Constraints (2017) 22:191–229
DOI 10.1007/s10601-016-9248-8

Combining restarts, nogoods and bag-connected decompositions for solving CSPs

Philippe Jégou[1] · Cyril Terrioux[1]

Published online: 2 August 2016

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 or AI) has concentrated its efforts on the minimization of a single parameter, namely the tree-width. Nevertheless, experimental studies have shown that this parameter is not always the most relevant to consider when solving CSPs. So, in this paper, we highlight two fundamental problems related to the use of tree-decomposition and for which we offer two particularly appropriate solutions. First, we experimentally show that the decomposition algorithms of the state of the art produce clusters (a tree-decomposition is a rooted tree of clusters) having several connected components. We highlight the fact that such clusters create a real disadvantage which affects significantly the efficiency of solving methods. To avoid this problem, we consider here a new graph decomposition called *Bag-Connected Tree-Decomposition*, which considers only tree-decompositions such that each cluster is connected. We analyze such decompositions from an algorithmic point of view, especially in order to propose a first polynomial time algorithm to compute them. Moreover, even if we consider a very well suited decomposition, it is well known that sometimes, a bad choice for the root cluster may significantly degrade the performance of the solving. We highlight an explanation of this degradation and we propose a solution based on restart techniques. Then, we present a new version of the BTD algorithm (for Backtracking with Tree-Decomposition Jégou and Terrioux, *Artificial Intelligence, 146* 43–75 2003) integrating restart techniques. From a theoretical viewpoint, we prove that reduced nld-nogoods

This paper is an extension of the works published in [29, 30].

✉ Cyril Terrioux
cyril.terrioux@univ-amu.fr

Philippe Jégou
philippe.jegou@lsis.org

1 Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, 13397 Marseille, France

can be safely recorded during the search and that their size is smaller than ones recorded by MAC+RST+NG (Lecoutre et al., *JSAT, 1(3–4)* 147–167 2007). We also show how structural (no)goods may be exploited when the search restarts from a new root cluster. Finally, from a practical viewpoint, we show experimentally the benefits of using independently bag-connected tree-decompositions and restart techniques for solving CSPs by decomposition methods. Above all, we experimentally highlight the advantages brought by exploiting jointly these improvements in order to respond to two major problems generally encountered when solving CSPs by decomposition methods.

Keywords Constraint satisfaction problems · Tree-decomposition · Bag-connected tree-decomposition · Restart

1 Introduction

Constraint Satisfaction Problems (CSPs, see [41] 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*, also called *constraint network*, 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 $C = \{C_1, \ldots, C_m\}$ is a finite set of m 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* that contains assignments of variables of the scope which satisfy the constraint C_i. 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 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 $(X, \{S(C_1), \ldots S(C_m)\})$ 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 the time complexity of backtracking algorithms which are usually exploited to solve CSPs, is naturally exponential, at least in $O(m.d^n)$ where d is the size of the largest domain.

Many works have been realized to make the solving more efficient in practice, by using, for example, optimized backtracking algorithms, heuristics, constraint learning, non-chronological backtracking or filtering techniques [41]. In order to ensure an efficient solving, most solvers commonly exploit jointly several of these techniques. Moreover, often, they also derive benefit from the use of restart techniques [19, 23]. In particular, restart techniques generally allow to reduce the impact of bad choices performed thanks to heuristics (like the variable ordering heuristic) or of the occurrence of heavy-tailed phenomena [19]. For efficiency reasons, they are usually exploited with some learning techniques (like recording of *nld-nogoods* in [34]).

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 [16]. Using and generalizing these theoretical results, some methods to solve CSPs have been defined, such as Tree-Clustering [11] and other methods that have improved this original approach (like BTD [28]). This kind of methods is based on the notion of *tree-decomposition of graphs* [39], roughly speaking, a tree of subsets (called *clusters*) of variables. Their advantage is related to their theoretical complexity, that is d^{w+1} where w is the *tree-width* of the constraint graph, that is the size of the larger cluster minus one. When this graph has nice topological properties and thus when w is small, these methods allow to solve large and hard 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 [1]. 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 as efficient as possible. 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 [24].

In this paper, we show that this lack of efficiency for solving CSPs using decomposition can be explained by the nature of the decompositions for which w^+ is close to w. Indeed, minimizing w^+ can produce decompositions such that some clusters have several connected components. Unfortunately, this lack of connectedness may lead the solving method to spend a 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* [36]. 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 Bag-Connected Tree-Decompositions. The notion of Bag-Connected Tree-Width has been introduced very recently in [36] and to date, only studied from a mathematical viewpoint [13, 22, 36] without any perspective to be used in practice. Here we analyze this concept in terms of its algorithmic properties. So, we firstly prove that its computation is NP-hard. Then, we propose a first polynomial time algorithm (in $O(n(n+m))$) 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.

Moreover, if the use of a well suited decomposition for solving a constraint network is necessary to ensure some practical efficiency, a second problem often arises. It concerns the choice of the root of the tree decomposition. Indeed, in [25], it has been shown that this choice plays a crucial role in ensuring the efficiency of the solving, in a similar manner to the choice of first variables to assign for the usual backtracking methods. Generally, this issue is dealt by a choice of the root cluster before starting the search. Of course, this solution can be promising but it imposes strong constraints on the ordering used for the assignment of the variables all along the search. Indeed, if this choice is not the most appropriate, the efficiency of search can be particularly deteriorated. In [27], an approach has been proposed to

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

choose a variable ordering with more freedom but its efficiency still depends on the choice of the root cluster. And this initial choice may be inappropriate for all the search. To overcome this difficulty, we introduce for the first time the restart techniques in the context of decomposition methods for solving CSPs. To describe this approach, we consider here the BTD method [28] which is a reference in the state of the art for decomposition methods [38]. Note that before presenting the implementation of the restarts in BTD, we give a detailed description of BTD-MAC which has never been described before in the literature. Indeed, previous implementations of BTD were in fact RFL-BTD, i.e. BTD based on Real Full Look-ahead [37] (see [43] for a comparison between MAC and RFL). While the implementation of restarts is not particularly difficult for backtracking algorithms, unless to ensure termination, for decomposition methods, additional difficulties arise. In particular, if we consider the use of the usual *nogoods* (as the *nld-nogoods* [34]), the difficulty that arises is related to a possible change of the structure of the decomposition. Moreover, the change of root can question the validity and therefore the use of nogoods. So, from a theoretical viewpoint, we prove that reduced nld-nogoods can be safely recorded during the search and that their size is smaller than ones recorded by MAC+RST+NG [34]. Moreover, since the practical efficiency of BTD is especially due to the use of *structural goods* and *structural nogoods* (which are induced by the considered decomposition), we need to analyze and adapt their management in case of restarts. We also show how structural (no)goods can be exploited when the search restarts from a new root cluster. To this end, we define here the notion of *oriented structural good*. From a practical viewpoint, we show experimentally the benefits of the use of restart techniques for solving CSPs by decomposition methods. Finally, we highlight experimentally the benefits of joint use of Bag-Connected Tree-Decompositions and restart techniques, showing that their joint use significantly improves the efficiency of search.

Note that the present work is applied to tree-decompositions, but it can also be adapted to most decompositions (e.g. Hypertree-Decomposition [20] or Hinge-Decomposition [21]). 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 recalls the principles of backtracking algorithms using nld-nogoods, and the principles of tree-decomposition methods for solving CSPs. This section also recalls the frame of BTD and describes in details the BTD-MAC algorithm. Section 3 points out some problems related to the computing of “good” tree-decompositions, i.e. for a given instance, find a suitable tree of clusters, and also, choose for this tree, a relevant root cluster. Section 4 presents the notion of bag-connected tree-decomposition, proposing a first algorithm to achieve one. Then, Section 5 presents the algorithm BTD-MAC+RST which introduces restarts in decompositions methods. In Section 6, we assess the benefits of restarts and bag-connected tree-decomposition when solving CSPs thanks to a decomposition-based method and we conclude in Section 7.

2 Background

In this section, we recall the necessary background about the solving of CSPs by backtracking methods or by methods exploiting tree-decompositions.

2.1 Solving CSPs by backtracking methods

In the past decades, many solvers have been proposed for solving CSPs. Generally, from a practical viewpoint, they succeed in solving efficiently a large kind of instances despite of the NP-completeness of the CSP decision problem. In most cases, they rely on optimized backtracking algorithms whose time complexity is at least in $O(m.d^n)$. In order to ensure an efficient solving, they commonly exploit jointly several techniques among which we can cite heuristics, constraint learning, non-chronological backtracking, or filtering techniques (see [41] for more details). For instance, most solvers of the state of the art maintain some consistency level at each step of the search, like MAC (Maintaining Arc-Consistency [42]) or RFL (Real Full Look-ahead [37]) do for arc-consistency.

We now recall MAC with more details. During the solving, MAC develops a binary search tree unlike RFL whose search tree corresponds to a d-way branching (see [43] for more details). More precisely, MAC can make two kinds of decisions:

- *positive decisions* $x_i = v_i$ which assign the value v_i to the variable x_i (we denote $Pos(\Sigma)$ the set of positive decisions in a sequence of decisions Σ),
- *negative decisions* $x_i \neq v_i$ which ensure that x_i cannot be assigned with v_i.

Let us consider $\Sigma = \langle \delta_1, \ldots, \delta_i \rangle$ (where each δ_j may be a positive or negative decision) as the current decision sequence. A new positive decision $x_{i+1} = v_{i+1}$ is chosen and an AC filtering is achieved. If no dead-end occurs, the search goes on by choosing a new positive decision. Otherwise, the value v_{i+1} is deleted from the domain $d_{x_{i+1}}$, and an AC filtering is realized. If a dead-end occurs again, we backtrack and change the last positive decision $x_\ell = v_\ell$ to $x_\ell \neq v_\ell$.

More recently, restart techniques have been introduced in the CSP framework (e.g. in [34]). They generally allow to reduce the impact of bad choices performed thanks to heuristics (like the variable ordering heuristic) or of the occurrence of heavy-tailed phenomena. For efficiency reasons, they are usually exploited with some learning techniques (like recording of nld-nogoods in [34]).

Before introducing the reduced nld-nogoods (for negative last decision nogoods), we first recall the notion of nogood:

Definition 1 ([34]) Given a CSP $P = (X, D, C)$ and a set of decisions Δ, $P_{|\Delta}$ is the CSP (X, D', C) with $D' = (d'_{x_1}, \ldots, d'_{x_n})$ such that for any positive decision $x_i = v_i$, $d'_{x_i} = \{v_i\}$ and for any negative decision $x_i \neq v_i$, $d'_{x_i} = d_{x_i} \backslash \{v_i\}$. Δ is a *nogood* of P if $P_{|\Delta}$ is inconsistent.

In the following, like in [34], we assume that for any variable x_i and value v_i, the positive decision $x_i = v_i$ is considered before the decision $x_i \neq v_i$. By so doing, nogoods can be used to represent some unfruitful part of the search tree, as stated in the following proposition.

Proposition 1 ([34]) *Let $\Sigma = \langle \delta_1, \ldots, \delta_k \rangle$ be the sequence of decisions taking along the branch of the search tree when solving a CSP P. For any subsequence $\Sigma' = \langle \delta_1, \ldots, \delta_\ell \rangle$ of Σ s.t. δ_ℓ is a negative decision, the set $Pos(\Sigma') \cup \{\neg\delta_\ell\}$ is a nogood (called a reduced nld-nogood) of P with $\neg\delta_\ell$ the positive decision corresponding to δ_ℓ.*

In other words, given a sequence Σ of decisions taking along the branch of a search tree, each reduced nld-nogood characterizes a visited inconsistent part of this search tree. When a restart occurs, an algorithm like MAC+RST+NG [34] can record several new reduced nld-nogoods and exploit them later to prevent from exploring again an already visited part of the search space. These nld-nogoods can be efficiently computed and stored as a global constraint with an efficient specific propagator for enforcing AC [34].

2.2 Solving CSPs using graph decomposition

From a historical point of view, Tree-Clustering [11] 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 [39].

Definition 2 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 w^+ of a tree-decomposition (E, T) is equal to $max_{i \in I}|E_i| - 1$. The *tree-width* w of G is the minimal width over all the tree-decompositions of G.

