Strategies and heuristics for exploiting tree-decompositions of constraint networks

Philippe Jégou and **Samba Ndojh Ndiaye** and **Cyril Terrioux** [1]

Abstract. This paper deals with methods exploiting tree-decomposition approaches for solving constraint networks. We consider here the practical efficiency of these approaches, by defining three classes of variable orders more and more dynamic which preserve the time complexity bound and an extension of this theoretical time complexity bound to increase the dynamic aspect of these orders. We propose heuristics in each class to improve the runtime of the methods. Then, we study empirically the practical interest of the proposed heuristics in order to point up the most interesting ones. Finally, the proposed theoretical extension of the time complexity bound is assessed from a practical viewpoint.

1 Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. Modelling 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.

The usual method for solving CSPs is based on backtracking 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 $O(e.d^n)$ for an instance having n variables and e constraints and whose largest domain has d values. Several works have been developed, in order to provide better theoretical complexity bounds according to particular features of the instance. 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 $O(n.d^{w+1})$. Different methods have been proposed to reach this bound like *Tree-Clustering* [2] (see [4] 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. Yet, the space complexity, often linear for enumerative methods, may make such an approach unusable in practice. It can be reduced to $O(n.s.d^s)$ with s the size of the largest minimal separators of the graph [1]. Several works based on this approach have been performed. Most of them only present theoretical results. Except [7, 8, 3], no practical results have been provided. In [7], the method BTD based on enumerative approaches allows a more efficient exploitation of structural properties.

While the problem of finding the best decomposition has been studied in the literature firstly from a theoretical point of view, recently, some studies have been realized in the field of CSP, integrating as quality parameter for a decomposition, its efficiency for solving the considered CSP [5]. Nevertheless, these studies did not consider the questions related to an efficient exploitation 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 the first version of BTD, the variable order was static and compatible with a depth first traversal of the associated cluster tree. Enumerative methods highlight the efficiency of dynamic variable orders. We give conditions that permit to exploit in a more dynamic way the tree-decomposition and guarantee the time complexity bound. We propose three classes of orders respecting these conditions and an extension giving more freedom to order variables dynamically with a new time complexity bound. Based on the properties of these classes, we proposed several heuristics which aim to compute a good order on clusters and more generally on variables. They rely on topological and semantic properties of CSP instance. Heuristics based on the expected number of solutions enhance significantly the performances of BTD.

Finally, we report here experiments to evaluate the interest of our heuristics and for the extensions based on time complexity.

This paper is organized as follows. The next section provides the basic notions about CSPs and methods based on tree-decompositions. Then, we define several classes of variable orders in section 3 and an extension of one class giving a new time complexity bound in section 4. We propose heuristic methods for guiding the exploration of the cluster tree and variables in section 5. Section 6 is devoted to experimental results to assess the practical interest of our propositions. Finally, in section 7, we conclude and we 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 an assignment of each variable which satisfies each constraint. This problem is NP-complete. In this paper, without loss of generality, we only consider

[1] LSIS - UMR CNRS 6168, Université Paul Cézanne (Aix-Marseille 3) {philippe.jegou, samba-ndojh.ndiaye, cyril.terrioux}@univ-cezanne.fr

binary constraints (i.e. constraints which involve two variables). So, the structure of a CSP 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.

Tree-Clustering [2] is the reference method for solving CSPs thanks to the structure of its constraint graph. It is based on the notion of tree-decomposition of graphs [9]. Let $G = (X, C)$ be a graph, a *tree-decomposition* of G is a pair $(E, \mathcal{T})$ where $\mathcal{T} = (I, F)$ 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 a node of $\mathcal{T}$ and verifies:

- $\cup_{i \in I} E_i = X$,
- for each edge $\{x, y\} \in E$, there exists $i \in I$ with $\{x, y\} \subseteq E_i$,
- for all $i, j, k \in I$, if k is in a path from i to j in $\mathcal{T}$, then $E_i \cap E_j \subseteq E_k$.

The width of a tree-decomposition $(E, \mathcal{T})$ 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.

The time complexity of Tree-Clustering is $O(n.d^{w+1})$ (denoted $O(exp(w + 1))$) 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 [1]. Note that Tree-Clustering did not provide interesting results in practical cases. So, an alternative approach, also based on tree-decomposition of graphs was proposed in [7]. This method is called BTD and seems to provide empirical results among the best ones obtained by structural methods.

The BTD method (for Backtracking with Tree-Decomposition) 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.

