ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-465

Combining Restarts, Nogoods and Decompositions for Solving CSPs

Philippe Jégou and **Cyril Terrioux** [1]

Abstract. From a theoretical viewpoint, the (tree-)decomposition methods offer a good approach when the (tree)-width of constraint networks (CSPs) is small. In this case, they have often shown their practical interest. However, sometimes, a bad choice for the root cluster (a tree-decomposition is a tree of clusters) may drastically degrade the performance of the solving.

In this paper, 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 [8]) integrating restart techniques. From a theoretical viewpoint, we prove that reduced nld-nogood can be safely recorded during the search and that their size is smaller than ones recorded by MAC+RST+NG [9]. 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 restart techniques for solving CSPs by decomposition methods.

1 INTRODUCTION

Constraint Satisfaction Problems (CSPs, see [14] for a state of the art) provide an efficient way of formulating problems in computer science, especially in Artificial Intelligence.

Formally, a *constraint satisfaction problem* is a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of n variables, $D = (d_{x_1}, \ldots, d_{x_n})$ is a list of finite domains of values, one per variable, and $C = \{C_1, \ldots, C_e\}$ is a finite set of e constraints. Each constraint C_i is a pair $(S(C_i), R(C_i))$, where $S(C_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ is the *scope* of C_i, and $R(C_i) \subseteq d_{x_{i_1}} \times \cdots \times d_{x_{i_k}}$ is its *compatibility relation*. The *arity* of C_i is $|S(C_i)|$. A CSP is called *binary* if all constraints are of arity 2. The structure of a constraint network is represented by a hypergraph (which is a graph in the binary case), called the constraint (hyper)graph, whose vertices correspond to variables and edges to the constraint scopes. In this paper, for sake of simplicity, we only deal with the case of binary CSPs but this work can easily be extended to non-binary CSP by exploiting the 2-section [1] 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 $(X, \{S(C_1), \ldots S(C_e)\})$ by (X, C). An assignment on a subset of X is said to be *consistent* if it does not violate any constraint. Testing whether a CSP has a *solution* (i.e. a consistent assignment on all the variables) is known to be NP-complete. So the time complexity of backtracking algorithms which are usually exploited to solve CSPs, is naturally exponential, at least in $O(e.d^n)$.

[1] Aix-Marseille Université, LSIS UMR 7296, France {philippe.jegou, cyril.terrioux}@lsis.org

Many works have been realized to make the solving more efficient in practice, by using optimized backtracking algorithms, heuristics, constraint learning, non-chronological backtracking, filtering techniques, etc. 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. 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. They have been recently introduced in the CSP framework (e.g. in [9]). For efficiency reasons, they are usually exploited with some learning techniques (like recording of nld-nogoods in [9]).

In this paper, we introduce for the first time the restart techniques in the context of decomposition methods for solving CSPs. Decomposition methods (e.g. [4, 8]) solve CSPs by taking into account some particular features of the constraint networks. Often, they rely on the notion of tree-decomposition of graphs [12]. In such a case, their advantage is related to their theoretical complexity, i.e. $d^{w^+ + 1}$ where w^+ is the width of the considered tree-decomposition. Since computing an optimal tree-decomposition is NP-Hard, the used tree-decompositions are generally computed by heuristic methods and so approximate optimal tree-decompositions. When this graph has nice topological properties and thus when w^+ is small, these methods allow to solve large instances, e.g. radio link frequency assignment problems [3]. From a practical viewpoint, they have obtained promising results on such instances. However, their efficiency may drastically be degraded by some bad choices performed by heuristics. To present this issue, we consider here the BTD method [8] which is a reference in the state of the art for this type of approach [11].

For BTD, the considered tree-decomposition and the choice of the root cluster (i.e. the first studied cluster) induce a particular variable ordering. Hence, as it is well known that the variable ordering has a significant impact on the efficiency of the solving, the choice of the root cluster is crucial. In [7], 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. In the next section, we explain why it is difficult to propose a suitable choice for the root cluster. As a consequence, in order to reduce the impact of the root cluster on the practical efficiency, we propose an alternative based on restart techniques. Then, we present a new version of BTD integrating restart techniques. From a theoretical viewpoint, we prove that reduced nld-nogood can be safely recorded during the search and that their size is smaller than ones recorded by MAC+RST+NG [9]. We also show how structural (no)goods can be exploited when the search restarts from a new root cluster. Finally, from a practical viewpoint, we show experimentally the benefits of the use of restart techniques for solving CSPs by decomposition methods.