Figure 1b presents a tree-decomposition of the graph depicted in Fig. 1a. It is a possible tree-decomposition for this graph. So, we get $E_1 = \{x_1, x_2, x_3\}$, $E_2 = \{x_2, x_3, x_4, x_5\}$, $E_3 = \{x_3, x_4, x_5, x_6\}$, $E_4 = \{x_5, x_6, x_7\}$, $E_5 = \{x_3, x_8, x_9\}$, $E_6 = \{x_8, x_9, x_{10}\}$ and $E_7 = \{x_{10}, x_{11}\}$. One can see that the proposed tree satisfies the three conditions of a tree-decomposition. Moreover, the tree-width of this graph is 3 since this tree-decomposition has minimal width over all the tree-decompositions of the graph and because its maximum size of clusters is 4.

The first version of Tree-Clustering [11], begins by computing a tree-decomposition (using the algorithm *MCS* [45]). Note that the computed tree-decomposition is not necessarily optimal, that is its width may be different from w. Thus, for this width w^+ (the size of the largest cluster), we have $w + 1 \leq w^+ + 1 \leq n$. In the second step, the clusters are

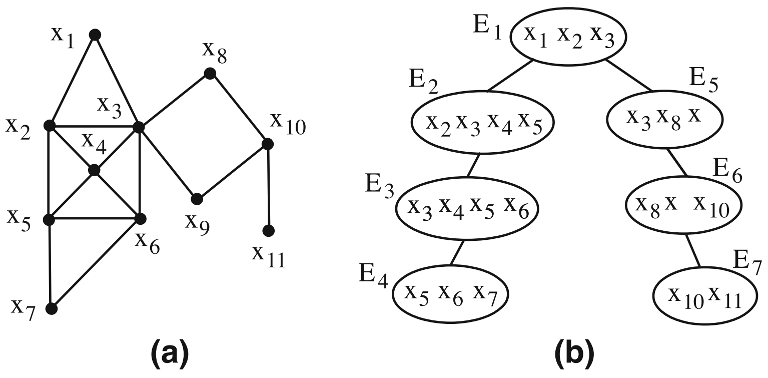

Fig. 1 A constraint graph on 11 variables (**a**) and an optimal tree-decomposition (**b**)

solved independently, considering each cluster as a subproblem, and then, enumerating all its solutions. The next step consists in building an acyclic CSP whose variables correspond to the clusters. Finally, a global solution of the initial CSP, if one exists, can be found efficiently by solving this acyclic CSP. The time and space complexities of this first version is $O(n.d^{w^+ +1})$. Note that this first approach has been improved to reach a space complexity in $O(n.s.d^s)$ [7, 8] 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 [28]) 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 [38]. While this approach (which will be described in details in the next subsection) has shown its practical interest, from a theoretical viewpoint, in the worst case, it has the same complexities as the improved version of Tree-Clustering (e.g. [7, 8]), 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 [1]. So, many works deal with this problem. They often exploit an algorithmic approach related to *triangulated* graphs which are also called *chordal* graphs. An undirected graph is called triangulated if every cycle of length strictly greater than 3 possesses a *chord*, that is, an edge joining two nonconsecutive vertices in the cycle (see [18] for an introduction to triangulated graphs). For example, if we consider the graph given in Fig. 1, one can associate a triangulated graph induced by the addition of two new edges (depicted with dotted lines in Fig. 2) which join two nonconsecutive vertices in the cycles whose length is greater than 3 (cycles $[x_2, x_3, x_6, x_5, x_2]$ and $[x_3, x_8, x_{10}, x_9, x_3]$). Note that the maximal cliques of this triangulated graph correspond to the clusters of the depicted tree-decomposition.

To compute tree-decompositions, one can distinguish different classes of approaches based on triangulated graphs. 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

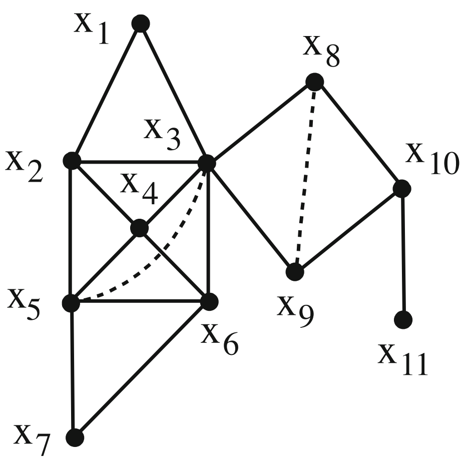

Fig. 2 The constraint graph of Fig. 1 after a triangulation

[11, 24, 31]. They run in polynomial time (between $O(n+m)$ 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 [32]. In practice, the most used methods to find tree-decompositions are based on *MCS* [45] and *Min-Fill* [40] which give good approximations of w^+. Moreover, in [24], 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 sometimes s), not on the structure of clusters which seems to be a 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 the *Min-Fill* heuristic computes a tree-decomposition. The first step 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 is 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*. For example, if we consider the graph given in Fig. 1, a possible order found by *Min-Fill* is $[x_1, x_7, x_{11}, x_{10}, x_8, x_9, x_2, x_3, x_4, x_5, x_6]$. So, when the vertex x_{10} is numbered, *Min-Fill* adds the edge $\{x_8, x_9\}$ while when the vertex x_2 is numbered, *Min-Fill* adds the edge $\{x_3, x_5\}$. So, one can see in Fig. 2 that $[x_1, x_7, x_{11}, x_{10}, x_8, x_9, x_2, x_3, x_4, x_5, x_6]$ is 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 $O(n + m')$ where $m' = |C \cup C'|$ [17, 18]. Each maximal clique corresponds to a cluster of the associated tree-decomposition. In the example in Fig. 2, the maximal cliques are $\{x_1, x_2, x_3\}$, $\{x_2, x_3, x_4, x_5\}$, $\{x_3, x_4, x_5, x_6\}$, $\{x_5, x_6, x_7\}$, $\{x_3, x_8, x_9\}$, $\{x_8, x_9, x_{10}\}$ and $\{x_{10}, x_{11}\}$ which correspond to the clusters of the 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 $O(n^3)$.

2.3 Backtracking on tree-decomposition: the BTD method

It is well known that the backtracking methods (with additional improvements described above) can be really efficient in practice, even if they do not give guarantee with respect to the time complexity in the worst case. In contrast, the decompositions methods have complexity bounds which can be significantly better, but sometimes at the expense of a good practical efficiency. Following these observations, the BTD method (for Backtracking on Tree-Decomposition) has been proposed to take advantage of both the practical efficiency of backtracking algorithms and complexity bounds of decomposition methods. We now describe BTD [28] in more detailed way. Given a tree-decomposition (E, T) and a root cluster E_r, we denote $Desc(E_j)$ the set of vertices (variables) belonging to the union of the descendants E_k of E_j in the tree rooted in E_j, E_j included. As indicated before, Fig. 1b presents a possible tree-decomposition of the graph depicted in Fig. 1a, whose root is E_1 and such that $Desc(E_1) = X$, $Desc(E_2) = E_2 \cup E_3 \cup E_4 = \{x_2, x_3, x_4, x_5, x_6, x_7\}$, $Desc(E_3) = E_3 \cup E_4 = \{x_3, x_4, x_5, x_6, x_7\}$ and $Desc(E_4) = E_4 = \{x_5, x_6, x_7\}$.

Given a compatible cluster ordering $<$ (i.e. an ordering which can be produced by a depth-first traversal of T from the root cluster E_r), BTD achieves a backtrack search by using a variable ordering $\preceq$ (said *compatible*) s.t. $\forall x \in E_i$, $\forall y \in E_j$, with $E_i < E_j$, $x \preceq y$. In other words, the cluster ordering induces a partial ordering on the variables since the variables in E_i are assigned before those in E_j if $E_i < E_j$. For the example of Fig. 1, $E_1 < E_2 < E_3 < E_4 < E_5 < E_6 < E_7$ (respectively $x_1 \preceq x_2 \preceq x_3 \preceq \ldots \preceq x_{11}$) is a possible compatible ordering on E (respectively X). In practice, BTD starts its backtrack search by assigning consistently the variables of the root cluster E_r before exploring a child cluster. When exploring a new cluster E_i, since the variables in the parent cluster $E_{p(i)}$ (and so in the separator $E_i \cap E_{p(i)}$[2]) are already assigned, it only has to assign the variables which appear in $E_i \backslash (E_i \cap E_{p(i)})$.

In order to solve each cluster, BTD can exploit any solving algorithm which does not alter the structure. For instance, BTD can rely on the algorithm MAC (for Maintaining Arc-Consistency [42]). We denote BTD-MAC the version of BTD relying on MAC for solving each cluster. We can note that, in BTD-MAC, the next positive decision necessarily involves a variable of the current cluster E_i and that only the domains of the future variables in $Desc(E_i)$ can be impacted by the AC filtering (since $E_i \cap E_{p(i)}$ is a separator of the constraint graph and all its variables have already been assigned).

When BTD has consistently assigned the variables of a cluster E_i, it then tries to solve each subproblem rooted in each child cluster E_j. More precisely, for a child E_j and a current decision sequence Σ, it attempts to solve the subproblem induced by the variables of $Desc(E_j)$ and the decision set $Pos(\Sigma)[E_i \cap E_j]$ (i.e. the set of positive decisions involving the variables of $E_i \cap E_j$). Once this subproblem solved (by showing that there is a solution or showing that there is none), it records a structural good or nogood. Formally, given a cluster E_i and E_j one of its children, a *structural good* (resp. *nogood*) of E_i with respect to E_j is a consistent assignment A of $E_i \cap E_j$ such that A can (resp. cannot) be consistently extended on $Desc(E_j)$ [28]. In the particular case of BTD-MAC, the consistent assignment of A will be represented by the restriction of the set of positive decisions of Σ on $E_i \cap E_j$, namely $Pos(\Sigma)[E_i \cap E_j]$. These structural (no)goods can be used later in the search in order to avoid exploring a redundant part of the search tree. Indeed, once the current decision sequence Σ contains a good (resp. nogood) of E_i w.r.t. E_j, BTD has already proved previously that the corresponding subproblem induced by $Desc(E_j)$ and $Pos(\Sigma)[E_i \cap E_j]$ has a solution (resp. none) and so does not need to solve it again. In the case of a good, BTD keeps on the search with the next child cluster. In the case of a nogood, it backtracks. For example, let us consider a CSP on 11 variables $x_1, \ldots, x_{11}$ for which each domain is $\{a, b, c\}$ and whose constraint graph and a possible tree-decomposition are given in Fig. 1. Assume that the current consistent decision sequence $\Sigma = \langle x_1 = a, x_2 \neq b, x_2 = c, x_3 = b\rangle$ has been built according to a variable order compatible with the cluster order $E_1 < E_2 < E_3 < E_4 < E_5 < E_6 < E_7$. BTD tries to solve the subproblem rooted in E_2 and once solved, records $\{x_2 = c, x_3 = b\}$ as a structural good or nogood of E_1 w.r.t. E_2. If, later, BTD tries to extend the consistent decision sequence $\langle x_1 \neq a, x_3 = b, x_1 = b, x_2 \neq a, x_2 = c\rangle$, it keeps on its search with the next child cluster of E_1, namely E_4, if $\{x_2 = c, x_3 = b\}$ has been recorded

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

as a good, or backtracks to the last decision in E_1 if $\{x_2 = c, x_3 = b\}$ corresponds to as a nogood.

