An Algorithmic Framework for Decomposing Constraint Networks

Philippe Jégou Hanan Kanso Cyril Terrioux
Aix-Marseille Université, CNRS
ENSAM, Université de Toulon, LSIS UMR 7296
13397 Marseille Cedex 20 (France)
{*philippe.jegou, hanan.kanso, cyril.terrioux*}*@lsis.org*

***Abstract*—Depending on the nature of CSP instances to consider, the decomposition methods offer an approach often efficient for the solving, the counting of solutions or the optimization. So, the community has focused a large part of its efforts on the design of algorithms aiming to find the best decompositions, i.e. ones which minimize the width of the decomposition, the fundamental parameter in terms of theoretical complexity. In this frame, the heuristic *Min-Fill* constitutes the reference method.**

In this paper, we introduce an algorithmic framework for network decomposition aiming to improve *Min-Fill*. It computes tree-decompositions based on a traversal of the graph using properties related to separators and their associated connected components. Its time complexity is lower than the one of *Min-Fill*. Moreover, it permits the implementation of several heuristics which can guide the decomposition over several criteria like the size of the clusters, the size of the separators, the connectivity of the clusters, which are particularly relevant to improve the efficiency of the solving by decomposition methods. Experiments assess this new approach and demonstrate its practical advantages for decomposing and solving constraint networks.

***Keywords*-CSPs; constraint networks; decomposition;**

I. Introduction

An instance of a finite *Constraint Satisfaction Problem* (CSP) is given by a triple (X, D, C), with $X = \{x_1, \ldots, x_n\}$ a set of n variables, $D = (D_{x_1}, \ldots, D_{x_n})$ a set of finite domains, and $C = \{c_1, \ldots, c_e\}$ a set of constraints. Each constraint c_i is a pair $(S(c_i), R(c_i))$, where $S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ defines 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 $|S(c_i)|$. If the arity of each constraint is two, the instance is a *binary* CSP. The structure of a constraint network (other name of a CSP) is given by a hypergraph (a graph for a binary CSP), called the *constraint (hyper)graph*, whose vertices correspond to variables while edges correspond to the scopes of the constraints. To simplify notations, we denote the hypergraph $(X, \{S(c_1), \ldots S(c_e)\})$ by (X, C). Without loss of generality, we assume that considered networks are connected. An assignment on a subset of X is called *consistent* if all the constraints are satisfied. Checking whether a CSP has a *solution* (a consistent assignment of X) is well known to be NP-complete. So, many works have been done to improve the solving in practice, even if the complexity of these approaches remains exponential, at least in $O(n.d^n)$ where d is the maximum size of domains. To circumvent this theoretical intractability, other approaches have been proposed. Some of them rely on a structural tractable class based on the notion of *tree-decomposition of graphs* [1] (a tree of clusters of vertices). Their primary advantage is related to their theoretical complexity, d^{w+1} where w is the *tree-width* of the constraint network (this notion can also be considered when the network is a hypergraph using its *2-section* [2]). Thus, these methods can be efficient on large instances of small tree-width as it is the case for example for well known optimization problems of radio frequency allocations [3]. Moreover, tree-decompositions can be used to address more difficult problems as counting problems or knowledge compilation. Unfortunately, computing an optimal decomposition is an NP-hard problem [4] and thus in practice, the time complexity is generally $d^{w^+ + 1}$ where $w^+ \geq w$ is an approximation of w. So, computing efficiently a decomposition of small width is today a very challenging issue with numerous practical applications. Thereby, the design of decomposition algorithms tools is an important field of research for solving CSPs, as well as in many other domains (e.g. in AI with the Probabilistic Reasoning, in Operational Research, etc.). Thus, in practice, the decompositions are generally computed by heuristic approaches for which *Min-Fill* [5] is to date the best compromise between the computation time and the quality of the decomposition. We may even see that the efficient computation tools for hypertree-decomposition [6] (a generalization of the tree-decomposition) are based on *Min-Fill* [7]. However, *Min-Fill* has some drawbacks. First, it proceeds by *triangulation*, that is by adding edges. This can make it time-expensive, and thus ineffective in the case of large graphs. Moreover, *Min-Fill* does not explicitly take into account the topological properties of the graph, and thus sometimes leads to the achievement of bad tree-width approximations. Another major drawback is related to the fact that the clusters of the tree-decompositions computed by *Min-Fill* are frequently disconnected. As pointed in [8], this phenomenon can significantly slow down the search. A last significant drawback is related to the maximum size s of the separators (intersections between clusters of the tree-decomposition). Indeed, its value is frequently close to the value of w^+

and since the space complexity of decomposition methods is related to d^s, this can be really problematic for the efficiency of the search.

In order to avoid these drawbacks, we hereby introduce an algorithmic framework, i.e. a generic algorithm called *H-TD-WT* (for *Heuristic Tree-Decomposition Without Triangulation*), which can implement several heuristics whose quality of decompositions is generally better than the one of *Min-Fill*. *H-TD-WT* operates without triangulation. It performs the computation of a set of clusters based on a traversal of the graph using properties related to separators and their associated connected components. Its time complexity is related to the heuristic which is implemented, but this complexity is generally lower than the one of *Min-Fill*. The most interesting advantage of the framework is that, considering a suitable heuristic, it can avoid the associated drawbacks of *Min-Fill*. For example, if we want to minimize the width of the achieved decompositions, we can use a special heuristic which will generally find a better width than the one achieved by *Min-Fill*.