Section 2 recalls the frame of BTD and describes the BTD-MAC algorithm[2]. Then, section 3 presents the algorithm BTD-MAC+RST. In section 4, we assess the benefits of restarts when solving CSPs thanks to a decomposition-based method and conclude in section 5.

2 THE BTD METHOD

BTD [8] relies on the notion of tree-decomposition of graphs [12].

Definition 1 *A* tree-decomposition *of a graph $G = (X, C)$ is a pair (E, T) with $T = (I, F)$ a tree and $E = \{E_i : i \in I\}$ a family of subsets of X, such that each subset (called cluster) E_i is a node of T and satisfies: (i) $\cup_{i \in I} E_i = X$, (ii) for each edge $\{x, y\} \in C$, there exists $i \in I$ with $\{x, y\} \subseteq E_i$, and (iii) for all $i, j, k \in I$, if k is in a path from i to j in T, then $E_i \cap E_j \subseteq E_k$. The width of a tree-decomposition (E, T) is equal to $max_{i \in I}|E_i| - 1$. The* tree-width w *of G is the minimal width over all the tree-decompositions of G.*

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. Figure 1(b) presents a tree whose nodes correspond to the maximal cliques of the graph depicted in Figure 1(a). It is a possible tree-decomposition for this graph. So, we get $E_1 = \{x_1, x_2, x_3\}$, $E_2 = \{x_2, x_3, x_4, x_5\}$, $E_3 = \{x_4, x_5, x_6\}$, and $E_4 = \{x_3, x_7, x_8\}$. As the maximum size of clusters is 4, the tree-width of this graph is 3. We have $Desc(E_1) = X$ and $Desc(E_2) = \{x_2, x_3, x_4, x_5, x_6\}$.

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 Figure 1, $E_1 < E_2 < E_3 < E_4$ (resp. $x_1 \preceq x_2 \preceq x_3 \preceq \ldots \preceq x_8$) is a possible compatible ordering on E (resp. 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, it only assigns the variables which appears in the cluster E_i but not in its parent cluster $E_{p(i)}$, that is all the variables of the cluster E_i except the variables of the separator $E_i \cap E_{p(i)}$[3].

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 [15]). During the solving, 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) and *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 $x_\ell \neq v_\ell$. Regarding BTD-MAC (i.e. BTD relying on MAC for solving each cluster), we can note that 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 $Desc(E_j)$ [8]. 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 8 variables $x_1, \ldots, x_8$ for which each domain is $\{a, b, c\}$ and whose constraint graph and a possible tree-decomposition are given in Figure 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 $E_1 < E_2 < E_3 < E_4$. 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 studies the consistent decision sequence $\langle x_1 \neq a, x_3 = b, x_1 = b, x_2 \neq a, x_2 = c \rangle$, it will keep on its search with the next child cluster of E_1, namely E_4, if $\{x_2 = c, x_3 = b\}$ has been recorded as a good, or backtrack to the last decision in E_1 if $\{x_2 = c, x_3 = b\}$ corresponds to as a nogood.

Algorithm 1 without the lines 21-24 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 16-27 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 $O(n.s^2.e.\log(d).d^{w^+ + 2})$ 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.

From a practical viewpoint, generally, BTD efficiently solves CSPs having a small tree-width [6, 7, 8]. 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[4] and, for each instance, we run BTD from each cluster of its considered tree-decomposition.

[2] BTD-MAC has never been described before in the literature. The algorithm MAC-BTD evoked in [8] is in fact RFL-BTD, i.e. BTD based on Real Full Look-ahead [10] (see [16] for a comparison between MAC and RFL).