Algorithm 1 BTD-MAC (**InOut**: $P = (X, D, C)$: CSP; **In**: Σ: sequence of decisions, E_i: Cluster, V_{E_i}: set of variables; **InOut**: G: set of goods, N: set of nogoods)

```
if $V_{E_i} = \emptyset$ then
    $result \leftarrow true$
    $S \leftarrow Sons(E_i)$
    while $result = true$ and $S \neq \emptyset$ do
        Choose a cluster $E_j \in S$
        $S \leftarrow S \backslash \{E_j\}$
        if $Pos(\Sigma)[E_i \cap E_j]$ is a nogood in $N$ then $result \leftarrow false$
        else if $Pos(\Sigma)[E_i \cap E_j]$ is not a good of $E_i$ w.r.t. $E_j$ in $G$ then
            $result \leftarrow$ BTD-MAC($P$,$\Sigma$,$E_j$,$E_j \backslash (E_i \cap E_j)$,$G$,$N$)
            if $result = true$ then
                Record $Pos(\Sigma)[E_i \cap E_j]$ as good of $E_i$ w.r.t. $E_j$ in $G$
            else if $result = false$ then
                Record $Pos(\Sigma)[E_i \cap E_j]$ as nogood of $E_i$ w.r.t. $E_j$ in $N$
    return $result$
else
    Choose a variable $x \in V_{E_i}$
    Choose a value $v \in d_x$
    $d_x \leftarrow d_x \backslash \{v\}$
    if $AC$ ($P$,$\Sigma \cup \langle x = v \rangle$) then $result \leftarrow$ BTD-MAC($P$, $\Sigma \cup \langle x = v \rangle$, $E_i$, $V_{E_i} \backslash \{x\}$, $G$, $N$)
    else $result \leftarrow false$
    if $result = false$ then
        if $AC$ ($P$,$\Sigma \cup \langle x \neq v \rangle$) then $result \leftarrow$ BTD-MAC($P$,$\Sigma \cup \langle x \neq v \rangle$,$E_i$,$V_{E_i}$,$G$,$N$)
    return $result$
```

Algorithm 1 corresponds to the algorithm BTD-MAC. Initially, the current decision sequence Σ and the sets G and N of recorded structural goods and nogoods are empty and the search starts with the variables of the root cluster E_r. Given a current cluster E_i and the current decision sequence Σ, lines 16-23 consist in exploring the cluster E_i by assigning the variables of V_{E_i} (with V_{E_i} the set of unassigned variables of the cluster E_i) like MAC would do while lines 1-14 allow to manage the children of E_i and so to use and record structural (no)goods. BTD-MAC(P, Σ, E_i, V_{E_i}, G, N) returns $true$ if it succeeds in extending consistently Σ on $Desc(E_i) \backslash (E_i \backslash V_{E_i})$, $false$ otherwise. It has a time complexity in $O(n.s^2.m. \log(d).d^{w^+ +2})$ while its space complexity is $O(n.s.d^s)$ with w^+ the width of the used tree-decomposition and s the size of the largest intersection between two clusters.

The next section discusses some issues related to the computation of suitable tree-decompositions, both as regards the construction of the clusters, but also for the choice of the root cluster.

3 What impacts the efficiency of decomposition methods?

The fact that a constraint network has a small tree-width should allow, using a suitable decomposition, to take advantage of this topological property of the instance. However, besides the computation of a decomposition that well approximates an optimal decomposition, several problems may be encountered. Assuming that we have a very good approximation of an optimal tree-decomposition, a very important problem may arise due to the choice of the root. Indeed, the search will begin with the assignment of variables

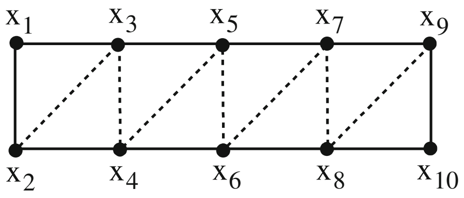

Fig. 3 Cycle without chord on $n = 10$ vertices with added edges (*dotted lines*) by a triangulation using *Min-Fill*

contained in this root cluster. Because of the imposed ordering to guarantee the complexity bounds, this ordering will not be changed all along the search. This point has already been discussed in the literature and the proposed solutions try to offer a little more freedom in that ordering, sometimes with significant improvements of the efficiency, but not systematically [25–27]. Note that this is similar to the goal of ordering heuristics for classical backtracking algorithms. However, for decomposition methods, this issue has never been quantified for truly identifying this problem and we will revisit it in this section. A second problem concerns the existence of clusters that may consist of several connected components. This can lead to the existence of subproblems circumscribed to such clusters that are too under-constrained. We develop first this second problem.

3.1 Tree-decompositions with disconnected clusters

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. For example, a triangulation using *Min-Fill* can find an optimal tree-decomposition for the graph given in Fig. 3. The order found by *Min-Fill* is given by the numbering of vertices. We get $n - 2 = 8$ clusters of size 3, whose two are connected ($\{x_1, x_2, x_3\}$ and $\{x_8, x_9, x_{10}\}$), while $n - 4 = 6$ clusters, $\{x_2, x_3, x_4\}$, $\{x_3, x_4, x_5\}, \ldots$ and $\{x_7, x_8, x_9\}$, have two connected components.

Such an example can be generalized to more complicated constraint graphs. Let us consider, for example, the graph whose a partial view is given in Fig. 4. We assume here that $x_8, x_9, x_{10}, \ldots x_{15}, \ldots$ have a large number of neighboring vertices which are not represented in the figure. So, a triangulation using *Min-Fill* can find an order which is compatible with the one given by the numbering of vertices. A such order induces the clusters $\{x_1, x_2, x_3\}$ which is connected, but also $\{x_2, x_3, x_4\}$, $\{x_3, x_4, x_5\}$, $\{x_4, x_5, x_6\}$, $\{x_4, x_5, x_6\}$ and $\{x_5, x_6, x_7\}$ which have two connected components. But such a triangulation finds the cluster $\{x_6, x_7, x_8, x_9, x_{10}\}$. Worse, we will find the cluster $\{x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}\}$. Of course, this example can be generalized to find disconnected clusters of larger size.

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[3] 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 $w^+ = 20$).

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

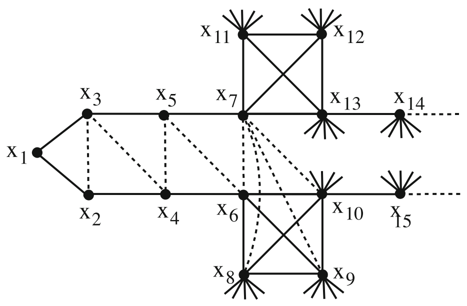

Fig. 4 A graph for which the triangulation using *Min-Fill* can induce clusters of arbitrarily large size

A detailed analysis of these tree-decompositions shows that they have 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, the fact that a constraint network is not connected can have important consequences on the efficiency of its solving. For example, if one of its connected 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 Tree-Clustering, 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 very high 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 while domains may have more than 100 values. 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. Consider again the constraint graph given in Fig. 4. Assume that the constraints whose scopes are $\{x_1, x_2\}$ and $\{x_1, x_3\}$ are equality constraints while all other constraints are constraints of differences. In this case, assuming that during the solving, the cluster $\{x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}\}$ will be solved before clusters containing variables located on the left in Fig. 4, a large number of local solutions will be considered before finding an incompatibility one time the equality constraints will be checked. This example can be generalized by the constraint graph given in Fig. 5 which

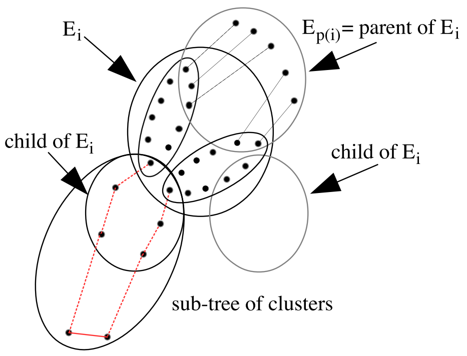

Fig. 5 Disconnected cluster in a Tree-Decomposition

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 Tree-Clustering in its last step. This leads Tree-Clustering to a large consumption of time and memory, making this approach unrealistic in practice.

To avoid this kind of phenomenon where clusters are initially solved independently, other methods were proposed like BTD. Although BTD has shown its practical advantage, unfortunately, the observed phenomenon still exists, even if it will generally be attenuated. To well understand this, 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 removes no or few values from 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 Figs. 4 and 5). 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 [42].

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

– if $G[E_i \backslash (E_i \cap E_{p(i)})]$ is connected: since E_i is a disconnected cluster, $G[E_i \cap E_{p(i)}]$ is necessarily disconnected. As the variables of the separator $E_i \cap E_{p(i)}$ 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 in the literature (e.g. in [24]). Indeed, even if the width is the same, decompositions computed by *Min-Fill* offer best results for solving than the ones obtained by *MCS* [24] and is considered as the best heuristic of the state of the art now [7]. 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.

The occurrence of the negative effect of the presence of disconnected clusters is confirmed by the following observation. If we consider a version of BTD using a less powerful filtering than AC like Forward Checking, we see that the phenomenon is accentuated. In order to illustrate this, we analyze the solving with a particular implementation of BTD using nFC5 [3] which is the most powerful non-binary version of Forward Checking. We can thus observed that, with $Min\text{-}Fill$, BTD-nFC5 only solves 520 instances over the 1,668 instances with disconnected clusters we will consider in Section 6 while BTD-MAC solves 1,167 instances (see Section 6 for more details about the experimental protocol). Moreover, if we consider the gap between the number of instances solved by BTD-MAC and BTD-nFC5 for a given decomposition method, we can note that this gap is reduced when exploiting the tree-decompositions with connected clusters introduced in the next section, namely between 485 and 499 instances against 647 with $Min\text{-}Fill$. This is explained by the fact that the use of a more powerful filtering as AC may lessen the phenomenon. Indeed the propagation is not limited to the neighborhood of the last assigned variable, but can reach the leaf clusters, and thus, to find some connectivity as we can see in Fig. 5. Nevertheless, this level of filtering may be ineffective if one considers the example of the constraint graph given in Fig. 4 with equality constraints on $\{x_1, x_2\}$ and $\{x_1, x_3\}$ and constraints of differences for all other constraints. Indeed, for this instance, the inconsistency appears on most cases, when x_2 and x_3 are assigned, and not when we are looking for a local solution in the cluster $\{x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}\}$.

To avoid this kind of phenomenon, in Section 4 we study classes of tree-decompositions for which all the clusters are connected.

3.2 The importance of the choice of the root

From a practical viewpoint, generally, BTD efficiently solves CSPs having a small tree-width. However, sometimes, a bad choice for the root cluster may drastically degrade the performance of the solving. The choice of the root cluster is crucial since it impacts on the variable ordering, in particular on the choice of the first variables. Hence, in order to make a smarter choice, we have selected some instances of the CSP 2008 Competition and, for each instance, we run BTD from each cluster of its considered tree-decomposition. First, we have observed that for a given instance, the runtimes may differ from several orders of magnitude according to the chosen root cluster. For instance, for the scen11-f12 instance (which is the easiest instance of the scen11 family), BTD succeeds in proving the inconsistency for only 75 choices of root cluster among the 301 possible choices. Secondly, we have noted that solving some clusters (not necessarily the root cluster) and their corresponding subproblems is more expensive for some choice of the root cluster than for another. This is explained by the choice of the root cluster which induces some particular ordering on the clusters and the variables. In particular, since for a cluster E_i, BTD only considers the variables of $E_i \backslash (E_i \cap E_{p(i)})$, it does not handle the same variable set for E_i depending on the chosen root cluster. Unfortunately, it seems to be utopian to propose a choice for the root cluster based only on features of the instance to solve because this choice is too strongly related to the solving efficiency. In [27], an approach has been proposed to choose a variable ordering with more freedom but its efficiency still depends on the choice of the root cluster. So, an alternative to limit the impact of the choice of the cluster is required. In Section 5, we propose a possible one consisting in exploiting restart techniques.

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 [36]. 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. Müller 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 3 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 [36] 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). This bound has been improved in [22]. Nevertheless, note that $w_c = \lceil \frac{n}{2} \rceil$ for graphs defined by cycles of length n and without chord. But 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,

[5]Note that we use the term of Bag-Connected Tree-Width rather than one of Connected Tree-Width exploited in [36] because the term of Connected Tree-Width has been introduced before in [15] 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.

necessarily appear below Tree-Decomposition methods in the constraint tractability hierarchy introduced in [20]. 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 consequences 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.

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. 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 2 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. This heuristic may rely on any criteria as soon as it produces a connected cluster. 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

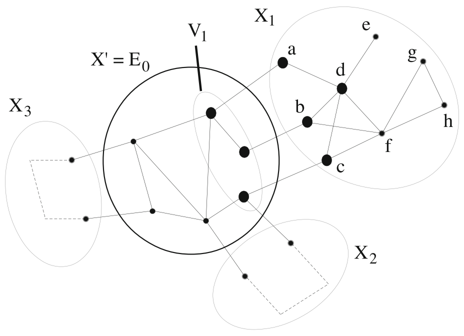

Fig. 6 First pass in the loop for *Bag-Connected-TD*

$V_i \subseteq X$ 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 6 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 Fig. 6, 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 $\{f, g, h\}$.

