Dynamic heuristics for branch and bound search on tree-decomposition of Weighted CSPs

Philippe Jégou, Samba Ndojh Ndiaye, and Cyril Terrioux

LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, samba-ndojh.ndiaye, cyril.terrioux}@univ-cezanne.fr

Abstract. This paper deals with methods exploiting tree-decomposition approaches for solving weighted constraint networks. We consider here the practical efficiency of these approaches by defining five classes of variable orders more and more dynamic which preserve the time complexity bound. For that, we define extensions of this theoretical time complexity bound to increase the dynamic aspect of these orders. We define a constant k allowing us to extend the classical bound from $O(exp(w+1))$ firstly to $O(exp(w+k))$, and finally to $O(exp(2(w+k)))$, where w denotes the "tree-width" of a Weighted CSP. Finally, we assess the defined theoretical extension of the time complexity bound from a practical viewpoint.

1 Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. Modeling a problem as a CSP consists in defining a set X of variables $x_1, x_2, \ldots x_n$, which must be assigned in their respective finite domain, by satisfying a set C of constraints which express restrictions between the different possible assignments. A solution is an assignment of every variable which satisfies all the constraints. Determining if a solution exists is a NP-complete problem. This framework has been extended in order to capture notions like preference or possibility or, when there is no solution, to produce an assignment minimizing a given criterion on constraint satisfaction. Hence, recently, many extensions of the CSP framework have been proposed (e.g. [1–3]).

In this paper, we focus our study on Weighted CSPs (WCSPs) which is a well known framework for soft constraints. In this extension of CSPs, a weight (or a cost) is associated with each tuple of each constraint. So, each assignment has a cost defined by the sum of the costs of all the tuples included in the considered assignment. Solving a WCSP instance requires to find an assignment whose cost is minimum. This task is NP-hard. Many algorithms have been defined in the past years for solving this problem. On the one hand, the usual complete method for solving WCSPs is based on branch and bound search, which, in order to be

efficient, must use both filtering techniques and heuristics for choosing the next variable or value. This approach, often efficient in practice, has an exponential theoretical time complexity in $O(exp(n))$ for an instance having n variables. On the other hand, some other methods are based on the dynamic programming approach [4–9]. Some of them exploit the problem structure like [5, 6, 10, 9]. Exploiting the structure often allows to improve the solving methods and, in particular, to provide better theoretical time complexity bounds. Several bounds exist like the induced width [11] or the tree-height [12, 13]. Yet, the best known complexity bounds are given by the "tree-width" of a CSP (often denoted w). This parameter is related to some topological properties of the constraint graph which represents the interactions between variables via the constraints. It leads to a time complexity in $O(exp(w+1))$. Different methods have been proposed to reach this bound like *Tree-Clustering* [14] (see [15] for a survey and a theoretical comparison of these methods). They rely on the notion of tree-decomposition of the constraint graph. They aim to cluster variables such that the cluster arrangement is a tree. Depending on the instances, we can expect a significant gain w.r.t. enumerative approaches. Most of works based on this approach only present theoretical results. Few practical results have been provided (e.g. [16, 17]). So, we study these approaches by concentrating us on the BTD method (for Backtracking with Tree-Decomposition [18]) which seems to be one of the most effective method proposed until now within the framework of these structural methods.

The problem of finding the best decomposition (w.r.t. the tree-width) has been firstly studied in the literature from a theoretical point of view. More recently, some studies (e.g. [19]) have been realized in the field of CSP, integrating as quality parameter for a decomposition, its efficiency for solving the considered CSP. Yet, these studies do not consider the questions related to an efficient use of the considered decompositions. This paper deals with this question. Given a tree-decomposition, we study the problem of finding good orders on variables for exploiting this decomposition in a branch and bound search like one achieved by BTD. Similar works have been already performed for SAT or CSP (e.g. [20, 21]). As presented in [22, 17], the order on the variables is static and compatible with a depth first traversal of the associated cluster tree. Since enumerative methods highlight the efficiency of dynamic variable orders, we give conditions which allow to exploit in a more dynamic way the tree-decomposition and guarantee the time complexity bound. We propose five classes of orders respecting these conditions, two of them giving more freedom to order variables dynamically. Consequently, their time complexity possess larger bounds: $O(exp(w+k))$ and $O(exp(2(w+k)))$, where k is a constant to parameterize. Based on the properties of these classes, we exploit several heuristics which aim to compute a good order on clusters and more generally on variables. They rely on topological and semantic properties of the WCSP instance. Heuristics based on the expected number of solutions enhance significantly the performances of BTD. Meanwhile, those based on the cluster size or on the dynamic variable ordering heuristic provide often similar improvements and may outperform the first ones on real-world instances. Finally,

we report here experiments to assess the interest of the extensions based on the time complexity bound.

