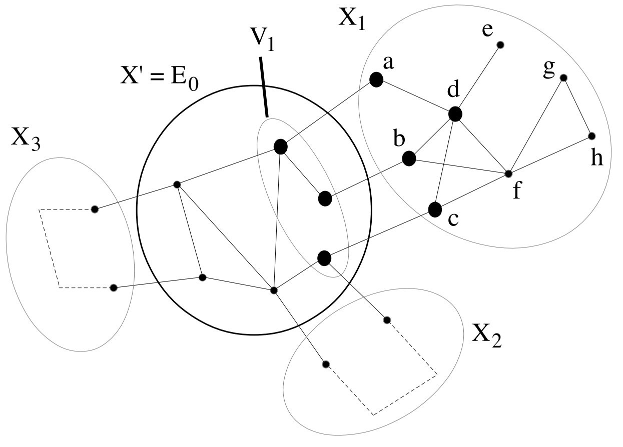

Figure 2: First pass in the loop for $Connected\text{-}TD$

Figure 2 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 $G[V_1 \cup X_1]$.

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, 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 $\{f, g, h\}$. The algorithm is summarized below.

Algorithm *Connected-TD*
Input: A graph $G = (X, C)$
Output: A set of clusters $E_0, E_1, \dots E_m$ of a 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 $E_i$ */
   (a) Remove $X_i$ from $F$
   (b) Let $V_i \subseteq X'$ be the neighborhood of $X_i$ in $G$
   (c) $E_i \leftarrow V_i$
   (d) Search in $G[V_i \cup X_i]$ starting from $V_i$ plus $x \in X_i$. 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.
   (e) if $V_i$ belongs to the set of clusters already found
       then the cluster $V_i$ is deleted (because $V_i \subsetneq E_i$)
   (f) $X' \leftarrow X' \cup E_i$
   (g) Let $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ be the connected components of
       $G[X_i \backslash E_i]$
   (h) $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$
```

Note that the step 5(e) is only useful when the algorithm builds a cluster E_i containing all the vertices of a previously built cluster E_j (i.e. $E_j \subsetneq E_i$). In such a case, the cluster E_j can be removed since it is useless in the tree-decomposition.

We have now to prove the validity of this algorithm.

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

Proof: We need only to prove the step 5 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 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 step (5e) is applied, we obtain a tree-decomposition of the graph $G[X']$. □

We now show that the complexity of this algorithm is polynomial.

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

Proof: The steps (1), (2), (3) and (4) are feasible in linear time, that is O$(n+e)$, since the cost of computing the connected components of $G[X \backslash E_0]$ is bounded by O$(n+e)$. Nevertheless, we can note that the step (1) can be done by a more expensive heuristic to get a more relevant first cluster, but at most in 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 (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.

(a) Obtaining the first element X_i of F is bounded by O(n), thus globally O(n^2).

(b) Obtaining the neighborhood $V_i \subseteq X'$ of X_i in G is bounded by O$(n+e)$, thus globally by O$(n(n+e))$.

(c) This step is feasible in O(n), thus globally O(n^2).

(d) 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 O$(n+e)$. Since the while loop runs at most n times, the global cost of the search in these subgraphs is bounded by 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 O$(n+e)$. Note since a such vertex is added at most one time, globally, the cost of this test is bounded by O$(n(n+e))$. So, the cost of the step (d) is globally bounded by O$(n(n+e))$.

(e) Using an efficient data structure, this step can be realized in O(n), thus globally O(n^2).

(f) This step is feasible in O(n), thus globally O(n^2).

(g) The cost of finding the connected components of $G[X_i \backslash E_i]$ is bounded by $O(n+e)$. So, globally, the cost of this step is $O(n(n+e))$.

(h) The insertion of a set X_{i_j} in F is feasible in $O(n)$, thus globally $O(n^2)$ since there is less than n insertions in F.

Finally, the time complexity of the algorithm *Connected-TD* is $O(n(n+e))$, due to the step (5). □

From a practical point of view, it can be assumed that the choice of the first cluster E_0 can be crucial for the quality of the decomposition which will be computed. This applies to both the parameter w_c but also to its location in the graph. The more this cluster will be "central" in the graph, the more it seems that the decomposition could be useful. To realize a good choice for this first cluster, it would then be possible to use a method as the one proposed recently in (Benlic and Hao 2013) to realize this first step.