Algorithm 2 *Bag-Connected-TD*

```
   Input: A graph $G = (X, C)$
   Output: A set of clusters $E_0, \dots E_q$ 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     $F \leftarrow F \cup \{X_{i_1}, X_{i_2}, \dots X_{i_{k_i}}\}$
```

Note that 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 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 $E_0, E_1, \dots E_q$ 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 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 Lines 1-4 are feasible in linear time, that is $O(n+m)$, since the cost of computing the connected components of $G[X \backslash E_0]$ is bounded by $O(n+m)$. Nevertheless, we can note that line 1 can be done by a more expensive heuristic to get a more relevant first cluster, but at most in $O(n(n + m))$ 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 are 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 $O(n)$, thus globally $O(n^2)$.
- Line 7: obtaining the neighborhood $V_i \subseteq X'$ of X_i in G is bounded by $O(n+m)$, thus globally by $O(n(n+m))$.
- Line 8: this step is feasible in $O(n)$, thus globally $O(n^2)$.
- 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 $O(n+m)$. Since the while loop runs at most n times, the global cost of the search in these subgraphs is bounded by $O(n(n+m))$. Moreover, for each new added vertex x, the connectivity of $G[E_i]$ is tested with an additional cost bounded by $O(n+m)$. Note since such a vertex is added at most one time, globally, the cost of this test is bounded by $O(n(n+m))$. So, the cost of line 9 is globally bounded by $O(n(n+m))$.
- Line 10: using an efficient data structure, this step can be realized in $O(n)$, thus globally $O(n^2)$.
- Line 11: the union is feasible in $O(n)$, thus globally $O(n^2)$ since there are at most n iterations.
- Line 12: the cost of finding the connected components of $G[X_i \backslash E_i]$ is bounded by $O(n+m)$. So, globally, the cost of this step is $O(n(n+m))$.
- Line 13: the insertion of a set X_{i_j} in F is feasible in $O(n)$, thus globally $O(n^2)$ since there are less than n insertions in F.

Finally, the time complexity of the algorithm *Bag-Connected-TD* is $O(n(n+m))$. □

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 Section 6. 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 6.

5 Exploiting restarts within BTD

In this section, we explain how BTD can safely exploit restarts.

It is well known that any method exploiting restart techniques must as much as possible avoid exploring the same part of the search space several times and that randomization and learning are two possible ways to reach this aim [33]. Regarding the learning, BTD already exploits structural (no)goods. However, depending on when the restart occurs, we have no warranty that a structural (no)good has been recorded yet. Hence, another form of learning is required to ensure a good practical efficiency. Here, we consider the reduced nld-nogoods [34].

The use of learning in BTD may endanger its correctness as soon as we add to the initial problem a constraint whose scope is not included in a cluster. So recording reduced nld-nogoods in a global constraint involving all the variables like proposed in [34] is impossible. However, by exploiting the features of a compatible variable ordering, Property 2 shows that this global constraint can be safely decomposed in a global constraint per cluster E_i.

Proposition 2 *Let $\Sigma = \langle \delta_1, \dots, \delta_k \rangle$ be the sequence of decisions taking along the branch of the search tree when solving a CSP P by exploiting a tree-decomposition (E, T) and a compatible variable ordering. Let $\Sigma[E_i]$ be the subsequence built by considering only the decisions of Σ involving the variables of E_i. For any prefix subsequence $\Sigma'_{E_i} = \langle \delta_{i_1}, \dots, \delta_{i_\ell} \rangle$ of $\Sigma[E_i]$ s.t. δ_{i_ℓ} is a negative decision, and every variable in $E_i \cap E_{p(i)}$ appears in a decision in $Pos(\Sigma'_{E_i})$, the set $Pos(\Sigma'_{E_i}) \cup \{\neg\delta_{i_\ell}\}$ is a reduced nld-nogood of P.*

Proof Let P_{E_i} be the subproblem induced by the variables of $Desc(E_i)$ and Δ_{E_i} the set of the decisions of $Pos(\Sigma_{E_i})$ related to the variables of $E_i \cap E_{p(i)}$. As $E_i \cap E_{p(i)}$ is a separator of the constraint graph, $P_{E_i|\Delta_{E_i}}$ is independent from the remaining part of the problem P. Let us consider $\Sigma[E_i]$ the maximal subsequence of Σ which only contains decisions involving variables of E_i. According to Proposition 1 applied to $\Sigma[E_i]$ and $P_{E_i|\Delta_{E_i}}$, $Pos(\Sigma'_{E_i}) \cup \{\neg\delta_{i_\ell}\}$ is necessarily a reduced nld-nogood. □

It ensues that we can bound the size of produced nogoods and compare them with those produced by Proposition 1:

Corollary 1 *Given a tree-decomposition of width w^+, the size of reduced nld-nogood produced by proposition 2 is at most $w^+ + 1$.*

Corollary 2 *Under the same assumptions as Proposition 2, for any reduced nld-nogood Δ produced by Proposition 1, there is at least one reduced nld-nogood Δ' produced by Proposition 2 s.t. $\Delta' \subseteq \Delta$.*

Proof Let $\Sigma' = \langle \delta_1, \dots, \delta_\ell \rangle$ be a subsequence of Σ s.t. δ_ℓ is a negative decision. From Proposition 1, it follows that $\Delta = Pos(\Sigma') \cup \{\neg\delta_\ell\}$ is a reduced nld-nogood. Now let us consider a cluster E_i s.t. the variable x_ℓ related to δ_ℓ belongs to E_i and $E_i \cap E_{p(i)} \subseteq Pos(\Sigma')$. There exists necessarily such a cluster by construction of Σ. Then, from Proposition 2, it follows that $\Delta' = Pos(\Sigma'[E_i]) \cup \{\neg\delta_\ell\}$ is a reduced nld-nogood. As $Pos(\Sigma'[E_i]) \subseteq Pos(\Sigma')$, we have $\Delta' \subseteq \Delta$. □

BTD already exploits a particular form of learning by recording structural (no)goods. Any structural (no)good of a cluster E_i w.r.t. to a child cluster E_j is by definition oriented from E_i to E_j. This orientation is directly induced by the choice of the root cluster. When a restart occurs, BTD may choose a different cluster as root cluster. If so, we have to consider structural (no)goods with different orientations. Proposition 3 states how these structural (no)goods can be safely exploited when BTD uses the restart technique.

Proposition 3 *A structural good of E_i w.r.t. E_j can only be used if the choice of the current root cluster induces that E_j is a child cluster of E_i. A structural nogood of E_i w.r.t. E_j can be used whatever the choice of the root cluster.*

Proof Let us consider a good Δ of E_i w.r.t. E_j produced for a root cluster E_r. By definition of structural goods, the subproblem $P_{E_j|\Delta}$ has a solution and its definition only depends on Δ and the fact that E_j is a child cluster of E_i. So, for any choice of the root cluster s.t. E_j is a child cluster of E_i, Δ will be a structural good of E_i w.r.t. E_j and can be used to prune safely redundant part of the search. Now, if, due to the choice of the root cluster, E_j is the parent cluster of E_i, a good Δ of E_i w.r.t. E_j gives no information about the existence of a solution on the subproblem rooted in E_i and so cannot be safely used. Indeed, as counter-example, it suffices to consider a decomposition with three clusters E_i, E_j and E_k s.t. the cluster E_k has no solution and E_i is the parent cluster of E_j and E_k. Assume that we first consider E_i as the root cluster and by so doing we produce a good Δ of E_i w.r.t. E_j. Then, if we choose E_j as the root cluster, exploiting Δ leads to conclude that Δ can be consistently extended on $E_i \cup E_k$, what is impossible by definition of E_k.

Regarding structural nogoods, any structural nogood Δ of E_i w.r.t. E_j is a nogood and so any decision sequence Σ s.t. $\Delta \subseteq Pos(\Sigma)$ cannot be extended to a solution, independently from the choice of the root cluster. Hence, structural nogoods can be used regardless the choice of the root cluster. □

It follows that unlike the nogoods, for the goods, the orientation is required. So, it could be better to call them *oriented structural goods*.

Algorithm 4 describes the algorithm BTD-MAC+RST which exploits restart techniques jointly with recording reduced nld-nogoods and structural (no)goods. Exploiting the restart techniques can be seen as choosing a root cluster (line 3) and running a new instance of BTD-MAC+NG (line 4) at each restart until the problem is solved by proving there is a solution or none. Algorithm 3 presents the algorithm BTD-MAC+NG. Like BTD-MAC, given a current cluster E_i and the current decision sequence Σ, BTD-MAC+NG explores the cluster E_i (lines 16-29) by assigning the variables of V_{E_i} (with V_{E_i} the set of unassigned variables of E_i). When E_i is consistently assigned, it manages the children of E_i and so uses and records structural (no)goods (lines 1-14). The used structural (no)goods may have been recorded during the current call to BTD-MAC or during a previous one. Indeed, if the first call of BTD-MAC+NG is achieved with empty sets G and N of structural goods and nogoods, G and N are not reset at each restart. Note that their uses (lines 7-8) are performed according to Proposition 3. Then, unlike BTD-MAC, BTD-MAC+NG may stop its search as soon as a restart condition is reached (line 23). If so, it records reduced nld-nogoods w.r.t. the decision sequence Σ restricted to the decisions involving variables of E_i (line 24) according to Proposition 2. We consider that a global constraint is associated to each cluster E_i to handle the nld-nogoods recorded w.r.t. E_i and that their use is performed via a specific propagator when the arc-consistency is enforced (lines 19 and 27) like in [34]. The restart condition may involve some global parameters (e.g. the number of backtracks achieved since the begin of the current call to BTD-MAC+NG), some local ones (e.g. the number of backtracks performed in the current cluster or the number of recorded structural (no)goods) or a combination of these two approaches.

Algorithm 3 BTD-MAC+NG (**InOut**: $P = (X, D, C)$: CSP; **In**: Σ: sequence of decisions, E_i: Cluster, V_{E_i}: set of variables; **InOut**: G: set of goods, N: set of nogoods)

```
1  if $V_{E_i} = \emptyset$ then
2      $result \leftarrow true$
3      $S \leftarrow Sons(E_i)$
4      while $result = true$ and $S \neq \emptyset$ do
5          Choose a cluster $E_j \in S$
6          $S \leftarrow S \backslash \{E_j\}$
7          if $Pos(\Sigma)[E_i \cap E_j]$ is a nogood in $N$ then $result \leftarrow false$
8          else if $Pos(\Sigma)[E_i \cap E_j]$ is not a good of $E_i$ w.r.t. $E_j$ in $G$ then
9              $result \leftarrow$ BTD-MAC+NG($P$,$\Sigma$,$E_j$,$E_j \backslash (E_i \cap E_j)$,$G$,$N$)
10             if $result = true$ then
11                 Record $Pos(\Sigma)[E_i \cap E_j]$ as good of $E_i$ w.r.t. $E_j$ in $G$
12             else if $result = false$ then
13                 Record $Pos(\Sigma)[E_i \cap E_j]$ as nogood of $E_i$ w.r.t. $E_j$ in $N$
14     return $result$
15 else
16     Choose a variable $x \in V_{E_i}$
17     Choose a value $v \in d_x$
18     $d_x \leftarrow d_x \backslash \{v\}$
19     if $AC$ ($P$,$\Sigma \cup \langle x = v \rangle$) then
20         $result \leftarrow$ BTD-MAC+NG($P$, $\Sigma \cup \langle x = v \rangle$, $E_i$, $V_{E_i} \backslash \{x\}$, $G$, $N$)
21     else $result \leftarrow false$
22     if $result = false$ then
23         if $must\ restart$ then
24             Record nld-nogoods w.r.t. the decision sequence $(\Sigma \cup \langle x \neq v \rangle)[E_i]$
25             $result \leftarrow unknown$
26         else
27             if $AC$ ($P$,$\Sigma \cup \langle x \neq v \rangle$) then
28                 $result \leftarrow$ BTD-MAC+NG($P$,$\Sigma \cup \langle x \neq v \rangle$,$E_i$,$V_{E_i}$,$G$,$N$)
29     return $result$
```

Algorithm 4 BTD-MAC+RST (**In**: $P = (X, D, C)$: CSP)

```
1  $G \leftarrow \emptyset$; $N \leftarrow \emptyset$
2  repeat
3      Choose a cluster $E_r$ as root cluster
4      result $\leftarrow$ BTD-MAC+NG ($P$,$\emptyset$,$E_r$,$E_r$,$G$,$N$)
5  until $result \neq unknown$
6  return $result$
```

BTD-MAC+NG(P, Σ, E_i, V_{E_i}, G, N) returns $true$ if it succeeds in extending consistently Σ on $Desc(E_i)\backslash(E_i \backslash V_{E_i})$, $false$ if it proves that Σ cannot be consistently extended on $Desc(E_i)\backslash(E_i \backslash V_{E_i})$ or $unknown$ if a restart occurs. BTD-MAC+RST(P) returns $true$ if P has at least a solution, $false$ otherwise.