This paper is organized as follows. Section 2 provides basic notions about WCSPs and methods based on tree-decompositions. Then, in section 3, we define several classes of variable orders which preserve the classical bounds for time complexity. Section 4 introduces two extensions giving new time complexity bounds. Section 5 presents the different heuristics we use for guiding the exploration of the cluster tree and variables. Then, in section 6 is devoted to experimental results to assess the practical interest of our propositions. Finally, in section 7, we conclude and outline some future works.

2 Preliminaries

A *constraint satisfaction problem* (CSP) is defined by a tuple (X, D, C). X is a set $\{x_1, \ldots, x_n\}$ of n variables. Each variable x_i takes its values in the finite domain d_{x_i} from D. The variables are subject to the constraints from C. Given an instance (X, D, C), the CSP problem consists in determining if there is a solution (i.e. an assignment of each variable which satisfies each constraint). This problem is NP-complete. In this paper, we focus our study on an extension of CSPs, namely Weighted CSPs (WCSPs). In this extension of CSPs, a weight (or a cost) is associated with each tuple of each constraint. The cost of a tuple allowed by a constraint is 0, while a forbidden one has a cost greater than 0. Then, each assignment has a cost defined by the sum of the costs of all the tuples included in the considered assignment. Solving a WCSP instance requires to find an assignment whose cost is minimum. This task is NP-hard. In this paper, without loss of generality, we only consider binary constraints (i.e. constraints which involve two variables). So, the structure of a WCSP can be represented by the graph (X, C), called the *constraint graph*. The vertices of this graph are the variables of X and an edge joins two vertices if the corresponding variables share a constraint.

Methods providing interesting theoretical time complexity bound often rely on the structure of its constraint graph, and in particular the notion of tree-decomposition of graphs [23]. Let $G = (X, C)$ be a graph, a *tree-decomposition* of G is a pair (E, T) where $T = (I, F)$ is a tree with nodes I and edges F and $E = \{E_i : i \in I\}$ a family of subsets of X, such that each subset (called cluster) E_i is associated to a node of T and verifies:

(i) $\cup_{i \in I} E_i = X$,
(ii) for each edge $\{x, y\} \in C$, there exists $i \in I$ with $\{x, y\} \subseteq E_i$,
(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.

Assume that we have a tree-decompositions of minimal width (w). The reference structural method, Tree-Clustering [14], has a time complexity in $O(exp(w+1))$ while its space complexity can be reduced to $O(n.s.d^s)$ with s the size of the

largest minimal separators of the graph [24]. Note that Tree-Clustering does not provide interesting results in practical cases. So, an alternative approach, also based on tree-decomposition of graphs was proposed in [22]. This method is called BTD (for Backtracking with Tree-Decomposition) and seems to provide empirical results among the best ones obtained by structural methods.

The BTD method proceeds by an enumerative search guided by a static pre-established partial order induced by a tree-decomposition of the constraint network. So, the first step of BTD consists in computing a tree-decomposition of the constraint graph. The computed tree-decomposition induces a partial variable ordering which allows BTD to exploit some structural properties of the graph and so to prune some parts of the search tree. In fact, variables are assigned according to a depth-first traversal of the rooted tree. In other words, we first assign the variables of the root cluster E_1, then we assign the variables of E_2, then E_3's ones, and so on. For example, $(x_1, x_2, \ldots, x_{14})$ or $(x_2, x_1, x_4, x_3, x_6, x_5, x_7, x_9, x_8, x_{13}, x_{10}, x_{12}, x_{11}, x_{14})$ are possible variable orderings for the problem whose constraint graph is presented in figure 1. Furthermore, the tree-decomposition and the variable ordering allow BTD to divide the problem $\mathcal{P}$ into many subproblems. Given two clusters E_i and E_j (with E_j a E_i's son), the subproblem rooted in E_j depends on the current assignment $\mathcal{A}$ on the separator $E_i \cap E_j$. It is denoted $\mathcal{P}_{\mathcal{A},E_i/E_j}$. Its variable set is equal to $Desc(E_j)$ where $Desc(E_j)$ denotes the set of variables belonging to a cluster E_j or to any descendant E_k of E_j in the cluster tree rooted in E_j. The domain of each variable which belongs to $E_i \cap E_j$ is restricted to its value in $\mathcal{A}$. Regarding the constraint set, it contains the constraints which involve at least one variable which exclusively appears in E_j or in a descendant of E_j. For instance, let us consider the WCSP whose constraint graph and a possible tree-decomposition are provided in figure 1. Given an assignment $\mathcal{A}$ on $E_2 \cap E_3$, the variable set of $\mathcal{P}_{\mathcal{A},E_2/E_3}$ is $Desc(E_3) = \{x_5, x_6, x_7, x_8, x_9\}$, and its constraint set is $\{c_{57}, c_{59}, c_{67}, c_{69}, c_{78}, c_{79}\}$ (with c_{ij} the constraint involving the variables x_i and x_j). Note that the constraint c_{56} does not belong to its constraint set because x_5 and x_6 appear both in E_2. Remark that the definition of subproblems defines a partition of the constraint set. Such a partition ensures that BTD takes into account each constraint only once and so that it safely computes the cost of any assignment. Finally, the tree-decomposition notion permits to define the *valued structural good* notion. A structural valued good of E_i with respect to E_j (with E_j a E_i's son) is a pair $(\mathcal{A}, v)$ with $\mathcal{A}$ an assignment on $E_i \cap E_j$ and v the optimal cost of the subproblem $\mathcal{P}_{\mathcal{A},E_i/E_j}$.