Section 2 recalls notions about tree-decompositions and their computation. Section 3 introduces the algorithmic framework *H-TD-WT* while Section 4 presents experiments that assess the relevance of our approach.

II. Tree-Decompositions of Constraint Networks

Decomposition methods for solving CSPs (or more difficult problems) like, for instance, *Tree-Clustering* [9], *Mini-Buckets and Bucket Elimination* [10], *BTD* [11] are based on the notion of *tree-decomposition of graphs* [1].

Definition 1: A *tree-decomposition* of a graph $G = (X, C)$ is a pair (E, T) with $T = (I, F)$ a tree (I the set of nodes and F the set of edges of T) 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 $E_i \cap E_j \subseteq E_k$. 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 of 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 $E_4 = \{x_3, x_7, x_8\}$. As the maximum size of clusters is 4, the tree-width of this graph is 3.

The complexity of decomposition methods for solving CSPs is based on the tree-width w of the constraint network, but more generally, on an approximation w^+ of w. More precisely, the time complexity of Tree-Clustering is $O(n.w^+.log(d).d^{w^+ + 1})$ and $O(n.d^{w^+ + 1})$ for space, where $w^+ + 1$ is the size of the largest cluster ($w + 1 \leq w^+ + 1 \leq n$). This first approach has been improved to obtain a space complexity in $O(n.s.d^s)$ [11], [12], [13] where s is the size of the

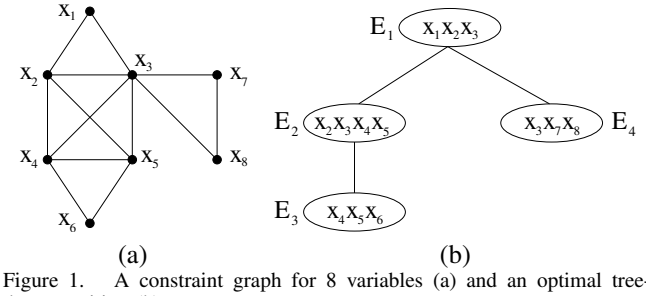

Figure 1. A constraint graph for 8 variables (a) and an optimal tree-decomposition (b).

largest intersection (*separator*) between two clusters ($s \leq w^+$). Making these structural methods effective requires the minimization of the values of w^+ and s in the computation of the tree-decomposition. This is even more crucial when considering, beyond the decision problems, optimization problems or counting problems. Unfortunately, computing an optimal tree-decomposition (i.e. with width equal to w) is NP-hard [4]. So, numerous works have addressed this issue. The methods for computing tree-decompositions often use algorithmic approaches based on the notion of *triangulated graphs* (see [14] for an introduction to this class of graphs), knowing that for these graphs, computing an optimal tree-decomposition is linear. We can distinguish different kinds of approaches. First, there are the exact methods [15] and the approximation methods with warranty [16]. But to date, they have not shown a practical advantage due to their large runtime. In fact, obtaining an optimum width seems unattainable in practice, except for very small graphs (a few dozen vertices at most [17]). Moreover, they only achieve a small improvement in the value of w^+ w.r.t. heuristic approaches. So, heuristic methods are the most used in practice even though they do not offer warranty in terms of optimality. These methods are generally based on *heuristic triangulations*. Their time complexity is polynomial (between $O(n + e)$ and $O(n^3)$), they are very simple to implement, and their practical advantage seems quite justified. Indeed, these heuristics seem to get quite close to the optimum triangulations [18]. The most known is *Min-Fill* [5] which often computes optimal decompositions in a much shorter time than the exact approaches [19]. So, in practice, *Min-Fill* is the reference approach since several decades. *Min-Fill* establishes a numbering and thus an ordering of the vertices of a graph G from 1 to n. In order to get a *perfect elimination ordering*, adding edges is required. The resulting graph G' is a triangulated graph. In this ordering, the neighbors of a given vertex which appear later in the ordering form a clique. At each step, *Min-Fill* chooses among the unnumbered vertices the vertex which leads to add the minimum number of edges in order to complete the subgraph induced by its unnumbered neighbors. The process continues until all the vertices of G are numbered. Once the graph is triangulated, it is very easy to compute all maximal cliques, and then to build a tree-decomposition since these

cliques form the set of clusters. Its time complexity is $O(n(n+e'))$ with e' the number of edges in the triangulated graph G'. Although *Min-Fill* turns to be the most expensive heuristic of the state of the art w.r.t. the time complexity, in practice, its runtime is much better than the one of exact methods. Moreover, the decompositions based on *Min-Fill* are often quite close to the optimum. Nevertheless, *Min-Fill* has several drawbacks that can lead to decompositions of very poor quality in some cases, either with respect to the value of w or for the solving step:

- The addition of edges caused by the principle of triangulation generates a significant additional cost in practice. For example, if we consider a *Grid Graph* with $k \times k$ vertices, the number of added edges by *Min-Fill* is in $\Omega(k^2)$ while the width is bounded by $\Theta(k)$.
- *Min-Fill* does not take into account explicitly the topological properties of the considered graph. For example, let us consider a graph with one *cut vertex* (an *articulation point*). If this vertex is chosen first by *Min-Fill*, while there are at least two biconnected components in the graph, these components will be directly connected after the triangulation. Nonetheless, if this topological feature is considered, these pointless connections will be avoided. This phenomenon is due to the fact that *Min-Fill* only considers (local) numerical parameters and does not consider explicitly the topological features of graphs.
- The obtained decomposition may contain disconnected clusters. As pointed out in [8], this phenomenon, occurs very often with *Min-Fill*. Moreover the existence of disconnected clusters can make the solving inefficient. So, in [8], it is shown that using decompositions with connected clusters generally leads to a significant improvement of the solving.
- The value of s is not controlled and can often be close to the value of w^+. Indeed, let us consider the first (w.r.t. the ordering found by *Min-Fill*) vertex x_i of a cluster denoted E_i whose size is $w^+ + 1$ in the decomposition. Assume that x_{i+1} is the first neighbor of x_i after x_i in the ordering and that it has one further neighbor x_k which is not connected to x_i. In this case, x_{i+1} is the second vertex of E_i and the first vertex of a cluster E_{i+1} whose size is also $w^+ + 1$. Indeed, since $x_{i+1} \in E_i$, the neighbors of x_{i+1} which are later in the ordering are exactly $(E_i \backslash \{x_i, x_{i+1}\}) \cup \{x_k\}$. So, $E_{i+1} = (E_i \backslash \{x_i\}) \cup \{x_k\}$ and thus $|E_{i+1}| = |E_i| - 1 + 1 = w^+ + 1$. As a consequence, the size of the separator between E_{i+1} and E_i is $|E_{i+1} \cap E_i| = w^+$. We can see that this phenomenon occurs very frequently using *Min-Fill*. Moreover as recalled above, a too high value of s can significantly slow down the search.

So, in the next section, we introduce a new algorithmic framework to avoid these disadvantages.

III. Decompose thanks to the Topology

A. Decomposition without triangulation

We propose an algorithm called *H-TD-WT* for *Heuristic Tree-Decomposition Without Triangulation*, which computes a tree-decomposition of a graph $G = (X, C)$, in polynomial time. Like *Min-Fill*, no warranty about its optimality is given. In the contrary, unlike *Min-Fill*, *H-TD-WT* is not based on triangulation and takes into account the topology of the graph. The goal is threefold: (1) to achieve better computation time, (2) to avoid some drawbacks of *Min-Fill* while exploiting some topological properties, and (3) to allow parameterization of the decomposition taking into account several criteria, some of them being related to w^+ and/or s. *H-TD-WT* can be considered as a generalization of the algorithm *Bag-Connected-TD* (denoted *BC-TD*) [8] which also computes tree-decompositions without triangulation. Indeed, we will see later that *BC-TD* can be considered as a special heuristic for *H-TD-WT*. The first step of *H-TD-WT* (line 1 in Algorithm 1) computes a first cluster, denoted E_0, thanks to a heuristic technique. X' which denotes the set of already considered vertices is initialized to E_0 (line 2). We denote $X_1, X_2, \dots X_k$ the connected components of the subgraph $G[X \backslash E_0]$ induced by the deletion in G of vertices of E_0[1]. Each one of these sets X_i is inserted in a queue F (line 4). For each element X_i deleted from F (line 6), V_i denotes the set of vertices of X' which are adjacent to at least one vertex of X_i (line 7). One can note that V_i is a separator in the graph G since removing V_i from G makes G disconnected (X_i being disconnected from the rest of G). We then consider the subgraph of G induced by V_i and X_i, that is $G[V_i \cup X_i]$.

The next step (line 8) can be parameterized. It looks for a subset of vertices $X''_i \subseteq X_i$ such that $X''_i \cup V_i$ will be a new cluster E_i of the decomposition. This can be ensured if there is at least one vertex v of V_i s.t. all its neighbors in X_i appear in X''_i. More precisely, if $N(v, X_i) = \{x \in X_i : \{v, x\} \in C\}$, we must ensure that $\exists v, N(v, X_i) \subseteq X''_i$. We then define a new cluster $E_i = X''_i \cup V_i$ (line 10). This process is repeated until the queue is empty.

Consider the example given in Figure 2. We first show the computation of E_1, the second cluster (after E_0) during the first pass through the loop. Assume that we consider a particular parameterization denoted *H_1-TD-WT* where we select X''_i by trying to minimize the size of the next cluster as follows. Among the vertices of $V_1 = \{v, w, x\}$, we consider the vertex which has a minimum of neighbors in X_1. This vertex is v since we have $N(v, X_1) = \{a\}$ and for the other vertices, this set is $N(w, X_1) = N(x, X_1) = \{b, c\}$. By this way, $X''_i = \{a\}$ and we have $N(v, X_i) \subseteq X''_i$. So the cluster E_1 is $V_1 \cup \{a\} = \{v, w, x, a\}$. Then, we get one new connected component:

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

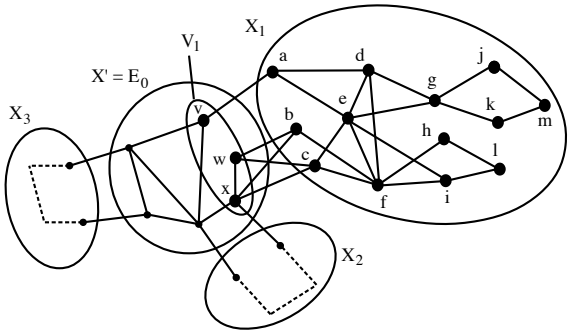

Figure 2. View of *H1-TD-WT*

Algorithm 1: $H\text{-}TD\text{-}WT$

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

1 Choose a first 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 new cluster` E_i `*/`
6 Delete X_i from F
7 Let $V_i \subseteq X'$ be the neighborhood of X_i in G
8 Find a subset $X_i'' \subseteq X_i$ such that there is at least one vertex $v \in V_i$ such that $N(v, X_i) \subseteq X_i''$
9 $E_i \leftarrow X_i'' \cup V_i$
10 $X' \leftarrow X' \cup X_i''$
11 Let $X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}$ be the connected components of $G[X_i \backslash E_i]$
12 $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$

$X_{1_1} = \{b, c, d, e, f, g, h, i, j, k, l, m\}$, which is then added to the queue F. When X_{1_1} is removed from the queue, we have $V_i = V_2 = \{a, w, x\}$. So, a new cluster E_2 is then computed. We have two possibilities: $V_2 \cup \{b, c\} = \{a, w, x, b, c\}$ or $V_2 \cup \{d, e\} = \{a, w, x, d, e\}$ because $N(a, X_{1_1}) = \{d, e\}$, $N(w, X_{1_1}) = \{b, c\}$ and $N(x, X_{1_1}) = \{b, c\}$. Assume that we choose $E_2 = \{a, w, x, b, c\}$. So, we get one new connected component $X_{2_1} = \{d, e, f, g, h, i, j, k, l, m\}$ which is added to F. When X_{2_1} is removed from the queue, we have $V_i = V_3 = \{a, b, c\}$. So, as $N(a, X_{2_1}) = \{d, e\}$, $N(b, X_{2_1}) = \{f\}$ and $N(c, X_{2_1}) = \{e, f\}$, the vertex which has a minimum of neighbors in X_{2_1} is b. So, the cluster E_3 is $V_3 \cup \{f\} = \{a, b, c, f\}$ and $X_{3_1} = \{d, e, g, h, i, j, k, l, m\}$ is added to F. When X_{3_1} is removed from F, we have $V_i = V_4 = \{a, c, f\}$. So, a new cluster E_4 is computed. We have one possibility: $V_4 \cup \{e\} = \{a, c, e, f\} = E_4$. So, we get two new connected components $X_{4_1} = \{d, g, j, k, m\}$ and $X_{4_2} = \{h, i, l\}$ which are added to F. And the process continues independently on each connected component.

Thus, one can note that the algorithm explicitly uses the topology of the graph through separators and related connected components. Each element in F is processed as described above until the queue is empty.

In [8], the algorithm *BC-TD* is illustrated by a very close example, but it is easy to see that this algorithm performs differently than *H_1-TD-WT*. This can be seen on the example of Figure 2 since *BC-TD* and *H_1-TD-WT* do not find the same tree-decompositions.

Theorem 1: H-TD-WT computes the clusters of a tree-decomposition.

Proof: It is sufficient to prove the correctness of lines 5 to 12 of the algorithm. First of all, we prove that the algorithm ends. For each iteration of the loop, at least one vertex of X_i is added to the set of treated vertices X'. This is due to the addition of the neighborhood $N(v, X_i)$ of at least one considered vertex v of the separator to X' (line 10). We guarantee that the newly added vertices of X_i'' in X' will not be included later in any element of the queue F. In fact, these elements are defined as the connected components of $G[X_i \backslash E_i]$, a subgraph which contains strictly less vertices than X_i. Hence, after a finite number of iterations, the set $X_i \backslash E_i$ will be empty and no new element will be added to F.

We now show that the set of clusters $E_0, E_1, \dots E_m$ induces a tree-decomposition of G. We prove that by induction on the set of added clusters, which will especially induce a tree-decomposition of the graph $G[X']$. Firstly, it is clear that the first cluster E_0 induces a tree-decomposition of the graph $G[E_0] = G[X']$. The inductive assumption says that the set of clusters already computed $E_0, E_1, \dots E_{i-1}$ induces a tree-decomposition of the graph $G[E_0 \cup E_1 \cup \dots \cup E_{i-1}]$. We consider now the newly computed cluster E_i. We show by construction that $E_0, E_1, \dots E_{i-1}$ and E_i induce a tree-decomposition of $G[X']$ by verifying that, the three conditions (i), (ii) et (iii) that define a tree-decomposition are satisfied.

(i) Every new added vertex to X' belongs to E_i.
(ii) Every new added edge of $G[X']$ is included in the cluster E_i.
(iii) We can consider two different cases for a vertex $x \in E_i$. For vertices not belonging to E_i, the property is a consequence of the inductive assumption. If $x \in V_i$, the property is also a consequence of the inductive assumption. If $x \in E_i \backslash V_i = X_i''$, x cannot appear in any other cluster than E_i. Thus the property is verified.