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

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

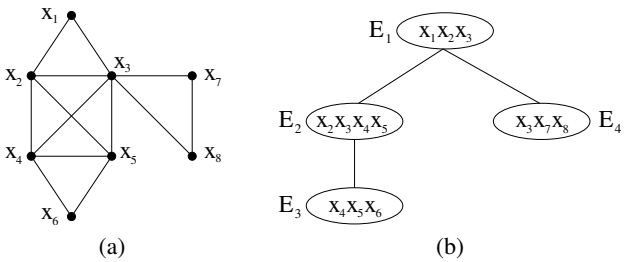

Figure 1. A constraint graph for 8 variables (a) and an optimal tree-decomposition (b).

We have first observed that for a same instance, the runtimes may differ from several orders of magnitude according 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 [7], 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 3, we propose a possible one consisting in exploiting restart techniques.

3 EXPLOITING RESTARTS WITHIN BTD

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. Regarding the learning, BTD already exploits structural (no)goods. However, depending on when the restart occurs, we have no warranty that a (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 (for negative last decision nogoods) whose practical interest has been highlighted in the MAC+RST+NG algorithm [9]. We first recall the notion of nogood in the case of MAC:

Definition 2 ([9]) *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, we assume that for any variable x_i and value v_i, the positive decision $x_i = v_i$ is considered before the decision $x_i \neq v_i$.

Proposition 1 ([9]) *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 can record several new reduced nld-nogoods and exploit them later to prevent from exploring again an already visited part of the search tree. These nld-nogoods can be efficiently computed and stored as a global constraint with an efficient specific propagator for enforcing AC [9].

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 [9] 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 $\Sigma = \langle \delta_1, \ldots, \delta_k \rangle$ 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 $\Sigma'_{E_i} = \langle \delta_{i_1}, \ldots, \delta_{i_\ell} \rangle$ 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. $\Delta' \subseteq \Delta$.*

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 regardless 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. 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 2 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 1 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 16-27) 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 21). If so, it records reduced nld-nogoods w.r.t. the decision sequence Σ restricted to the decisions involving variables of E_i (line 22) 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 25) like in [9]. 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.

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 1 *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. □

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

Algorithm 2: BTD-MAC+RST (**In**: $P = (X, D, C)$: CSP)

```
1 G ← ∅; N ← ∅
2 repeat
3     Choose a cluster E_r as root cluster
4     result ← BTD-MAC+NG (P,∅,E_r,E_r,G,N)
5 until result ≠ unknown
6 return result
```

Theorem 2 *BTD-MAC+RST has a time complexity in* $O(R.((n.s^2.e.\log(d) + w^+.N).d^{w^+ +2} + n.(w^+)^2.d))$ *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* $E_i \cap E_j$, R *the number of restarts and* N *the number of recorded reduced nld-nogoods.*

Proof: BTD-MAC without nld-nogoods has a time complexity in $O(n.s^2.e.\log(d).d^{w^+ +2})$. According to Propositions 4 and 5 of [9], 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 $O(R.((n.s^2.e.\log(d) + w^+.N).d^{w^+ +2} + n.(w^+)^2.d))$.

By exploiting the data structure proposed in [9], 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 [17] 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* $\left\lceil \frac{\log(n)+(w^+ +1).\log(d)-\log(n_0)}{\log(r)} \right\rceil$.

4 EXPERIMENTATIONS

In this section, we assess the benefits of restarts when solving CSPs thanks to a decomposition-based method. With this aim in view, we compare BTD-MAC+RST with BTD-MAC and MAC+RST+NG on 647 instances (of arbitrary arity) among the instances of the CSP 2008 Competition. The selected instances are ones which have suitable tree-decompositions (i.e. a ratio n/w^+ at least equal to 2). These tree-decompositions are computed thanks to Min-Fill [13] which is considered as the best heuristic of the state of the art [5]. The runtime of BTD-MAC(+RST) includes the time required to compute the tree-decomposition. All the methods exploit the *dom/wdeg* variable heuristic [2]. We have tried several heuristics for the choice of the root cluster. 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 of dom/wdeg). This heuristic is also one exploited by BTD-MAC.
- RA: we choose alternatively 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 a cluster according to the decreasing ratio number of constraints over size of the cluster minus one.