To satisfy the bounds of complexity, the variable ordering exploited in BTD is related to the cluster ordering. Formally, consider (E, T) a tree-decomposition of the CSP where $T = (I, F)$ is a tree. We suppose that the elements of $E = \{E_i : i \in I\}$ are indexed w.r.t. the notion of *compatible numeration*. A numeration on E compatible with a prefix numeration of $T = (I, F)$ with E_1 the root is called compatible numeration. An order $\preceq_X$ of variables of X such that $\forall x \in E_i$, $\forall y \in E_j$, with $i < j$, $x \preceq_X y$ is a compatible enumeration order. The numeration on the clusters gives a partial order on the variables since the variables in E_i

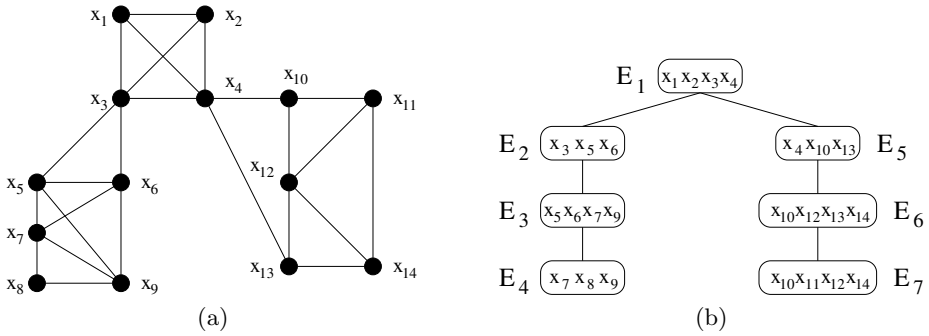

Fig. 1. (a) A graph and (b) a possible tree-decomposition.

are assigned before those in E_j if $i < j$, except variables in the descent of a good, namely those located in the subproblem rooted on the cluster containing the good. In fact, using goods allows not to explore twice subproblems if their optimal cost is known. If we use a good $(\mathcal{A}, v)$ to avoid visiting again a subtree, we know that the variables in it can be optimally assigned with a cost v. So BTD does not assign them effectively, but they are considered done. They are named *assignable variables* thanks to a good. Of course, if we are interested by providing an optimal assignment, an additional work must be performed to assign these variables at the end of the search [17]. Thus the variables in E_j are assigned if the variables in E_i are either already assigned or assignable thanks to a good. To complete this order, we have to choose variable ordering heuristics inside a cluster. Finally, a compatible enumeration order on the variables is given by a compatible numeration on clusters and a variable order in each cluster.

In [17], the results were presented without heuristics for the choice of the clusters and thus the choice of the variables, except on the level of the order used inside the cluster which corresponded to a traditional dynamic order. Obviously, the variable ordering have a great impact on the efficiency of enumerative methods. Thus, we study here how the benefits of variable orderings can be fully exploited in BTD. Nevertheless, to guarantee the time complexity bounds, it is necessary to respect some conditions. So, in the next section, we define classes of orders guaranteeing complexity bounds.

3 Dynamic orders in $O(exp(w + 1))$

The first version of BTD was defined with a compatible static variable ordering. We prove here that it is possible to consider more dynamic orders without loosing the complexity bounds. The defined classes contain orders more and more dynamic. These orders are in fact provided by the cluster order and the variable ordering inside each cluster.

Definition 1 *Let (X, D, C) be a WCSP and (E, T) a tree-decomposition of the graph (X, C), we define:*