Theorem 4 *BTD-MAC+RST is sound, complete and terminates.*

Proof BTD-MAC+NG differs from BTD-MAC by exploiting restart techniques, recording reduced nld-nogoods and starting its search with sets G and N which are not necessarily empty. When a restart occurs, the search is stopped and reduced nld-nogoods are safely recorded from Proposition 2. Regarding structural (no)goods, N and G only contain valid structural (no)goods and their uses (lines 7-8) are safe according to Proposition 3. So, as BTD-MAC is sound and terminates and as these properties are not endangered by the differences between BTD-MAC and BTD-MAC+NG, it is the same for BTD-MAC+NG. Then, as BTD-MAC is complete, BTD-MAC+NG is complete under the condition that no restart occurs. Moreover, restarts stop the search without changing the fact that if a solution exists in the part of the search space visited by BTD-MAC+NG, BTD-MAC+NG would find it. As BTD-MAC+RST only performs several calls to BTD-MAC+NG, it is sound. For the completeness, if the call to BTD-MAC+NG is not stopped by a restart (what is necessarily the case of the last call to BTD-MAC+NG if BTD-MAC+RST terminates), the completeness of BTD-MAC+NG implies one of BTD-MAC+RST. Furthermore, recording reduced nld-nogoods at each restart prevents from exploring a part of the search space already explored by a previous call to BTD-MAC+NG. It ensues that, over successive calls to BTD-MAC+NG, one has to explore a more and more reduced part of the search space. Hence, the termination and completeness of BTD-MAC+RST are ensured by the unlimited nogood recording achieved by the different calls to BTD-MAC+NG and by the termination and completeness of BTD-MAC+NG. □

While the use of tree-decompositions of different classes does not change the expression of the complexity of a method such as BTD, we see now that the use of restarts has a significant impact on the complexity analysis. In particular, we must take into account both the number R of restarts and the number N of recorded reduced nld-nogoods.

Theorem 5 *BTD-MAC+RST has a time complexity in $O(n.(w^+)^2.d) + R.((w^+.N + n.s^2.m.\log(d)).d^{w^++2})$ and a space complexity in $O(n.s.d^s + w^+.(d + N))$ with w^+ the width of the considered tree-decomposition, s the size of the largest intersection $E_i \cap E_j$.*

Proof BTD-MAC without nld-nogoods has a time complexity in $O(n.s^2.m.\log(d).d^{w^++2})$. According to Propositions 4 and 5 of [34], storing and managing nld-nogoods of size at most n can be achieved respectively in $O(n^2.d)$ and $O(n.N)$. As, according to Corollary 1, the size of nld-nogoods is at most $w^+ + 1$, this two operations can be achieved respectively in $O((w^+)^2.d)$ and $O(w^+.N)$. BTD-MAC+RST makes at most R calls to BTD-MAC. So we obtain a time complexity for BTD-MAC+RST in $O(R.((n.s^2.m.\log(d) + w^+.N).d^{w^++2} + n.(w^+)^2.d))$.

By exploiting the data structure proposed in [34], the worst case space complexity for storing reduced nld-nogoods is $O(w^+.(d + N))$ since according to Corollary 1, BTD-MAC+RST records N nogoods of size at most $w^+ + 1$. Regarding the storage of structural (no)goods, BTD-MAC+RST has the same space complexity as BTD, namely $O(n.s.d^s)$. So, its whole space complexity is $O(n.s.d^s + w^+.(d + N))$. □

If BTD-MAC+RST exploits a geometric restart policy [46] based on the number of allowed backtracks (i.e. a restart occurs as soon as the number of performed backtracks exceeds the number of allowed backtracks which is initially set to n_0 and increased by a factor r at each restart), we can bound the number of restarts:

Proposition 4 *Given a geometric policy based on the number of backtracks with an initial number n_0 of allowed backtracks and a ratio r, the number of restarts R is bounded by $\left\lceil \frac{\log(n)+(w^++1).\log(d)-\log(n_0)}{\log(r)} \right\rceil$.*

Proof In the worst case, the number of backtracks is bounded by $n.d^{w^++1}$ since we have at most n clusters and the number of backtracks for a cluster is at most $O(d^{w^++1})$. At

ith restart, the number of allowed backtracks is $n_0.r^i$. In the worst case, BTD-MAC+RST terminates as soon as $n_0.r^i \geq n.d^{w^+ +1}$, i.e. as soon as $i \geq \frac{\log(n)+(w^+ +1).\log(d)-\log(n_0)}{\log(r)}$. □

Of course, BTD-MAC+RST can exploit other restart policies like, for example, Luby policy [35].

6 Experiments

In this section, we perform extensive experiments in order to assess the practical interest of bag-connected tree-decompositions and restarts w.r.t. the solving efficiency of BTD. First, we describe our experimental protocol in Section 6.1. Then, in Section 6.2, we assess the benefits from bag-connected tree-decompositions w.r.t. *Min-Fill*. Section 6.3 deals with the practical interest of restarts in BTD. Finally, in Section 6.4, we consider the combination of bag-connected tree-decompositions and restarts.

6.1 Experimental protocol

Nowadays, most of the solvers relying on enumerative methods maintain some level of consistency (which often correspond to arc-consistency) and/or exploit restart techniques. Hence, the algorithm MAC (for Maintaining Arc-Consistency [42]) and its version with restarts and nld-nogood recording, namely MAC+RST+NG [34], can be considered as reference enumerative methods. In our experiments, MAC+RST+NG relies on a geometric restart policy with a ratio 1.1 and an initial number of backtracks of 100. Note that these values lead to the best results for MAC+RST+NG for the set of benchmarks instances we consider.

Regarding BTD, the solving in each cluster is based on MAC. BTD-MAC and BTD-MAC+RST exploit tree-decompositions produced by *Min-Fill* or bag-connected tree-decompositions computed thanks to the *Bag-Connected-TD* algorithm. As stated in Section 4, the behavior of *Bag-Connected-TD* relies on two choices, namely the choice of the first cluster (line 1) and the choice of the next vertex to add to the cluster E_i (line 9). Both can be achieved thanks to heuristics. Note that the heuristics described below allow us to preserve the time complexity of *Bag-Connected-TD*, namely $O(n(n+m))$. In our experiments, the choice of the first cluster in *Bag-Connected-TD* consists in computing greedily a maximal clique of the constraint network. More precisely, the heuristic begins by choosing the vertex having the highest degree in the constraint network. Given a set of already chosen neighbors N, it selects the vertex having the highest degree in the neighborhood of N. Note that we have tried several greedy methods for computing the first maximal clique, but the observed trends are similar to ones obtained with the presented method. Regarding the choice of the next vertex, we have considered several heuristics. These heuristics have the same goal, namely to build connected clusters as soon as possible (i.e. with the smallest size of clusters), but each one tries to reach this goal in a different way. We only present here the best four 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.

Once the tree-decomposition computed, we have to select a root cluster. We have tried several heuristics for this choice. Like for the variable ordering heuristics, these heuristics aim to follow the *first-fail* principle. We present here the best ones:

- RW: we choose the cluster maximizing the sum of weights of constraints whose scope intersects the cluster (the weights are those used by the variable ordering heuristic dom/wdeg [5]).
- RA: we choose alternatively:
  - (i) the cluster containing the next variable according to dom/wdeg applied on all the variables and maximizing sum of weights of constraints whose scope intersects the cluster, or
  - (ii) a cluster according to the decreasing ratio number of constraints over size of the cluster minus one.

In the following, the presented results for BTD-MAC are obtained by exploiting RW for the choice of the root cluster. It turns out that this choice leads to obtain better results than ones obtained with the heuristics presented in [25] or [30]. BTD-MAC+RST with RW uses either a geometric policy with a ratio 1.1 (we denote RWG this combination) or a Luby policy (we denoted RWL this combination). In both cases, the number of allowed backtracks is initially set to 50. For RA, we apply a geometric policy with a ratio 1.1 and initially 75 allowed backtracks when the root cluster is chosen according to the rule (i). For the rule (ii), we use a constant number of allowed backtracks set to 75.

Note that, for efficiency reasons, in BTD-MAC and BTD-MAC+RST, we record the structural nogoods of each cluster in the corresponding global constraint used for recording its nld-nogoods. All the solving methods exploit the algorithm AC-2001 [4] for enforcing the arc-consistency and the variable ordering heuristic dom/wdeg [5].

All the algorithms we exploit here 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 2.4 GHz and with 32 GB of memory.[7] Our set of benchmarks instances consists of 1,859 instances from the CSP 2008 competition. When selecting the instances, we have discarded notably the instances which are detected as inconsistent by the AC preprocessing, the instances which have a trivial tree-decomposition with a single cluster (e.g. the binary instances having a complete constraint graph) and the instances having global constraints (because the corresponding global constraints are not taken into account yet by our CSP library). Note that, when an instance has non-binary constraints, we exploit the 2-section (or primal graph) of its constraint hypergraph to compute (bag-connected) tree-decompositions. Every solving is performed within a time-out of 900 seconds (except for Table 4). The solving runtime includes the decomposition runtime, if any.

6.2 Exploitation of bag-connected tree-decompositions

In this subsection, we mainly compare the solving efficiency and the structural parameters for BTD-MAC using tree-decompositions produced by *Min-Fill* with ones for BTD-MAC exploiting bag-connected tree-decompositions computed thanks to the *Bag-Connected-TD*

[7]Note that this hardware configuration is very different from one used in [29, 30].

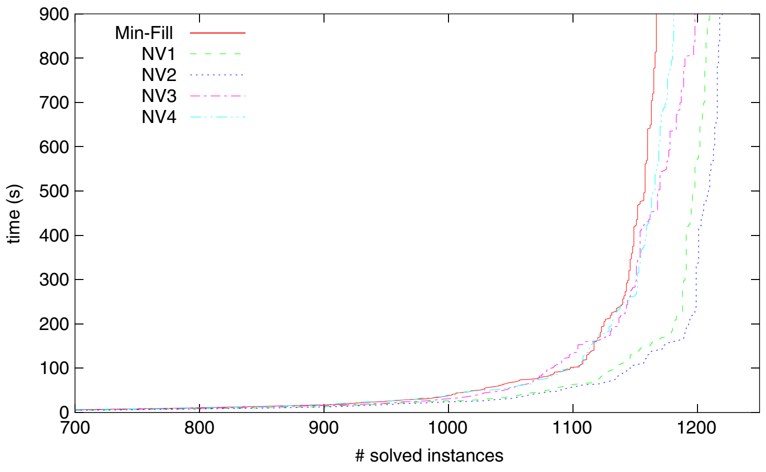

Fig. 7 The cumulative number of instances solved by BTD-MAC for each considered tree-decomposition for instances for which *Min-Fill* produces some disconnected clusters

algorithm. In order to assess only the benefits from the bag-connected tree-decompositions, we do not consider restarts here. Restarts will be used jointly with bag-connected tree-decompositions in Section 6.4.

6.2.1 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. *Min-Fill* produces a tree-decomposition with at least one disconnected cluster for 1,668 instances among the 1,859 instances we consider (i.e. about 90 % of the considered instances). Among these instances, we can notably find instances from families `rlfap`, `fapp`, `modifiedRenault`, `graphColoring`, `bqwh` or `travellingSalesman`. Figure 7 presents the cumulative number of instances solved by BTD-MAC for each considered tree-decomposition for these 1,668 instances.

First, we can observe that, by using any bag-connected tree-decomposition, BTD-MAC solves more instances than by using the disconnected tree-decompositions produced by *Min-Fill*. The best number of solved instances is reached thanks to the tree-decomposition based on the heuristic *NV1* and *NV2*. These decomposition allow us to solve respectively 1,210 and 1,220 instances against 1,167 instances for BTD-MAC with *Min-Fill*. Figure 8 clearly shows that BTD-MAC with *NV1* outperforms BTD-MAC with *Min-Fill* for most instances. Then we can remark that BTD-MAC based on *NV3* and *NV4* is also better than BTD-MAC with *Min-Fill* with respectively 1,198 and 1,180. Moreover, for any decomposition, most instances are solved in less than 60 s.