Both heuristics RW and RA aim to follow the first-fail principle. The second case of RA brings some diversity in the search. The used restart policies rely on the number of allowed backtracks. The presented values below are ones providing the best results among the tested values. More precisely, for MAC+RST+NG, we exploit a geometric policy where the initial number of allowed backtracks is 100 while the increasing factor is 1.1. BTD-MAC+RST with RW uses a geometric policy with a ratio 1.1 and initially 50 allowed backtracks. For RA, we apply a geometric policy with a ratio 1.1 and initially 75 allowed backtracks when the cluster is chosen according to the first case. In the second case, we use a constant number of allowed backtracks set to 75. All the implementations are written in C++. The experimentations are performed on a linux-based PC with an Intel Pentium IV 3.2 GHz and 1 GB of memory. The runtime limit is set to 1,200 s (except for Table 1).

Figure 2 presents the cumulative number of solved instances for each considered algorithm. First, we can note that the two heuristics RW and RA globally lead to a similar behavior for BTD-MAC+RST. Then it appears clearly that BTD-MAC+RST solves more instances

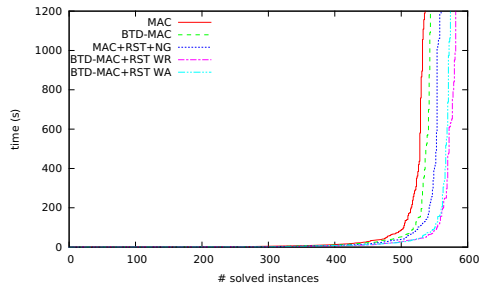

Figure 2. The cumulative number of solved instances per algorithm.

Table 1. Runtime in s (without timeout) for the scen11 instances.

Instance	MAC+RST+NG	BTD-MAC+RST
scen11-f12	0.51	0.30
scen11-f11	0.50	0.30
scen11-f10	0.65	0.35
scen11-f9	1.32	1.54
scen11-f8	1.60	1.78
scen11-f7	12.93	6.81
scen11-f6	20.23	9.86
scen11-f5	102	45.72
scen11-f4	397	202
scen11-f3	1,277	609
scen11-f2	3,813	1,911
scen11-f1	9,937	5,014

than any other algorithm. For instance, BTD-MAC+RST solves 582 instances in 15,863 s with RW (resp. 574 instances in 13,280 s for RA) while MAC+RST+NG only solves 560 instances in 16,943 s. Without restart techniques, the number of solved instances is still smaller with 536 and 544 instances in 18,063 s and 13,256 s for MAC and BTD-MAC respectively.

In order to better analyze the behavior of the different algorithms, we now consider the results obtained per family of instances[5]. Table 2 provides the number of solved instances and the cumulative runtime for each considered algorithm while Table 3 gives the runtime for instances which are solved by all the algorithms. First, we can note that, for some kinds of instances, like graph coloring, the use of restart techniques does not allow to improve 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 renault, superjobshop or scen11). In such a case, it clearly outperforms the three other algorithms. For example, it is twice faster than MAC+RST+NG for solving the instances of the scen11 family, which contains the more difficult RLFAP instances [3]. Table 1 presents the runtime of MAC+RST+NG and BTD-MAC+RST for these instances. We can remark that BTD-MAC solves only the three easiest instances. This is explained by bad choices for the root cluster. It turns that, for all the instances of this

[5] Note that we do not take into account all the instances of a given family, but only ones having a suitable tree-decomposition.

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 RW		BTD-MAC+RST RA	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
dubois	13	5	2,232	13	0.03	5	2,275	13	0.04	13	0.05
geom	83	83	415	83	819	83	479	83	468	83	460
graphColoring	39	29	1,989	33	1,291	29	2,783	34	2,825	33	2,769
haystacks	46	2	5.82	8	169	2	4.43	8	172	8	172
jobshop	46	37	617	35	469	46	14.87	46	13.15	46	10.93
renault	50	50	23.89	50	86.81	50	24.30	50	22.96	50	24.73
pret	8	4	250	8	0.05	4	552	8	0.06	8	0.05
scens11	12	8	1,632	3	1.25	9	537	10	878	10	882
Super-jobShop	46	19	1,648	21	1,179	33	2,315	34	1,553	27	449
travellingSalesman-20	15	15	191	15	229	15	214	15	346	15	294

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 RW	BTD-MAC+RST RA
dubois	5 / 13	2,232	0.01	2,275	0.01	0.01
graphColoring	27 / 39	951	1,051	1,308	846	1,277
haystacks	2 / 46	5.82	0	4.43	0	0.01
jobshop	33 / 46	392	468	5.63	5.10	4.48
pret	4 / 8	250	0.01	552	0.02	0
rlfapScens11	3 / 12	2.75	1.25	1.66	0.95	1.10
Super-jobShop	16 / 46	1,275	830	14.83	9.60	16.04

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.