- ***Class 1. Enumerative static order**. It is a static order of assignment of variables which is compatible.*
- ***Class 2. Static cluster order and dynamic variable order**. The cluster order is a compatible order (thus static). Yet, inside each cluster, the variable order is dynamic. Let Y be an assignment, if $x_i \in E_i$ is the last assigned variable in Y, then $\forall E_j \in E$, $j < i$, $\forall x_j \in E_j$, $x_j \in Y$. So, a variable $x_i \in E_i$ is assigned if and only if all the variables in clusters E_j, $j < i$, are already assigned or assignable thanks to goods. So one can observe that it is possible to have assignable variables thanks to goods in the assignment Y, which are not effectively assigned.*
- ***Class 3. Dynamic cluster order and dynamic variable order**. Let Y be an assignment, $x_i \in E_i$, if $x_i \in Y$, then $\forall E_j \in E$, $i \neq j$ such that E_j is on the path from the root cluster E_1 to E_i, $\forall x_j \in E_j$, $x_j \in Y$. So, a variable $x_i \in E_i$ is assigned if and only if all the variables in clusters on the path from the root cluster E_1 to E_i are assigned first.*
- ***Class ++. Enumerative dynamic order**. The variable ordering is completely dynamic. Consequently, the assignment order is not necessarily an enumerative compatible order. There is no restriction due to cluster tree.*

The defined classes form a hierarchy since we have: *Class 1* $\subset$ *Class 2* $\subset$ *Class 3* $\subset$ *Class ++*. In [17], the experiments use *Class 2* orders. Formally, only the orders of the *Class 1* are compatible. Nevertheless, for an unique assignment, one can find an order in the *Class 1* that coincides with the order of the *Class 3*. This property gives to the *Class 3* (thus *Class 2*) orders the ability of recording goods and using them to prune branches in the same way *Class 1* orders do. The *Class ++* gives a complete freedom. Yet, it does not guarantee the time complexity bounds because sometimes it is impossible to record goods. Indeed an order of *Class ++* may lead to assign some variables of a cluster E_j (with E_j a son of a cluster E_i) without having assigned the variables of the separator $E_i \cap E_j$. By so doing, we cannot safely compute the optimal solution of the subproblem rooted in E_j and so it is impossible to record a good on $E_i \cap E_j$. Hence, a subproblem may be solved several times and thus the time complexity bound is not guaranteed anymore. Meanwhile, the *Class 3* orders guarantee this bound.

Theorem 1 *Let the enumerative order be in the Class 3, the time complexity of BTD is $O(exp(w + 1))$.*

Proof We consider a cluster E_j in the cluster tree, and we must prove that any assignment on E_j is computed only once. Let E_i be the cluster parent of E_j and suppose that for a current assignment the last assigned variables are in E_i. Since the order is in the *Class 3*, the variables of the clusters on the path from the root to E_i are already assigned and those in the subtree rooted on E_j not yet. An assignment $\mathcal{A}$ on $E_i \cap E_j$ is computed when the variables in E_i are assigned

before those in the subproblem rooted in E_j. Solving this subproblem leads to the computation of its optimal cost v. Then, $(\mathcal{A}, v)$ is recorded as a good. Let $\mathcal{A}'$ be the assignment on E_j. The next assignment of variables in E_i leading to $\mathcal{A}$ on $E_i \cap E_j$ will not be pursued on the subproblem rooted on E_j. $\mathcal{A}'$ is not computed twice, only the variables in $E_i \cap E_j$ are assigned again. So the time complexity is $O(exp(w+1))$. □
The properties of the *Class 3* offer more possibilities in the variable ordering. So it is possible to choose any cluster to visit next since the variables on the path from the root cluster to that cluster are already assigned. And in each cluster, the variable ordering is totally free. In the next section, we propose two natural extensions of the complexity bound.

4 Bounded extensions of dynamic orders

We propose two extensions based on the ability given to the heuristics to choose the next variables to assign not only in one cluster, but also among k variables in a path rooted on the cluster that verifies some properties. So, we define two new classes of orders extending *Class 3*. First, we propose a generalization of the tree-decomposition definition.

Definition 2 *Let $G = (X, C)$ be a graph and k a non nil positive integer, the set of directed k-covering tree-decompositions of a tree-decomposition (E, T) of G with E_1 its root cluster, is defined by the set of tree-decompositions (E', T') of G that verify:*

- *$E_1 \subset E'_1$, E'_1 the root cluster of (E', T')*
- *$\forall E'_i \in E'$, $E'_i \subset E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_K}$, with $E_{i_1} \ldots E_{i_K}$ a path in T*
- *$|E'_i| \leq w^+ + k$, where $w^+ = max_{E_i \in E}|E_i|$*

Now, we give a definition of the *Class 4*.

Definition 3 *Let (X, D, C) be a WCSP, (E, T) a tree-decomposition of the graph (X,C) and k a non nil positive integer. A variable order is in the **Class 4**, if this order is in the Class 3 for some directed k-covering tree-decomposition of (E', T') such that $E_{i_1} \subset E'_i$.*

This definition enforces the order of one assignment to be in the *Class 3*. So we derive a natural theorem:

Theorem 2 *Let the enumerative order be in the Class 4 with constant k, the time complexity of BTD is $O(exp(w^+ + k))$.*

Proof This proof is similar to one given for Class 3 since we can consider that BTD runs on a tree-decomposition (E', T') of width $w^+ + k$. □

A second extension is possible in exploiting during the search, a dynamic computing of the tree-decomposition. Then, the time complexity bound changes because sometimes it would be impossible to record goods.