So, it is easy to verify that we get the set of clusters of a tree-decomposition of $G[X']$. This result is extended to G because at the end of processing, $X' = X$. □

From a practical point of view, we can assume that the choice of the first cluster E_0 can be crucial for the quality of the decomposition that is being calculated. Likewise, the choice of the vertex v selected (line 8) can be very important in practice when several vertices are eligible. Heuristics can be used. We propose one in Section 4. In the following, we present several heuristics that correspond to different possible parameterizations of the algorithm *H-TD-WT*. We will also see that the time complexity of this algorithm is

related to these heuristics.

B. Heuristics

Here we propose several heuristics. Their objective is either to minimize the width (*H_1-TD-WT*), or to make the decomposition more suited to the solving (*H_2-TD-WT*, *H_3-TD-WT* or *H_4-TD-WT*).

(1) *H_1-TD-WT*. As indicated above, this heuristic tries to minimize locally the size of the next cluster. So, we consider the smallest subset of vertices X_i'' to add to V_i to get the next cluster E_i. This is possible by looking for a vertex v of V_i which has a minimum of neighbors in X_i. Given this vertex, we find immediately the set $X_i'' = N(v, X_i)$ and thus $E_i = X_i'' \cup V_i$.

(2) *H_2-TD-WT*. This heuristic finds the next cluster E_i which must be connected. For more details, see the algorithm *BC-TD* which has been introduced in [8]. We show the computation of E_1 with the example given in Figure 2. First, we have $V_1 = \{v, w, x\}$ and $X_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m\}$. A possibility is to take $E_1 = \{v, w, x, a, c, e\}$ which is a connected subgraph such that $N(v, X_1) = \{a\} \subseteq X_i'' = \{a, c, e\}$.

(3) *H_3-TD-WT*. This heuristic constructs the next cluster E_i thanks to the topological properties of the graph. It aims to identify many independent parts of the graph and to separate them. To do so, it adds vertices to the next cluster E_i thanks to a breadth-first search starting from the vertices of V_i. Hence, the neighbors of V_i in X_i constitute the first level of X_i, the neighbors of neighbors of V_i in X_i constitute the second level of X_i, and so on. At the level $l = 1$, $E_{i_1} = N^*(V_i, X_i)$ where $N^*(V, X) = V \cup (\cup_{v \in V} N(v, X))$. At level l, $E_{i_l} = N^*(E_{i_{l-1}}, X_i)$. If we consider the example of Figure 2, $E_{1_1} = N^*(V_1, X_1) = \{v, w, x, a, b, c\}$, $E_{1_2} = N^*(E_{1_1}, X_1) = \{v, w, x, a, b, c, d, e, f\}$ and for the third level, $E_{1_3} = N^*(E_{1_2}, X_1) = \{v, w, x, a, b, c, d, e, f, g, h, i\}$ (there are 5 levels). The heuristic keeps on adding vertices while progressing through levels until X_i is separated in many connected components that is to say at $l = L$ where $G[X_i \backslash E_{i_L}]$ contains more than one connected component or until $G[X_i \backslash E_{i_L}]$ is empty. In the example, the breadth-first search is stopped at level 2 because $G[X_1 \backslash E_{1_2}]$ contains two connected components, $\{g, j, k, m\}$ and $\{h, i, l\}$. So, $E_1 = \{v, w, x, a, b, c, d, e, f\}$.

(4) *H_4-TD-WT*. This heuristic aims to limit the size of the separators of the decomposition. To do so, it considers a parameter S which represents the maximum allowed size for a separator. This heuristic adds new vertices to the next cluster E_i similarly to *H_3-TD-WT*. Nevertheless, the heuristic stops progressing through levels at $l = L$ when $G[X_i \backslash E_{i_L}]$ does not contain any connected component with separator's size greater than S.
With the example given in Figure 2, assuming that the maximum size of the separators is two. We find $E_1 = \{v, w, x, a, b, c, d, e, f\}$ because $\{d, e\}$ and $\{e, f\}$ are the separators respectively for $X_{1_1} = \{g, j, k, m\}$ and $X_{1_2} = \{h, i, l\}$.

Theorem 2: The time complexity is $O(n(n+e))$ for *H_1-TD-WT*, *H_2-TD-WT*, *H_3-TD-WT* and *H_4-TD-WT*.

Proof: Lines 1-4 can be done in linear time, that is to say in $O(n + e)$. This is due to the cost of computing the connected components of $G[X \backslash E_0]$ bounded by $O(n + e)$. Nevertheless, we note that line 1 can be achieved by a more expensive heuristic in order to obtain a more relevant first cluster. However, the heuristic's cost should be bounded by the global complexity so that the latter is not modified. Regarding line 5, we note that there are necessarily less than n insertions in the queue F. This is guaranteed because in each pass through the loop, we add new vertices to X' and delete them from the set of untreated vertices.

We analyze now the cost of each operation related to the newly added cluster. The given cost is the global cost of the operation.