Finally, we have observed that BTD-MAC+RST is generally more efficient on inconsistent instances than MAC+RST+NG. For example, it requires 4,260 s to solve the inconsistent instances which are solved by all the algorithms while MAC+RST+NG needs 7,105 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.

5 CONCLUSION

In this paper, we have firstly presented the integration of MAC in BTD. We have then shown how it is possible to enhance the decomposition-based methods with the integration of the principle of restarts. This has led us to significantly extend the BTD method. We have first described how classic nogoods can be incorporated into a decomposition-based method while preserving the structure induced by a considered decomposition. Next we have introduced the concept of *oriented structural good*. Indeed, if the structural nogoods can be used directly by BTD using restarts, the goods must verify certain properties on the order of exploration of a tree-decomposition, and then, the notion of oriented structural good becomes necessary. In the last part of this paper, the experimentations show clearly the practical interest of exploiting restarts in decomposition-based methods. It effectively overcomes the problem induced by the order of exploration of clusters which harms very often and significantly to their practical effectiveness. These results also show that adding restarts to BTD can significantly outperform the MAC+RST+NG method when the topology of the network constraints has a suitable width.

To extend this work, it would be interesting to define new restart policies specific to the case of the decompositions (e.g. by considering local and/or global policies). Moreover, we can propose smarter choices for the root cluster by exploiting specific information (e.g. the number of (no)goods). Finally, this approach could be applied at a meta level, for instance, to address the problem of choosing a suitable tree-decomposition.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] C. Berge, *Graphs and Hypergraphs*, Elsevier, 1973.
[2] F. Boussemart, F. Hemery, C. Lecoutre, and L. Saïs, 'Boosting systematic search by weighting constraints', in *ECAI*, pp. 146–150, (2004).
[3] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners, 'Radio Link Frequency Assignment', *Constraints*, **4**, 79–89, (1999).
[4] R. Dechter and J. Pearl, 'Tree-Clustering for Constraint Networks', *Artificial Intelligence*, **38**, 353–366, (1989).
[5] P. Jégou, S. N. Ndiaye, and C. Terrioux, 'Computing and exploiting tree-decompositions for solving constraint networks', in *CP*, pp. 777–781, (2005).
[6] P. Jégou, S.N. Ndiaye, and C. Terrioux, ''Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs', in *IJCAI*, pp. 112–117, (2007).
[7] P. Jégou, S.N. Ndiaye, and C. Terrioux, 'Dynamic Management of Heuristics for Solving Structured CSPs', in *CP*, pp. 364–378, (2007).
[8] P. Jégou and C. Terrioux, 'Hybrid backtracking bounded by tree-decomposition of constraint networks', *AIJ*, **146**, 43–75, (2003).
[9] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal, 'Recording and Minimizing Nogoods from Restarts', *JSAT*, **1**(3-4), 147–167, (2007).
[10] B. Nadel, *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*, 287–342, Search in Artificial Intelligence, 1988.
[11] J. Petke, *On the bridge between Constraint Satisfaction and Boolean Satisfiability*, Ph.D. dissertation, University of Oxford, 2012.
[12] N. Robertson and P.D. Seymour, 'Graph minors II: Algorithmic aspects of treewidth', *Algorithms*, **7**, 309–322, (1986).
[13] D. J. Rose, 'Triangulated Graphs and the Elimination Process', *Journal of Mathematical Analysis and Application*, **32**, 597–609, (1970).
[14] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, Elsevier, 2006.
[15] D. Sabin and E. Freuder, 'Contradicting Conventional Wisdom in Constraint Satisfaction', in *ECAI*, pp. 125–129, (1994).
[16] D. Sabin and E. Freuder, 'Understanding and Improving the MAC Algorithm', in *CP*, pp. 167–181, (1997).
[17] T. Walsh, 'Search in a small world', in *IJCAI*, pp. 1172–1177, (1999).