Definition 4 *Let (X, D, C) be a WCSP, (E, T) a tree-decomposition of the graph (X,C) and k a non nil positive integer. A variable order is in the* ***Class 5****, if for any assignment, its order is in the Class 3 for some directed k-covering tree-decomposition of (E, T).*

Theorem 3 *Let the enumerative order be in the Class 5, the time complexity of BTD is $O(exp(2(w^+ + k)))$.*

Proof Let (X, D, C) be a WCSP, (E, T) a tree-decomposition of the graph (X, C) and E_1 its root cluster. We have to prove that any assignment on a set V of $2(w^+ + k)$ variables on a path of the tree T is computed only once. Let $\mathcal{A}$ be an assignment containing V. The variable order of this assignment is in *Class 3* for a directed k-covering tree-decomposition (E', T') of (E, T) with E_1 the root cluster. The size of the clusters in (E', T') are bound by $w^+ + k$, so the set V is recovered by at least two clusters. Let $E'_{i_1} \dots E'_{i_r}$ be a path on (E', T') covering V. The solving of the subproblem rooted on E'_{i_1} with the assignment $\mathcal{A}$ leads to the recording of goods on the separators of these clusters. Actually, we record goods on the separators of the clusters of (E, T) covered by $E'_{i_1} \dots E'_{i_r}$ because they are minimal in order to split the problem into several connected components. And they are included into the separators of clusters in each k-covering tree-decomposition of (E, T). We suppose that E'_{i_1} is not the root cluster of (E', T'). If $r = 2$ then the size of E'_{i_1} and E'_{i_2} is $2(w^+ + k)$, what leads to the recording of a good on the separator of E'_{i_1} and its parent and another on the separator of E'_{i_1} and E'_{i_2}. If $r > 2$, we record at least 2 goods on the separators of the clusters $E'_{i_1} \dots E'_{i_r}$. Let $\mathcal{B}$ be an assignment that contains the variables ordered before the variables in V in the *Class 3* order. When we try to extend $\mathcal{B}$ on V with the same values in $\mathcal{A}$, one of the good will be computed first. Thus before all the variables in V are assigned, the search is stopped thanks to this good. So $\mathcal{A}$ is computed only once. Now, if E'_{i_1} is the root cluster of (E', T'), then V contains E_1. So, at least one good is recorded between E'_{i_1} and E'_{i_2}. If $r > 2$, then the assignment $\mathcal{A}$ will not be computed again because V contains at least 2 goods. If $r = 2$, only one good is recorded. Nevertheless, for each other directed k-covering tree-decompositions of (E, T), (E'', T''), a *Class 5* order assigns the variables in E_1 before at least $w^+ + k$ other variables in V. In fact, the variables in E_1 are also in the root cluster of (E'', T'') and are assigned first w.r.t the *Class 3* order. As soon as the variables in E_1 are assigned, the recorded good allows to stop the search in the rest of V since the optimal cost of the corresponding subproblem is already known. We prove that any assignment on V is computed only once. □

Note that the new defined classes are included in the hierarchy presented in section 3: *Class* $i \subset$ *Class* j, if $i < j$ and for $1 \leq i < j \leq 5$, with also *Class 5* $\subset$ *Class ++*.

To define the value of k, we have several approaches to choose variables to group. A good one consists in trying to reduce the value of the parameter s and, by this way, to enhance the space complexity bound. Then, we can observe that grouping clusters with large separators permits to achieve a significant reduction of s.

5 Heuristics

In this section we define several heuristics with the aim in view to improve in significant way the performances of BTD w.r.t. runtime.

5.1 Cluster orders

We propose here several heuristics computing the order the clusters are visited for the *Classes 1, 2* and 3. They are static for the *Class 1* and dynamic for the *Classes 2* and 3. They consist in choosing the first visited cluster (called the root cluster) and ordering the sons of each cluster. Precisely, we assign first the variables in the root cluster and recursively we assign the variables in the trees rooted on its son clusters according to the son order, considering the sons as the roots of the subproblems. Nevertheless, all these orders are used under the hypothesis the early use of goods does not enforce another order. Indeed this early use of goods improves a lot the method, by detecting earlier failures. In fact as soon as all the variables in the separator between the current cluster and one of its sons are assigned, we check whether this assignment is a good. If so, we do not explore the subtree rooted on this son cluster.

Static orders A static order is defined before the search begins. We propose criteria for the choice of the root cluster.