These results can allow us to provide new explanations about the observations made in [24]. In [24], it was observed that limiting the size of separators often leads to achieve a more efficient solving. In order to do so, when a cluster has a separator with its parent cluster whose size exceeds the limit, this cluster is merged with its parent cluster. Such a limitation allows to reduce the space requirements while offering more freedom to the variable ordering heuristic. In [24, 25], the resulting improvement of the solving efficiency is mainly attributed to these two parameters. However, the existence of disconnected clusters

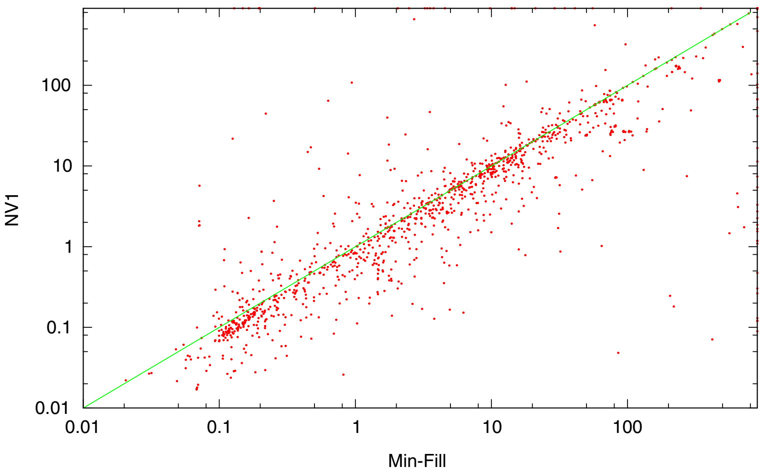

Fig. 8 Runtime of BTD-MAC with *Min-Fill* vs the runtime of BTD-MAC with *NV1* for instances for which *Min-Fill* produces some disconnected clusters

offers a new and complementary insight. Indeed, by limiting the size of separators, for each instance, the number of disconnected clusters decreases necessarily and so it is the same for the probability that the phenomenon described in Section 3.1 occurs. In practice, in the best case, depending on the chosen limit, *Min-Fill* may produce a bag-connected tree-decomposition after merging while, initially, it produces a tree-decomposition with some disconnected clusters. For example, if we set the limit to 15 (which offers a good time-space tradeoff), this is the case for 243 instances among the 1,668 instances we consider here.

Figure 9 presents the cumulative number of instances solved by BTD-MAC for each considered tree-decomposition for the 1,425 instances for which *Min-Fill* produces a tree-decomposition with some disconnected clusters when the size of the separators is limited to

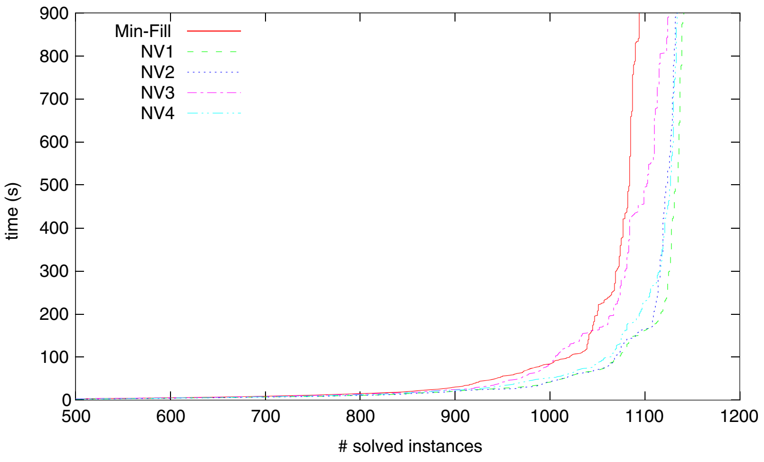

Fig. 9 The cumulative number of instances solved by BTD-MAC for each considered tree-decomposition for instances for which *Min-Fill* produces some disconnected clusters when the size of separators are limited to 15

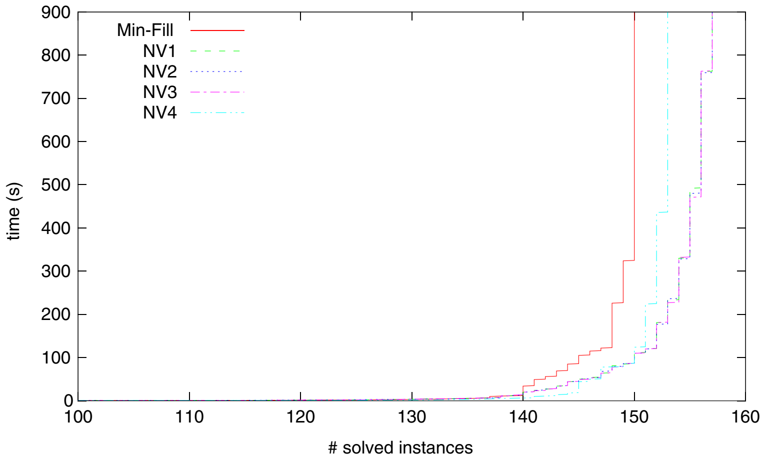

Fig. 10 The cumulative number of solved instances for each considered tree-decomposition for the 191 instances for which *Min-Fill* produces a bag-connected tree-decomposition

15. Again, BTD-MAC with any bag-connected tree-decomposition solves more instances than by using the disconnected tree-decompositions produced by *Min-Fill*. Indeed, BTD-MAC with *Min-Fill* solves 1,094 instances against 1,140, 1,133, 1,124 or 1,134 for *NV1*, *NV2*, *NV3* or *NV4* respectively.

Now, in order to fairly compare the runtimes, we only consider the instances which are solved by BTD-MAC for all the considered tree-decompositions, including *Min-Fill*. The runtime for solving these 1,057 instances by using the decompositions based on *Min-Fill* is 26,876 s while by using the connected decompositions based on *NV1* or *NV2*, it requires only 20,265 s or 19,776 s (i.e. BTD-MAC with *NV1* or *NV2* is at least 25 % faster). BTD-MAC

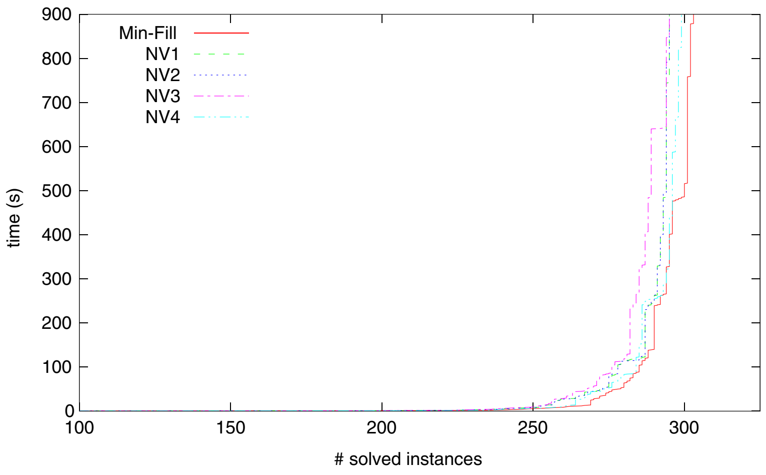

Fig. 11 The cumulative number of solved instances for each considered tree-decomposition for the 434 instances for which *Min-Fill* produces a bag-connected tree-decomposition when the size of separators are limited to 15

based on *NV4* is a little slower (with 23,108 s) but faster than BTD-MAC with *Min-Fill*. BTD-MAC based on *NV3* is the slowest (with 38,105 s).

Finally, if we focus on the 764 instances having a suitable structure (i.e. instances having a ratio n/w^+ greater than 2), again, we observe similar trends, namely that BTD-MAC with bag-connected tree-decomposition performs better than BTD-MAC with *Min-Fill*. For instance, BTD-MAC solves between 651 (for *NV3*) and 666 instances (for *NV1*) against 648 for *Min-Fill*. If we only consider the 616 instances solved by BTD-MAC for all the considered tree-decompositions, the cumulative runtime of BTD-MAC using *NV1*, *NV2* or *NV4* are relatively close to each other (respectively 12,659 s, 12,502 s and 12,966 s) while BTD-MAC using *Min-Fill* (resp. *NV3*) requires in 13,641 s (resp. 18,988 s).

In conclusion, these experimentations have clearly shown that the efficiency of BTD-MAC can be improved by the exploitation of bag-connected tree-decomposition.

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

This subsection 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.

As shown in Fig. 10, BTD-MAC using bag-connected tree-decompositions solves a few additional instances w.r.t. BTD-MAC with *Min-Fill*. Indeed, BTD-MAC with *NV1*, *NV2* or *NV3* solves 157 instances while BTD-MAC with *NV4* or *Min-Fill* solves respectively 153 and 150. However, if we limit the size of the separators to 15 (see Fig. 11), BTD-MAC using *Min-Fill* succeeds in solving more instances (namely 303 instances) than BTD-MAC using *NV1*, *NV2*, *NV3* (295 instances) and *NV4* (299 instances). If we focus our study on the 290 instances which are solved by BTD-MAC for all the considered tree-decompositions, including *Min-Fill*, BTD-MAC using *NV1* or *NV2* obtains the best cumulative runtime (respectively in 4,704 s and 4,759 s) while BTD-MAC using *NV4* or *Min-Fill* are slower (respectively 5,979 s and 6,217 s).

6.2.3 *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.

6.3 Exploitation of restarts

In this subsection, we are interested in the practical interest of the exploitation of restarts for BTD. In order to only assess the interest of restarts, we only consider here *Min-Fill* in

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	m	*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

order to compute the tree-decompositions used by BTD. Furthermore, we limit the size of the separators to 15.

Figure 12 presents the cumulative number of solved instances for each considered solving methods. First, we can note that BTD-MAC+RST with the combinations RWL, RWG or RA clearly outperforms BTD-MAC with respectively 1,467, 1,478 and 1,481 solved instances against 1,397. BTD-MAC+RST with RWG and RA globally lead to a similar behavior. Then we can observe MAC+RST+NG solves more instances (with 1,491 instances) than BTD-MAC+RST. However, in practice, some instances solved by

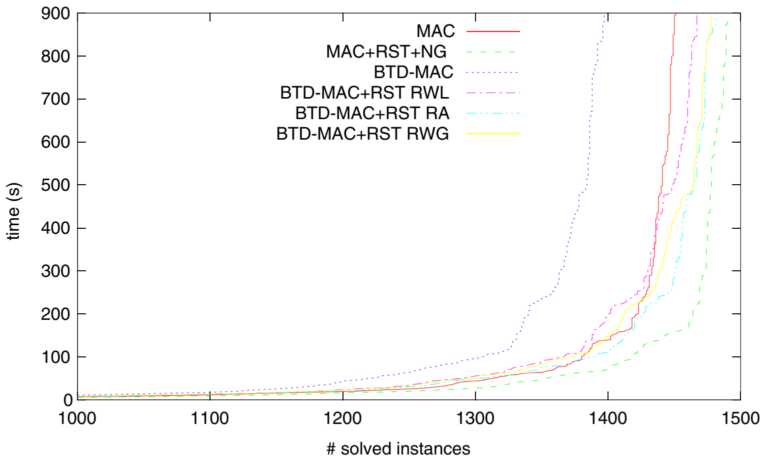

Fig. 12 The cumulative number of solved instances for different solving methods. The tree-decompositions are computed by *Min-Fill*. The size of separators are limited to 15

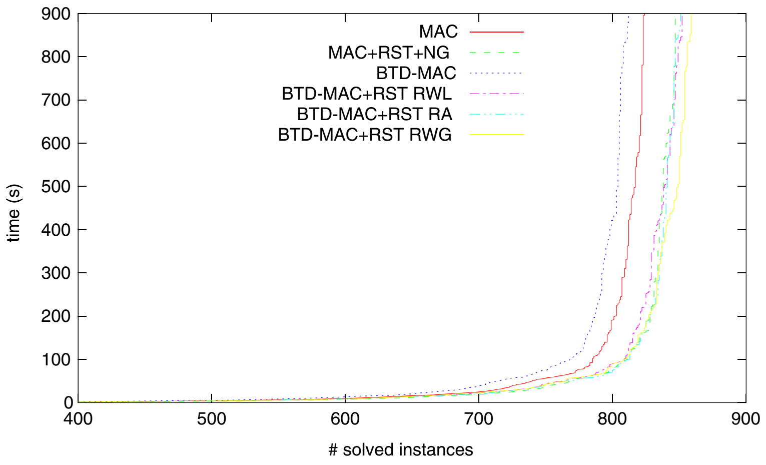

Fig. 13 The cumulative number of solved instances for different methods. The tree-decompositions are computed by *Min-Fill*. The size of separators are limited to 15. Each considered instances has a ratio $\frac{n}{w^+}$ at least 2

BTD-MAC+RST are not solved by MAC+RST+NG and conversely. We also observe that MAC+RST+NG performs sometimes better, sometimes worse than BTD-MAC+RST. This is explained by the fact that most of the 1,859 instances we consider are far from having a suitable structure. In contrast, when the structure has interesting features, BTD-MAC+RST outperforms MAC+RST. Moreover, the more suitable the structure is, the more BTD-MAC+RST outperforms MAC+RST+NG. This phenomenon is illustrated by Figs. 13 and 14 which consider instances having a more and more suitable structure (namely a ratio $\frac{n}{w^+}$ at least 2 and 5 respectively). Finally, we can note that MAC performs worse than BTD-MAC+RST.