- Line 6: we obtain the first element X_i of F in $O(n)$, that is to say $O(n^2)$ globally.
- Line 7: we obtain the neighborhood $V_i \subseteq X'$ of X_i in G in $O(n + e)$, that is to say $O(n(n + e))$ globally.
- Line 8: this step's cost depends on the chosen heuristic.
  - In *H_1-TD-WT*, this step is done in $O(n)$, that is to say $O(n^2)$ globally.
  - In *H_2-TD-WT*, this step is done in $O(n + e)$, that is to say $O(n(n + e))$ globally. In fact, the search for the subset X_i'' so that $V_i \cup X_i''$ is connected, is guaranteed to be bounded by $O(n(n+e))$ globally. More precisely, the connectivity test of $G[E_i]$, which is bounded by $O(n + e)$, is done only once for each added vertex.
  - In *H_3-TD-WT*, the search of vertices of level l, namely the neighbors of vertices of level $l - 1$ in $X_i \backslash E_{i_{l-1}}$, is done in $O(n + e)$. The cost of computing the connected components of $G[X_i \backslash E_{i_{l-1}}]$ is bounded by $O(n + e)$. Counting the number of connected components and testing it are done in $O(n)$. Hence, this step is done in $O(n + e)$. Since, in the worst case, each vertex is in a different level, this step is done at most n times that is to say a cost of $O(n(n + e))$ globally.
  - In *H_4-TD-WT*, similarly to *H_3-TD-WT*, the search of vertices of level l is bounded by $O(n + e)$. The computation of the connected components of $G[X_i \backslash E_{i_{l-1}}]$ and their associated separators is bounded by $O(n + e)$. Thus, the global cost of this step is bounded by $O(n(n + e))$.
- Lines 9 and 10: each of these steps is done in $O(n)$, that is to say $O(n^2)$ globally.
- Line 11: the cost of computing the connected compo-

nents of $G[X_i \backslash E_i]$ is bounded by $O(n+e)$. Thus the global cost of this step is $O(n(n+e))$.

- Line 12: the insertion of X_{i_j} in F is done in $O(n)$, that is to say $O(n^2)$ globally knowing that the number of insertions is bounded by n.

Finally, the time complexity of the algorithm is $O(n(n+e))$ for all the proposed heuristics *H_1-TD-WT*, *H_2-TD-WT*, *H_3-TD-WT* and *H_4-TD-WT*. Nonetheless, the complexity of the algorithm depends on the complexity of the used heuristic and thus a more costly heuristic increases the complexity of algorithm. □

IV. Experiments

In this section, we assess the practical interest of the tree-decompositions produced by *Min-Fill* and by *H-TD-WT*. First, we are interested in minimizing the width and so the comparison involves *Min-Fill* and *H_1-TD-WT*. Then, we consider as criterion the efficiency of the solving and so we compare *Min-Fill* with *H_2-TD-WT*, *H_3-TD-WT* and *H_4-TD-WT*.

For the choice of the first cluster E_0 in *H-TD-WT*, many heuristics can be used. Here, we compute greedily a clique, as large as possible, of G. The heuristic begins by choosing the vertex having the highest degree in the graph G. Given a set of already chosen neighbors N, it selects the vertex having the highest degree in the neighborhood of N.

All the algorithms are implemented in C++ in our own library. The experiments were performed on blade servers running Linux Ubuntu 14.04 each with two Intel Xeon processors E5-2609 2.4GHz and with 32 GB of memory. We consider 1,859 CSP instances from the CSP 2008 competition[2]. Note that, when an instance has non-binary constraints, we exploit the 2-section of its constraint hypergraph to compute tree decompositions.

A. Minimizing the width

As we try to minimize the width of the decompositions, we use the first heuristic, that is *H_1-TD-WT*. Table I provides the widths of the decompositions obtained for a selection of representative instances of the various observed trends. *H_1-TD-WT* computes decompositions having a width less than or equal to those of *Min-Fill* for 1,031 of the 1,859 instances. For 772 of these instances, *H_1-TD-WT* improves the width obtained by *Min-Fill*. This improvement can be very significant as it is the case, for example, with the instance `bqwh-18-141-37_ext` with a width of 54 for *H_1-TD-WT* against 73 for *Min-Fill*.

Regarding the runtime, *H_1-TD-WT* is faster than *Min-Fill*. In fact, in 86% (respectively 98%) of the instances, the decomposition of *H_1-TD-WT* is computed in less than one second (respectively one minute) against 85% (respectively 96%) for *Min-Fill*. These results are not surprising regarding the complexity of the algorithms.

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

Table I
Number of vertices and edges, width of decompositions produced by *Min-Fill* and *H_1-TD-WT*. For each instance, the best width is in bold.

Instances	n	e	*Min-Fill*		*H_1-TD-WT*	
			time	w^+	time	w^+
composed-25-10-20-5_ext	105	620	0.01	**24**	0.01	28
graph14-f28	916	4 638	0.45	239	0.48	**229**
par-8-2	700	2 800	0.04	47	0.06	**41**
par-16-4-c	648	3 723	0.05	**43**	0.08	83
radar-10-20-4.5-0.95-98	906	10 211	0.20	**112**	0.60	121
s4-4-3-6	624	9 152	0.43	192	0.27	**177**
tsp-25-3_ext	76	400	0.01	**25**	0.01	**25**
bf-0432-007	2 080	7 473	0.79	145	6.47	**141**
ii-8e2	1 740	10 785	1.70	179	11.04	**163**
js-taillard-20-15-95-2	300	3 130	0.06	117	0.04	**109**
4-insertions-4-5	475	1 795	0.07	94	0.14	**81**
fapp04-0300-1	300	1 799	0.07	134	0.03	**123**
bqwh-18-141-37_ext	141	883	0.01	73	0.01	**54**
graph4	400	2 244	0.05	**100**	0.05	109
ssa-0432-003	870	2 022	0.12	**34**	0.46	65
games120-7	120	638	0.01	**39**	0.01	43

B. Solving CSPs

In this subsection, we compare the described tree-decompositions from the viewpoint of the solving efficiency. We compare especially *Min-Fill* to *H_2-TD-WT*, *H_3-TD-WT* and to *H_4-TD-WT*. Regarding *H_1-TD-WT*, even if the results show a significant improvement for the approximation of the tree-width, its impact on solving efficiency is quite limited. This first heuristic seems clearly more suited to handle more difficult problems such as optimization, counting or compilation where the quality of the approximation of w is clearly more crucial than for decision. Hence, we will focus in the following on the other heuristics.