- *minexp*: this heuristic is based on the expected number of partial solutions of clusters [25] and on their size. Exploiting the expected number of solutions may appear surprising in the WCSP framework. However, some subproblems may have solutions while the whole problem has none. If so, it could be interesting to begin the search with the subproblems having no solution since they have a positive cost, what may result in increasing quickly the lower bound. The heuristic chooses as root cluster one which minimizes the ratio between the expected number of solutions and the size of the cluster. It allows to start the exploration with a large cluster having few solutions or no solution.
- *size*: we have here a local criteria: we choose the cluster of maximum size as root cluster
- *bary*: it is a global criterion based on the location of the cluster in the tree. For this criterion, we use the notion of distance, noted $dist(x, y)$, between two vertices x and y of a graph G, which is defined by the length of a shortest path between x and y. A *barycentre* of G is a vertex x s.t. x minimizes $\Sigma_{y \in X} dist(x, y)$. The *bary* heuristic chooses a barycentre cluster as a root cluster.

Likewise, we propose heuristics for ordering cluster sons.

- $minexp_s$: this heuristic is similar to *minexp* and orders the son clusters according to the increasing value of their ratio.
- $minsep_s$: we order the son clusters according to the increasing size of their separator with their parent.

Dynamic orders A dynamic order is defined during the search. But, the choice of the root cluster is done at the beginning of the search. So one can only use static heuristics to choose the root. We also propose a new heuristic: nv. The dynamic variable ordering heuristics improve very significantly the runtime of enumerative methods. To derive benefit of this property, we choose a dynamic variable ordering heuristic and the root cluster is one containing the first variable w.r.t. the chosen variable order. The dynamic aspect of the cluster orders is in the son cluster ordering.

- $minexp_{sdyn}$: the next cluster to visit minimizes the ratio between the current expected number of solutions and the size of the cluster. The current expected number of solutions of a cluster is modified by filtering the domains of unassigned variables. So we compute this number for unordered clusters as soon as their parent is fully instantiated. So the choice of the next cluster is more precise.
- nv_{sdyn}: this heuristic is similar to nv. We visit first the son cluster where appears the next variable in the variable order among the variables of the unvisited son clusters.

5.2 Variable orders

We define here static and dynamic variable orders according to which the variables inside a cluster are assigned.
Static orders A static order is defined before the search begins.

- mdd: the variables are ordered according to the increasing value of the ratio domain/degree. This heuristic gives good results compared to other static ones.

Dynamic orders A dynamic order is defined during the search.

- mdd_{dyn}: the next variable to assign minimizes the ratio domain/degree. The current ratio of a variable is modified by the domain filtering. So we compute again this number each time the domain is filtered. This heuristic gives very good results.

5.3 Heuristics for grouping variables in the *Class 4*

Grouping variables allows more freedom for dynamic variable ordering heuristics which may improve significantly the enumerative methods runtime. Furthermore, it is necessary to find a good value of the parameter k besides which BTD does not profit sufficiently of the problem structure and therefore its time complexity increases a lot. We propose several criteria for grouping variables which can be seen as a preliminary step before computing an order of the *Class 4*.

- sep: this heuristic has one parameter which is the maximum size of separators. We merge clusters $< parent, son >$ if their separator size exceeds the value of the parameter.

– *pv*: this heuristic has one parameter which is the minimum number of proper variables in a cluster. A proper variable of a cluster is a variable of a cluster which does not belong to the parent cluster. We merge a cluster with its parent if its number of proper variables is under the parameter.

All the heuristics we have defined, try to satisfy the first-fail principle, doing first the most constrained choices.

6 Experimental study

Applying a structural method on an instance generally assumes that this instance presents some particular topological features. So, our study is performed on instances having a structure which can be exploited by structural methods. In practice, we assess here the proposed strategies on particular random WCSPs and real-world instances in order to point up the best ones w.r.t. the WCSP solving. Regarding the random instances, we exploit partial structured instances. A random structured instance of a class (n, d, w, t, s, n_c) is built according to the model described in [18]. This structured instance consists of n variables having d values in their domain. Its constraint graph is a clique tree with n_c cliques whose size is at most w and whose separator size does not exceed s. Each constraint forbids t tuples. For each forbidden tuple, a weight between 1 and 10 is associated randomly. Then, for building a partial structured instance of a class (n, d, w, t, s, n_c, p), we remove randomly $p\%$ edges from a structured instance of a class (n, d, w, t, s, n_c). Secondly, we experiment the proposed heuristics on some real-world instances, namely radio-link frequency assignment problems from the FullRLFAP archive (for more details, see [26]).

All these experimentations are performed on a Linux-based PC with a Pentium IV 3.2GHz and 1GB of memory. For each considered random partial structured instance class, the presented results are the average on instances solved over 30. In the following tables, the letter M means that at least one instance cannot be solved because it requires more than 1GB of memory.

In [19], a study was performed on triangulation algorithms to find out the best way to compute a good tree-decomposition w.r.t. CSP solving. As MCS [27] obtains the best results and is very easy to implement, we use it to compute tree-decompositions in this study.