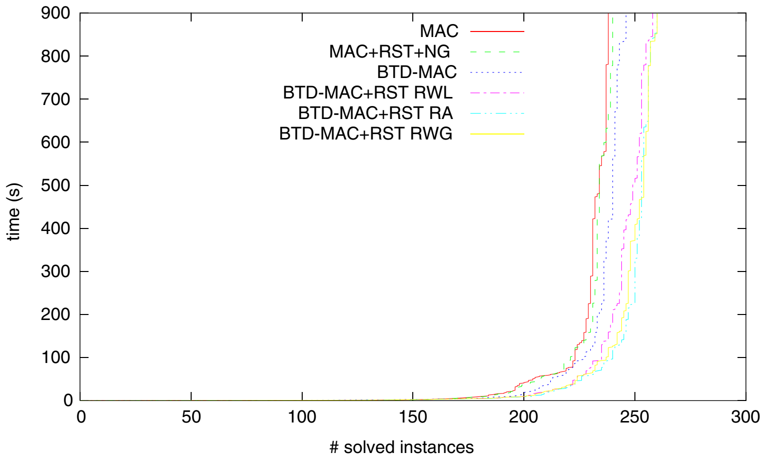

Fig. 14 The cumulative number of solved instances for different methods. The tree-decompositions are computed by *Min-Fill*. The size of separators are limited to 15. Each considered instances has a ratio $\frac{n}{w^+}$ at least 5

Table 2 The number of solved instances and the cumulative runtime in s for each considered algorithm

Family	#inst.	MAC		BTD-MAC		MAC+RST+NG		BTD-MAC+RST with					
								RA		RWL		RWG	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
`dubois`	13	4	1,067.79	13	1.60	4	1,085.41	13	1.63	13	1.61	13	1.61
`geom`	85	85	406.09	85	700.14	85	471.67	85	414.49	85	422.15	85	462.74
`graphColoring`	196	111	3,778.15	113	3,997.65	112	2,974.61	117	3,687.23	118	4,899.30	119	4,238.52
`haystack`	9	2	4.71	8	133.39	2	3.51	8	140.11	8	243.12	8	136.67
`jobshop`	46	37	390.05	35	282.14	46	11.41	46	10.55	46	14.83	46	12.31
`renault`	50	50	19.15	50	66.14	50	19.60	50	27.76	50	24.59	50	24.66
`pret`	8	4	253.57	8	1.06	4	489.56	8	1.07	8	1.07	8	1.07
`rlfapScens11`	12	8	711.28	3	7.55	10	814.15	10	565.93	10	596.84	10	557.48
`Super-jobshop`	46	19	1,016.32	22	1,587.10	33	1,475.36	28	115.80	32	401.94	34	1,051.02
`travellingSalesman-20`	15	15	140.95	15	165.60	15	163.20	15	179.50	15	178.48	15	237.21
Total	480	335	7,788.07	352	6,942.37	361	7,508.48	380	5,144.08	385	6,783.93	388	6,723.29

The tree-decompositions are computed thanks to *Min-Fill*. The size of separators are limited to 15

Table 3 The cumulative runtime in s for each considered algorithm for instances solved by all the algorithms

Family	#inst.	MAC	BTD-MAC	MAC+RST+NG	BTD-MAC+RST with		
					RA	RWL	RWG
dubois	4	1,067.79	0.30	1,085.41	0.30	0.30	0.30
graphColoring	106	3,089.07	3,047.03	2,006.23	2,364.89	2,159.91	1,894.05
haystack	2	4.71	0.03	3.51	0.03	0.03	0.03
jobshop	33	251.32	281.77	4.74	6.07	6.20	6.25
pret	4	253.57	0.28	489.56	0.28	0.28	0.28
rlfapScens11	3	2.91	7.55	2.43	7.43	7.39	7.39
Super-jobshop	16	762.04	486.78	13.46	11.46	11.49	11.44
Total	168	5,431.42	3,823.74	3,605.33	2,390.46	2,185.59	1,919.74

The tree-decompositions are computed thanks to *Min-Fill*. The size of separators are limited to 15

In order to better analyze the behavior of the different algorithms, we now consider the results obtained for some families of instances. Table 2 provides the number of solved instances and the cumulative runtime for each considered algorithm. First, we can note that, for some kinds of instances, like `graphColoring`, the use of restart techniques does not allow to improve significantly the efficiency of BTD-MAC+RST w.r.t. to MAC+RST+NG or BTD-MAC. On the other hand, for the other considered families, we can observe that BTD-MAC+RST provides interesting results. These good results are sometimes due only to the tree-decomposition (e.g. for the families `dubois` or `haystacks`) since they are close to ones of BTD-MAC. Likewise, in some cases, they mainly result from the use of restart techniques (e.g. for the families `jobshop` or `geom`) and they are then close to ones obtained by MAC+RST+NG. Finally, in other cases, BTD-MAC+RST derives fully benefit of both the tree-decomposition and the restart techniques (e.g. for the families `superjobshop` or `rlfapScens11`). In such a case, it often outperforms the three other algorithms.

Then, if we consider the runtime for instances which are solved by all the algorithms (see Table 3), we can note that BTD-MAC+RST with RA, RWL or RWG clearly outperforms the three other algorithms. For example, BTD-MAC+RST with RWG succeeds in solving the 168 instances in 1,919 s while MAC+RST+NG requires 3,605 s.

Now, let us consider the instances of the `rlfapScens11` family, which contains the more difficult RLFAP instances [6]. We can remark that BTD-MAC solves only the three easiest instances. This is explained by bad choices for the root cluster. It turns out that, for all the instances of this family, most choices for the root cluster lead to spend a lot of time to solve some subproblems. So, restart techniques are here very helpful. Table 4 compares the runtime of MAC+RST+NG and BTD-MAC+RST for these instances without any time-out. Clearly, the ratio of the runtime of MAC+RST+NG over one of BTD-MAC+RST increases with the hardness of the instances. At the end, for the hardest instances, BTD-MAC+RST is 30 % faster than MAC+RST+NG.

More generally, we have observed that BTD-MAC+RST is generally more efficient on inconsistent structured instances than MAC+RST+NG. For example, if we consider the instances with a ratio $\frac{n}{w^+}$ at least 5, BTD-MAC+RST with RW or RA requires respectively

Table 4 Runtime in s (without timeout) for the instances of family `rlfapScens11` for BTD-MAC+RST with RWG

Instance	MAC+RST+NG	BTD-MAC+RST RWG
scen11-f1	4,244.08	3,000.41
scen11-f2	1,697.12	1,202.21
scen11-f3	563.47	371.83
scen11-f4	182.15	124.96
scen11-f5	47.17	31.20
scen11-f6	9.74	8.62
scen11-f7	6.54	6.67
scen11-f8	1.42	3.50
scen11-f9	1.25	3.32
scen11-f10	0.87	2.48
scen11-f11	0.78	2.45
scen11-f12	0.77	2.46

The tree-decompositions are computed thanks to *Min-Fill*. The size of separators are limited to 15

1,419 s and 1,482 s to solve the 87 inconsistent instances which are solved by all the algorithms, while MAC+RST+NG needs 2,912 s. Such a phenomenon is partially explained by the use of the tree-decomposition. Indeed, if BTD-MAC+RST explores an inconsistent cluster at the beginning of the search, it may quickly prove the inconsistency of the problem.

Finally, as explained in Section 3, the negative impact due to the existence of disconnected clusters is partially related to the choice of the root cluster. As BTD-MAC+RST may choose a new root cluster at each restart, it is quite natural to wonder how BTD-MAC+RST behaves when the considered tree-decompositions have some disconnected clusters. To do so, we now take into account the 1,425 instances for which *Min-Fill* produces a tree-decomposition with some disconnected clusters when the size of the separators is limited to 15. We can note that the exploitation of restarts may limit the negative impact due to the existence of disconnected clusters. Indeed, BTD-MAC+RST with *Min-Fill* and RWL, RA or RWG succeeds in solving more instances than BTD-MAC with *Min-Fill* or any *NVi*. For example, BTD-MAC+RST with RWL, RA or RWG solves respectively 1,149, 1,167 and 1,157 against 1,140 instances for BTD-MAC with *NV1* which obtains the best results on these instances in the previous subsection.

In conclusion, clearly the exploitation of restarts allows us to improve the efficiency of BTD, with very significant gains for some kinds of instances. Moreover, it also limits the negative impact due to the existence of disconnected clusters by choosing possibly a new root cluster at each restart.

6.4 Exploitation of both restarts and bag-connected tree-decomposition

In this subsection, we assess the complementarity of restarts and bag-connected tree-decompositions. In practice we have considered a lot of combinations of restart policy and bag-connected tree-decompositions. However, for sake of simplicity, we only present here the results for three of them, knowing that similar trends are observed for the others. Figure 15 provides the cumulative number of solved instances for each considered solving method.

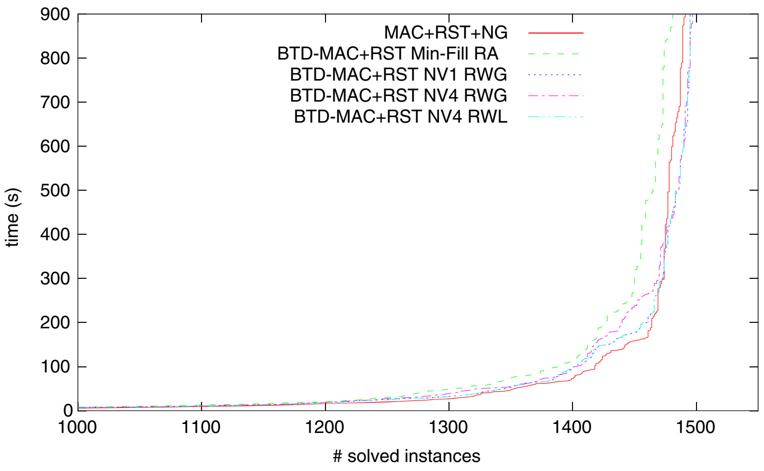

Fig. 15 The cumulative number of solved instances for different methods. The tree-decompositions are computed by *Min-Fill*, *NV1* or *NV4*. The size of separators are limited to 15

First, we note that the three combinations (namely *NV1* and RWG, *NV4* and RWG and *NV4* and RWL) lead to obtain close results. Indeed, BTD-MAC+RST with these combinations solves respectively 1,495, 1,497 and 1,499 instances. Then, these results improve ones obtained for BTD-MAC+RST with Min-Fill and RWG, RWL or RA which allows BTD-MAC+RST to solve respectively 1,478, 1,467 and 1,481 instances. Moreover, BTD-MAC+RST with these combinations also solves more instances than MAC+RST+NG (1,491 instances). Note that we have observed similar improvements if we focus our study on the instances for which *Min-Fill* produces a tree-decomposition with some disconnected clusters or on the instances having a suitable structure. Finally, we compare the runtime of BTD-MAC+RST with *Min-Fill* and RA with one of BTD-MAC+RST with *NV4* and RWL (see Fig. 16). Clearly, for most instances, BTD-MAC+RST with *NV4* and RWL performs faster than BTD-MAC+RST with *Min-Fill* and RA.