Regarding the solving, as the reference enumerative method, we take into account the algorithm MAC (for Maintaining Arc-Consistency [20]) with restarts and nogood recording like in [21]. For the solving with the help of the tree-decomposition, we consider BTD (for Backtracking with Tree-Decomposition [11]) integrating restart techniques and nogood recording [22]. Both algorithms rely on AC-2001 and exploits the variable ordering heuristic dom/wdeg [23]. They use a geometric restart policy with a ratio 1.1 and an initial number of backtracks of 100 for MAC and 50 for BTD. BTD chooses as root cluster the cluster maximizing the sum of weights of constraints whose scope intersects the cluster (the weights are those of dom/wdeg) like in [22]. The solvings are performed with a time-out of 900 seconds.

The heuristic *H_2-TD-WT* has been introduced in [8] with several variants depending on the choice of the next vertex. Here, we choose as next vertex the vertex which has the maximum number of neighbors in the set V_i (what corresponds to the heuristic NV4 in [8]). The results show that bag-connected tree-decompositions computed by this heuristic allow an increase in the number of solved instances. In fact, only 10% of the tree-decompositions (of the 1,859

used instances) computed by *Min-Fill* are bag-connected. As we have already seen in [8], the presence of disconnected clusters has a negative impact on the efficiency of the solving. Thus, the construction of connected clusters helps in increasing the number of solved instances. In particular, we see that 1,429 instances are solved by *Min-Fill* while 1,453 instances are solved if the tree-decomposition is bag-connected. The heuristic H_3*-TD-WT* shows also its practical efficiency. More precisely, 1,479 instances of 1,859 instances are solved with this heuristic. The results prove that computing tree-decompositions by exploiting the topological properties allows better efficiency in the solving. Finally, the heuristic H_4*-TD-WT* focuses on limiting the size of the produced separators. We observe a significant increase of the number of solved instances for the values chosen for the separator. 1,511 instances are solved when the parameter S is set to 15. Obviously the chosen value of S has an impact on the quality of the decomposition and thereby on the efficiency of the solving. Nevertheless, choosing a relevant value for S is beyond the scope of this work. Note that these results are consistent with ones of [24], which advocates the limitation the size of separators in order to achieve a more efficient solving.

In order to fairly compare the quality of tree-decompositions w.r.t. the solving efficiency, we apply the merging strategy of [24] on tree-decompositions achieved by *Min-Fill*, H_2*-TD-WT* and H_3*-TD-WT*. We limit the size of separators to 15. Thus, any cluster of the decomposition whose separator has a size greater than 15 is merged with this parent. This process is repeated until all the separators have at most a size of 15. As we see in figure 3(a), all the methods solve more instances, but the trend remains the same as before. More precisely, *Min-Fill* solves 1,478 instances while H_2*-TD-WT* solves 1,497 instances. Also, H_3*-TD-WT* is improved with a total of 1,499 solved instances higher than both previously mentioned heuristics. Finally, H_4*-TD-WT* outperforms again the other heuristics with 1,511 solved instances. Note that the increase of the number of solved instances comes with an improvement of the runtime. In particular, the cumulative runtime achieved for *Min-Fill*, H_2*-TD-WT*, H_3*-TD-WT* and H_4*-TD-WT* is, respectively, 45,082 s, 40,058 s, 33,905 s and 31,263 s. We also compare these heuristics with MAC and the virtual Best Solver (VBS) which corresponds to the best runtime among the runtime of MAC and BTD with the different decompositions. We can observe that MAC is outperformed by BTD with any H_i*-TD-WT* decomposition with 1,491 solved instances. Nonetheless, as expected, MAC performs better than *Min-Fill* on this bench. In order to highlight the efficiency of solving of these heuristics on instances having nice topological properties, we select the instances s.t. $\frac{n}{w^+} \geq 5$ among the 1,859 instances. We particularly see in figure 3(b) that *Min-Fill* performs better on these 292 instances thanks to its good approximation of the tree-width. Hence, *Min-Fill* solves more instances than H_2*-TD-WT*, H_3*-TD-WT* and MAC which solves the lowest number of instances. However, we can also see that H_4*-TD-WT* still performs better than *Min-Fill*. More precisely, H_4*-TD-WT* solves 265 instances within 7,467 s whereas *Min-Fill* solves 260 instances in 10,717 s.