In the following, the results for Class 5 are not presented since we cannot get good results. Table 1 shows the runtime of BTD based on FC with several heuristics of Classes 1, 2 and 3 on random partial structured instances. Clearly, we observe that the choice of the root cluster seems more important than the son ordering. Indeed, the obtained results appear to be similar as soon as we choose the root cluster with the same heuristic. Moreover, the main difference between the heuristics are observed for different choices of root cluster. The son cluster ordering has a limited effect because the considered instances have a few son clusters reducing the possible choices and so their impact. We can expect a more important improvement for instances with more son clusters. The heuristics *size* and *minexp* often provide interesting results but sometimes make bad choices.

Table 1. Runtime (in s) on random partial structured CSPs with *mdd* for class 1 and mdd_{dyn} for classes 2 and 3.

Instance	Class 1	Class 2			Class 3		
	size	*bary*	*minexp*	*size*	*minexp*	*size*	*nv*
	$minsep_s$	$minsep_s$	$minexp_s$	$minsep_s$	$minexp_{sdyn}$	nv_{sdyn}	nv_{sdyn}
(75,10,15,30,5,8,10)	M	22.31	M	M	M	M	M
(75,10,15,30,5,8,20)	3.27	4.77	6.13	3.34	6.24	2.88	M
(75,10,15,33,3,8,10)	8.30	6.16	7.90	8.67	7.87	8.82	5.36
(75,10,15,34,3,8,20)	2.75	2.29	3.42	2.82	3.52	2.84	2.14
(75,10,10,40,3,10,10)	11.81	1.33	3.02	11.89	4.73	11.87	1.43
(75,10,10,42,3,10,20)	1.02	0.67	0.76	1.02	0.83	1.03	0.79
(75,15,10,102,3,10,10)	11.76	3.74	12.10	12.07	12.09	11.70	4.93
(100,5,15,13,5,10,10)	M	M	M	M	M	M	M

Table 2. Runtime (in s) on random partial structured CSPs with mdd_{dyn} for class 4 orders and *sep* heuristic (the separator size is bounded to 5).

Instance	Class 4				
	minexp	*size*	*minexp*	*size*	*nv*
	$minexp_s$	$minsep_s$	$minexp_{sdyn}$	nv_{sdyn}	nv_{sdyn}
(75,10,15,30,5,8,10)	9.42	18.99	8.69	18.30	16.77
(75,10,15,30,5,8,20)	1.65	1.67	1.56	1.53	2.32
(75,10,15,33,3,8,10)	5.22	4.31	5.26	4.18	3.24
(75,10,15,34,3,8,20)	1.50	1.61	1.48	1.58	1.40
(75,10,10,40,3,10,10)	0.58	0.81	0.58	0.85	0.52
(75,10,10,42,3,10,20)	0.41	0.42	0.51	0.41	0.38
(75,15,10,102,3,10,10)	5.50	4.73	5.41	4.63	3.13
(100,5,15,13,5,10,10)	9.40	9.05	9.60	9.10	11.71

The heuristic *nv* leads to promising results except for one instance class which cannot be solved due to the required amount of memory. The heuristic *bary* seems the more robust heuristic: it obtains good results and succeeds in solving the instances of class $(75, 10, 15, 30, 5, 8, 10)$ while BTD with any other heuristics requires a too large amount of memory space. This memory problem can be solved by exploiting a *Class 4* order with the *sep* heuristic for grouping variables. Table 2 gives the runtime of BTD for this class with a separator size bounded to 5. When we analyze the value of the parameter k, we observe that in general, its value is limited (between 1 to 3). The results of Class 4 orders improve significantly ones obtained for the Classes 2 and 3. Like previously, the results are mostly influenced by the choice of the root cluster. The best results are obtained by $minexp + minexp_s$ and $minexp + minexp_{sdyn}$. For most of instance classes, the other heuristics obtain similar results. Unfortunately, for some other classes, the bad choices they make for the root cluster significantly increase the runtime.

Finally, in table 3, we assess the behaviour of the proposed heuristics on some real-world instances. Surprisingly, the considered Class 1 order obtains the best

Table 3. Runtime (in s) on some instances from the FullRLFAP archive with *mdd* for class 1 and mdd_{dyn} for classes 2 and 3.

Instance	Class 1	Class 2			Class 3		
	size $minsep_s$	*bary* $minsep_s$	*minexp* $minexp_s$	*size* $minsep_s$	*minexp* $minexp_{sdyn}$	*size* nv_{sdyn}	*nv* nv_{sdyn}
SUB_0	9.48	5.71	9.65	9.57	9.62	9.64	9.52
SUB_1	448	511	516	515	516	518	520
SUB_2	520	703	705	701	702	700	702
SUB_3	5,575	-	6,596	6,553	6,640	6,595	6,570
SUB_4	8,146	-	9,677	9,693	9,780	9,672	9,694

results. This result can be explained by the weak size of the considered instances (between 16 and 22 variables). It ensures that the cost of computing dynamically the variable ordering or the son ordering is not compensated. The same reason explains the close results obtained for Classes 2 and 3 orders. Whereas it obtains the more promising results on random instance, the heuristic *bary* requires more than 8 hours for solving the SUB_3 and SUB_4.