6.5 Summary

We have raised two issues (namely the existence of disconnected clusters and the choice of the root cluster) which may influence significantly the efficiency when solving CSPs by a decomposition method like BTD. In this section, we have shown the practical interest of the responses we have proposed to these two issues.

First, regarding the existence of disconnected clusters, our experimentations have clearly shown its negative impact on the solving efficiency and that the efficiency of BTD-MAC can be improved by the exploitation of bag-connected tree-decompositions. Indeed, when *Min-Fill* produces tree-decompositions with some disconnected clusters, BTD-MAC with *Min-Fill* solves less instances than BTD-MAC exploiting bag-connected tree-decomposition. However, when *Min-Fill* produces bag-connected tree-decompositions, BTD-MAC with *Min-Fill* and BTD-MAC with bag-connected tree-decompositions obtain similar results even if they do not use the same tree-decompositions. These results can also allow us to provide new explanations about the observations made in [24].

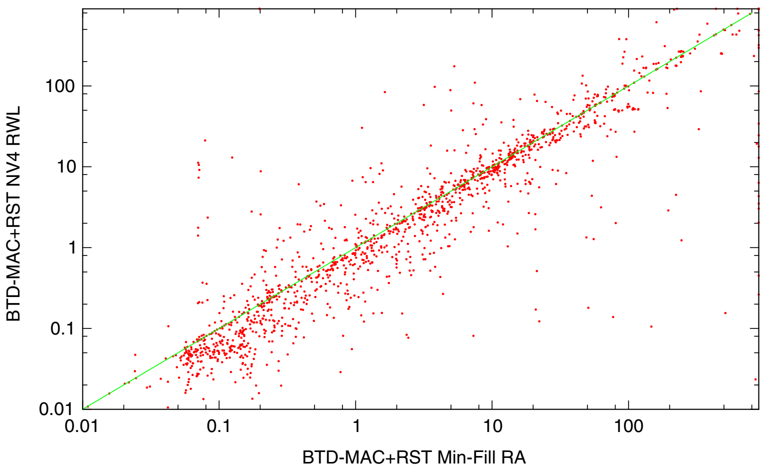

Fig. 16 Runtime of BTD-MAC+RST with *Min-Fill* and RA vs runtime of BTD-MAC+RST with *NV4* and RWL. The size of separators are limited to 15

Then, regarding the choice of the root cluster, the exploitation of restart techniques inside BTD-MAC+RST turns to be an interesting alternative. Notably, it leads to solve more instances than BTD-MAC. Moreover, it may limit the negative impact due to the existence of disconnected clusters.

Finally, the joint use of bag-connected tree-decompositions and restart techniques allows us to improve even more the solving efficiency. For instance, BTD-MAC+RST solves more instances than any considered algorithm including BTD-MAC, MAC and MAC+RST+NG. Furthermore, it often performs faster, especially when the instances have a suitable structure.

7 Conclusion

In this paper, we discussed two important issues that arise when solving CSPs with decomposition methods and which have a crucial role for the efficiency the these approaches. The first one concerns the choice of decompositions (tree of clusters) that will be considered. The second question concerns the choice of the cluster that will be used as root cluster during search.

If the first point has been much studied, most of the works done for a long time concerns the minimization of the width of the considered decompositions. We initially identified a phenomenon that occurs particularly when the used decompositions correspond to good approximations of the optimal tree-decomposition (i.e. the decompositions of minimal width). In such cases, a majority of the decompositions have clusters possessing several connected components. This phenomenon which has apparently never been observed before is both very frequent and, above all, can cause considerable damage to the efficiency of the search. So, in this paper, we have introduced the concept of bag-connected tree-decomposition (all their clusters are connected) in the field of constraint network decomposition. This concept was proposed recently in the area of combinatorics. After we have shown the interest of this new class of decompositions 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 CSPs using decomposition methods. Indeed, by using such decompositions, we show experimentally that decomposition methods like BTD succeed in solving many more instances than by using classical decompositions.

To address the problem of choosing the root cluster, we propose here to apply the principle of restarts in decomposition-based methods. To this end, first, we have described how usual nogoods can be incorporated into a decomposition-based method while preserving the structure induced by a given decomposition. Next we have introduced the concept of *oriented structural good*. Indeed, while the structural nogoods can be used directly, using an approach based on restarts, the goods must verify certain properties on the order of exploration of a tree-decomposition. Our experimentations show clearly the practical advantage of exploiting restarts in the context of decomposition-based methods. Finally, we present experiments demonstrating that the jointly use of bag-connected tree-decompositions and restarts allows to improve significantly the efficiency of decomposition methods.

The extension of this work concerns at least two main directions. The first one is on further study of the concept of restart. In particular, it is well known that this kind of techniques, for example in the case of SAT solvers, was the subject of numerous studies for a long time to provide better policies of restarts, but also for the management of recorded nogoods. Results from these previous works will not be necessarily usable here because it will be probably necessary to take into account the specificity of decompositions. On the one hand, "classic" nogoods are handled, but above all, structural goods and structural nogoods induce probably specific policies. This difference must be taken into account in the development of more effective policies which may, moreover, be considered locally or globally in a given decomposition. Moreover, the study of this approach to a meta-level needs to be addressed. For instance, the issue of the choice of a tree-decomposition, and then, of its modification during search needs to be addressed to enable the management of "dynamic" decompositions.

The second extension of this work is related to the study of bag-connected tree-decompositions and restarts in the more general field of Graphical Models in AI. This concerns the study of these notions for other classes of methods as Hypertree-Decomposition [20], And/Or Search [9, 10], Bucket Elimination [7], 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 [12]). Another promising study is related to the field of optimization and counting problems. We know that approaches as BTD has already been successfully used in these two areas [14, 44]. However, this extension is not trivial. For example, the use of valued structural goods in the case of optimization requests a comprehensive study to enable its implementation while using restarts.

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

1. Arnborg, S., Corneil, D., & Proskuroswki, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics*, *8*, 277–284.
2. Berge, C. (1973). Graphs and Hypergraphs. Elsevier.
3. Bessière, C., Meseguer, P., Freuder, E.C., & Larrosa, J. (2002). On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, *141*, 205–224.
4. Bessière, C., & Régin, J.C. (2001). Refining the basic constraint propagation algorithm. In *Proceedings of IJCAI* (pp. 309–315).
5. Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search by weighting constraints. In *Proceedings of ECAI* (pp. 146–150).
6. Cabon, C., de Givry, S., Lobjois, L., Schiex, T., & Warners, J.P. (1999). Radio link frequency assignment. *Constraints*, *4*, 79–89.
7. Dechter, R. (2003). Constraint processing. Morgan Kaufmann Publishers.
8. Dechter, R., & Fattah, Y.E. (2001). Topological parameters for time-space tradeoff. *Artificial Intelligence*, *125*, 93–118.
9. Dechter, R., & Mateescu, R. (2004). The impact of AND/OR search spaces on constraint satisfaction and counting. In *Proceedings of the 10th international conference on principles and practice of constraint programming (CP)* (pp. 731–736).
10. Dechter, R., & Mateescu, R. (2007). AND/OR search spaces for graphical models. *Artificial Intelligence*, *171*, 73–106.
11. Dechter, R., & Pearl, J. (1989). Tree-clustering for constraint networks. *Artificial Intelligence*, *38*, 353–366.
12. Dermaku, A., Ganzow, T., Gottlob, G., McMahan, B.J., Musliu, N., & Samer, M. (2008). Heuristic methods for hypertree decomposition. In *Proceedings of MICAI* (pp. 1–11).
13. Diestel, R., & Müller, M. (2014). Connected tree-width. arXiv:1211.7353v2.
14. Favier, A., de Givry, S., & Jégou, P. (2009). Exploiting problem structure for solution counting. In *Proceedings of CP* (pp. 335–343).
15. Fraigniaud, P., & Nisse, N. (2006). Connected treewidth and connected graph searching. In *Proceedings of LATIN* (pp. 479–490).
16. Freuder, E. (1982). A sufficient condition for backtrack-free search. *Journal of the ACM*, *29*(1), 24–32.
17. Gavril, F. (1972). 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.
18. Golumbic, M. (1980). *Algorithmic graph theory and perfect graphs*. New York: Academic Press.
19. Gomes, C.P., Selman, B., Crato, N., & Kautz, H.A. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, *24*(1/2), 67–100.
20. Gottlob, G., Leone, N., & Scarcello, F. (2000). A comparison of structural CSP decomposition methods. *Artificial Intelligence*, *124*, 243–282.
21. Gyssens, M., Jeavons, P., & Cohen, D. (1994). Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, *66*, 57–89.
22. Hamann, M., & Weißauer, D. (2015). Bounding connected tree-width. ArXiv:1503.01592.
23. Harvey, W.D. (1995). Nonsystematic backtracking search. Ph.D. thesis, Stanford University.
24. Jégou, P., Ndiaye, S.N., & Terrioux, C. (2005). Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of CP* (pp. 777–781).
25. Jégou, P., Ndiaye, S.N., & Terrioux, C. (2006). An extension of complexity bounds and dynamic heuristics for tree-decompositions of CSP. In *Proceedings of CP* (pp. 741–745).
26. Jégou, P., Ndiaye, S.N., & Terrioux, C. (2007). Dynamic heuristics for backtrack search on tree-decomposition of CSPs. In *Proceedings of IJCAI* (pp. 112–117).
27. Jégou, P., Ndiaye, S.N., & Terrioux, C. (2007). Dynamic management of heuristics for solving structured CSPs. In *Proceedings of CP* (pp. 364–378).
28. Jégou, P., & Terrioux, C. (2003). Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, *146*, 43–75.
29. Jėgou, P., & Terrioux, C. (2014). Combining restarts, nogoods and decompositions for solving csps. In *Proceedings of ECAI* (pp. 465–470).
30. Jėgou, P., & Terrioux, C. (2014). Tree-decompositions with connected clusters for solving constraint networks. In *Proceedings of CP* (pp. 407–423).
31. Karakashian, S., Woodward, R., & Choueiry, B.Y. (2013). Improving the performance of consistency algorithms by localizing and bolstering propagation in a tree decomposition. In *Proceedings of AAAI* (pp. 466–473).

32. Kjaerulff, U. (1990). Triangulation of graphs - algorithms giving small total state space. Tech. rep., Judex R.R. Aalborg, Denmark.
33. Lecoutre, C. (2009). Constraint networks - techniques and algorithms. ISTE/Wiley.
34. Lecoutre, C., Saïs, L., Tabary, S., & Vidal, V. (2007). Recording and minimizing nogoods from restarts. *JSAT*, *1*(3–4), 147–167.
35. Luby, M., Sinclair, A., & Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, *47*(4), 173–180.
36. Müller, M. (2012). Connected tree-width. ArXiv:1211.7353.
37. Nadel, B. (1988). Tree search and arc consistency in constraint-satisfaction algorithms. In *Search in artificial intelligence* (pp. 287–342). Springer-Verlag.
38. Petke, J. (2012). On the bridge between constraint satisfaction and Boolean satisfiability. Ph.D. thesis, University of Oxford.
39. Robertson, N., & Seymour, P. (1986). Graph minors II: algorithmic aspects of treewidth. *Algorithms*, *7*, 309–322.
40. Rose, D.J. (1973). A graph theoretic study of the numerical solution of sparse positive denite systems of linear equations, In Read, R.C. (Ed.) *Graph theory and computing* (pp. 183–217). New York: Academic Press.
41. Rossi, F., van Beek, P., & Walsh, T. (2006). Handbook of constraint programming. Elsevier.
42. Sabin, D., & Freuder, E. (1994). Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of ECAI* (pp. 125–129).
43. Sabin, D., & Freuder, E. (1997). Understanding and improving the MAC algorithm. In *Proceedings of CP* (pp. 167–181).
44. Sanchez, M., Bouveret, S., de Givry, S., Heras, F., Jégou, P., Larrosa, J., Ndiaye, S.N., Rollon, E., Schiex, T., Terrioux, C., Verfaillie, G., & Zytnicki, M. (2008). Max-CSP competition 2008: toulbar2 solver description. In *Proceedings of the 3rd CSP solver competition, CP workshop* (pp. 63–70).
45. Tarjan, R., & 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.
46. Walsh, T. (1999). Search in a small world. In *Proceedings of IJCAI* (pp. 1172–1177).