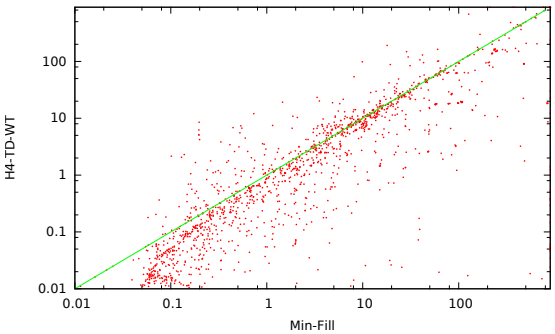

Figure 4. The comparison between state of the art heuristic *Min-Fill* and the proposed heuristic H_4*-TD-WT*

Since *Min-Fill* is the reference method of decomposition in the literature, the comparison between *Min-Fill* and H_4*-TD-WT* seems quite interesting. That is why we compare both methods performance on the initial benchmark in figure 4. It is clear that H_4*-TD-WT* dominates *Min-Fill* in most solving. Moreover, H_4*-TD-WT* constructs the tree-decomposition by one pass while the comparable tree-decomposition computed by *Min-Fill* needs another pass to have an appropriate separator's size.

Finally, the results of H_4*-TD-WT* seem very promising regarding the results achieved by the VBS. In fact, we can see that the VBS solves only 14 additional instances. Moreover, H_4*-TD-WT* seems also competitive on runtime.

V. Conclusion

In this paper, we introduced an algorithmic framework for network decomposition aiming to improve *Min-Fill*. It computes tree-decompositions based on a traversal of the graph using properties related to separators and their associated connected components. This algorithmic framework permits the implementation of different heuristics. This allows to take into account and thus to avoid some disadvantages of *Min-Fill* such as a very large size of the clusters or of the separators, or the non-connectivity of the clusters. Experimental evaluations allowed us to show that one of the implemented heuristics has improved the quality of decompositions w.r.t. the width, on a majority of instances, often significantly. In addition, in this case, the computation times are also improved. Regarding the solving of CSPs, we have shown that one of the implemented heuristics improves the efficiency of the search in practice.

For future investigations, there is a wide field of work to explore, in particular because of all the parameters to

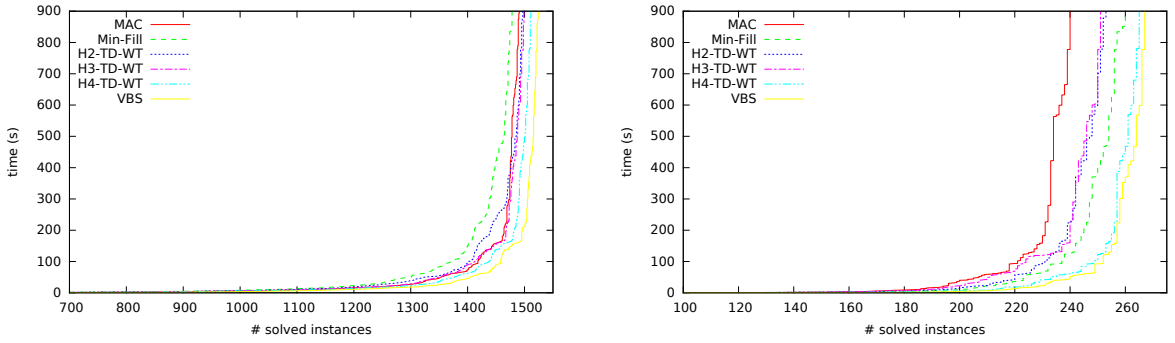

Figure 3. The cumulative number of solved instances for each considered tree-decomposition for the 1,859 instances (a), only for instances having a tree-decomposition of width w^+ s.t. $\frac{n}{w^+} \geq 5$ (b).

be taken into account in the elaboration of the heuristics, particularly to build the first cluster. Furthermore, the new algorithmic framework for decomposition proposed here must be now evaluated w.r.t. to the improvement of solving more difficult questions than of basic decision problem for CSPs. In fact, the most promising issue is related to significantly harder problems such as counting, optimization or knowledge compilation where minimizing the width or the size of the separators is a crucial issue.

REFERENCES

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

[2] C. Berge. *Graphs and Hypergraphs*. Elsevier, 1973.

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

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

[5] 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. Academic Press, 1972.

[6] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:243–282, 2000.

[7] A. Dermaku, T. Ganzow, G. Gottlob, B. J. McMahan, N. Musliu, and M. Samer. Heuristic methods for hypertree decomposition. In *MICAI*, pages 1–11, 2008.

[8] P. Jégou and C. Terrioux. Tree-decompositions with connected clusters for solving constraint networks. In *CP*, pages 407–423, 2014.

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

[10] R. Dechter. Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

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

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

[15] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *UAI*, pages 201–208, 2004.

[16] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *UAI*, pages 7–15, 2001.

[17] J. Berg and M. Järvisalo. Sat-based approaches to treewidth computation: An evaluation. In *ICTAI*, pages 328–335, 2014.

[18] U. Kjaerulff. Triangulation of graphs - algorithms giving small total state space. Technical report, Judex R.R. Aalborg, Denmark, 1990.

[19] S. Subbarayan. An empirical comparison of csp decomposition methods. In *CP Doctoral Program*, 2007.

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

[21] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Recording and minimizing nogoods from restarts. *JSAT*, 1(3-4):147–167, 2007.

[22] Philippe Jégou and Cyril Terrioux. Combining restarts, nogoods and decompositions for solving csps. In *ECAI*, pages 465–470, 2014.

[23] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, pages 146–150, 2004.

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