7 Discussion and Conclusion

In this article, we have studied the WCSP solving methods based on tree-decompositions in order to improve their practical interest. This study was done both theoretically and empirically. The analysis of the variable orders allows us to define more dynamic heuristics without loosing the time complexity bounds. So, we have defined classes of variable orders which allow a more and more dynamic ordering of variables and preserve the theoretical time complexity bound. This bound has been extended to enforce the dynamic aspect of orders that has an important impact on the efficiency of enumerative methods. Even though these new bounds are theoretically less interesting that the initial, it allows us to define more efficient heuristics which improve significantly the runtime of BTD. This study, which could not be achieved previously, takes now an importance for solving hard instances with suitable structural properties.

We have compared the classes of variable orders with relevant heuristics w.r.t. CSP solving. This comparison points up the promising results obtained by *Class 4* orders. These orders give more freedom to the variable ordering heuristic while their time complexity is $O(exp(w + k))$ where k is a constant to parameterize. Note that for the most dynamic class (the Class 5), we get a time complexity in $O(exp(2(w + k)))$. It seems that this bound should be too large to expect a significant practical improvement.

The experimental study presented in this paper is a preliminary step. We only assess the interest of some heuristics. Other variable heuristics must be studied (e.g. the jeroslow-like heuristic [28]). Likewise, for the choice of a root cluster or the son cluster ordering, we must propose heuristics well-adapted to the WCSP problem. For instance, the heuristic based on the expected number of solution

must be extended by taking into account the weights associated to each tuple. Then, for *Class 4*, we aim to improve the criteria used to compute the value of k and to define more general ones by exploiting better the problem features. Finally, these experiments must be performed by exploiting BTD jointly with local consistency techniques [29].

Acknowledgments This work is supported by a "programme blanc" ANR grant (STAL-DEC-OPT project).

References

1. E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
2. S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semirings. In *Proceedings of IJCAI*, pages 624–630, 1995.
3. T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proceedings of IJCAI*, pages 631–637, 1995.
4. G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *Proceedings of AAAI*, pages 181–187, 1996.
5. A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, University of Maastricht, November 1999.
6. P. Meseguer and M. Sánchez. Tree-based Russian Doll Search. In *Proceedings of CP Workshop on soft constraint*, 2000.
7. P. Meseguer and M. Sánchez. Specializing Russian Doll Search. In *Proceedings of CP*, pages 464–478, 2001.
8. P. Meseguer, M. Sánchez, and G. Verfaillie. Opportunistic Specialization in Russian Doll Search. In *Proceedings of CP*, pages 264–279, 2002.
9. J. Larrosa and R. Dechter. Boosting Search with Variable Elimination in Constraint Optimization and Constraint Satisfaction Problems. *Constraints*, 8(3):303–326, 2003.
10. J. Larrosa, P. Meseguer, and M. Sánchez. Pseudo-Tree Search with Soft Constraints. In *Proceedings of ECAI*, pages 131–135, 2002.
11. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.
12. E. Freuder and M. Quinn. Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems. In *Proceedings of IJCAI*, pages 1076–1078, 1985.
13. R. Marinescu and R. Dechter. AND/OR tree search for constraint optimization. In *Proceedings of CP workshop on Soft Constraints and Preferences*, 2004.
14. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
15. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
16. G. Gottlob, M. Hutle, and F. Wotawa. Combining hypertree, bicomp and hinge decomposition. In *Proceedings of ECAI*, pages 161–165, 2002.
17. P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proceedings of ECAI*, pages 196–200, 2004.
18. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.

19. P. Jégou, S. N. Ndiaye, and C. Terrioux. Computing and exploiting tree-decompositions for solving constraint networks. In *Proceedings of CP*, pages 777–781, 2005.
20. J. Huang and A. Darwiche. A structure-based variable ordering heuristic for SAT. In *Proceedings of IJCAI*, pages 1167–1172, 2003.
21. P. Jégou, S. N. Ndiaye, and C. Terrioux. An extension of complexity bounds and dynamic heuristics for tree-decompositions of CSP. In *Proceedings of CP*, 2006.
22. C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of CP*, pages 709–723, 2003.
23. N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *Algorithms*, 7:309–322, 1986.
24. R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.
25. B. Smith. The Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings of ECAI*, pages 100–104, 1994.
26. C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4:79–89, 1999.
27. R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3):566–579, 1984.
28. S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving Max-SAT as weighted CSP. In *Proceedings of CP*, 2003.
29. S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proceedings of AAAI*, 2006.