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 experimental work must now be done.

Conclusion

In this paper, we have introduced the concept of Connected Tree-Decomposition and a new graph invariant associated to this kind of decomposition, the Connected-Tree-Width. This notion has been proposed to make easier the solving of CSPs by decomposition methods. Indeed, we have experimentally observed that the best tree-decompositions used to solve CSPs, frequently admit collections of clusters which have several connected components, a topological feature which can degrade the efficiency of the solving. Since the problem of finding a Connected Tree-Decomposition of minimum width is NP-hard, we have proposed a first polynomial time algorithm which computes Connected Tree-Decompositions (not necessarily optimal). Its time complexity is $O(n(n+e))$ where n is the number of vertices (variables of CSPs) and e is the number of edges (binary constraints).

There are at least two natural ways to continue this work. The first one is related to an experimental study of this notion to evaluate the interest of this new class of tree-decompositions for solving CSPs. Is this new family of tree-decompositions more relevant than the existing ones? The answer to this fundamental question requires us to realize experiments with the decomposition methods of the state of the art, using classical tree-decompositions and their new implementation using connected tree-decompositions. 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? Or, are there problems which are hard when the tree-width is bounded by a constant, and which are easy when the connected tree-width is bounded by a constant?

Acknowledgments

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

The authors would like to thank Ioan Todinca for their fruitful discussion about this work.

References

Amir, E. 2001. Efficient approximation for triangulation of minimum treewidth. In *Proceedings of UAI*, 7–15.

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

Benlic, U., and Hao, J.-K. 2013. Breakout local search for the vertex separator problem. In *Proceedings of IJCAI*, 461–467.

Berge, C. 1973. *Graphs and Hypergraphs*. Elsevier.

Berry, A. 1999. A Wide-Range Efficient Algorithm for Minimal Triangulation. In *Proceedings of SODA'99 SIAM Conference*.

Cabon, C.; de Givry, S.; Lobjois, L.; Schiex, T.; and Warners, J. P. 1999. Radio Link Frequency Assignment. *Constraints* 4:79–89.

Dechter, R., and Fattah, Y. E. 2001. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence* 125:93–118.

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

Dechter, R. 2003. *Constraint processing*. Morgan Kaufmann Publishers.

Freuder, E. 1982. A Sufficient Condition for Backtrack-Free Search. *JACM* 29 (1):24–32.

Golumbic, M. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

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

Jégou, P.; Ndiaye, S. N.; and Terrioux, C. 2005. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of CP*, 777–781.

Jégou, P.; Ndiaye, S.; and Terrioux, C. 2007. 'Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs. In *Proceedings of IJCAI*, 112–117.

Karakashian, S. 2013. *Practical Tractability of CSPs by Higher Level Consistency and Tree Decomposition*. Ph.D. Dissertation, University of Nebraska-Lincoln, USA.

Kjaerulff, U. 1990. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical report, Judex R.R. Aalborg., Denmark.

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

Rose, D.; Tarjan, R.; and Lueker, G. 1976. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on computing* 5:266–283.

Rose, D. J. 1973. A graph theoretic study of the numerical solution of sparse positive denite systems of linear equations. In *Graph Theory and Computing*, 183–217. R.C. Read (ed.), Academic Press, New York.

Rossi, F.; van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier.

Sabin, D., and Freuder, E. 1994. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI*, 125–129.

Tarjan, R., and Yannakakis, M. 1984. 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.