The obtained tree-decomposition allows to exploit some structural properties of the graph, during the search, in order to prune some branches of the search tree, what distinguishes BTD from other classical techniques. Firstly, the order for the instantiation of the variables is induced by the considered tree-decomposition of the constraint graph. Secondly, some parts of the search space will not be visited again as soon as their consistency is known. This is possible by using the notion of *structural good*. A good is a consistent partial assignment on a set of variables (a separator) such that the part of the CSP located after the separator is consistent and admits a solution compatible with the good. So, it is not necessary to explore this part because we know its consistency. Thirdly, some parts of the search space will not be visited again if we know that the current instantiation leads to a failure. This is possible in applying the notion of *structural nogood*. A structural nogood is a particular kind of nogood justified by structural properties of the constraints network: the part of the CSP located after the nogood is not consistent (a nogood is a consistent assignment of a separator of the graph).

To satisfy the bounds of complexity, the ordering exploited in BTD in the assignment of variables is related to the cluster ordering. Formally, consider (E, T) a tree-decomposition of the CSP where $\mathcal{T} = (I, F)$ 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 $\mathcal{T} = (I, F)$ 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 the E_i are assigned before those in E_j if $i < j$. 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 an order on the variables in each cluster.

The experimental results given in [7] were obtained without using good heuristics to guide the search except for variable ordering in clusters which was dynamic. 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 we define classes of orders guaranteeing complexity bounds.

3 Complexity bounds and orders

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 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 in each cluster.

- **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 a set of variables, 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. In [7], the experiments use this kind of orders.
- **Class 3. Dynamic cluster order and dynamic variable order**. Let Y be a set of variables, $x_i \in E_i$, 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 4. 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: *Class i* $\subset$ *Class j*, if $i < j$. 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 coincide with the order of the *Class 3*. This property gives to the *Class 3* (thus *Class 2*) orders the ability of recording goods and nogoods and using them to prune branches in the same way *Class 1* orders do. The *Class 4* gives a complete freedom. Yet, it does not guarantee the time complexity bounds because sometimes it is impossible to record goods and nogoods. Indeed, let the cluster E_j be a son of the cluster E_i, we suppose that the enumerative order assigns the variables in E_i except those in $E_i \cap E_j$, as well as the variables in the clusters which are on the path from the root cluster to E_i. Let x, the next variable to assign, be in E_j and not in $E_i \cap E_j$. If the resolution of the subtree rooted on E_j leads to a failure, it is impossible to record a nogood on $E_i \cap E_j$ because the instantiations following the assignment of x, depend on it and the filtering induced. If the subproblem has a solution, we can record a good. Actually, this solution is a consistent extension of the assignment on $E_i \cap E_j$ which is a good. A nogood not recorded could be computed again and enabled a pruning of branches. Thus the time complexity bound is not

guaranteed anymore. Meanwhile, the *Class 3* orders guarantee this bound.

Theorem 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. A consistent assignment $\mathcal{A}$ on $E_i \cap E_j$ is computed when the variables in E_i are assigned and before those in the subproblem rooted in E_j. Solving this subproblem leads to a failure or a solution. In each case, $\mathcal{A}$ is recorded as a good or nogood. 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. Thus 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 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 an extension of the complexity bound.

4 An extension of the dynamic order which preserves complexity bounds

We propose an extension 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 a new class of orders similar to *Class 3*.

Let $G = (X, C)$ be a graph, the set of generalized tree-decompositions related to a tree-decomposition $(E, \mathcal{T})$ with root E_1 of G and k a no nil positive integer, is defined by the set of the tree-decompositions (E', T') of G that verify: for all E'_i a subset of X, there are $E_{i_1} \dots E_{i_K}$, subsets of X on a path of $(E, \mathcal{T})$ such that:

- $E_{i_1} \dots E_{i_K}$ is a path,
- $E'_i \subset E_{i_1} \cup E_{i_2} \cup \dots \cup E_{i_K}$,
- $|E'_i| \leq w + k$.

Let (X, D, C) be a CSP, $(E, \mathcal{T})$ a tree-decomposition with E_1 the root cluster of the graph (X,C) and k positive integer no nil. A variable order is in the *Class 3 k-extended*, if for any assignment, its order is in the *Class 3* for some generalized tree-decomposition related to $(E, \mathcal{T})$ and k.

This definition enforces the order of one assignment to be in the *Class 3*. The dynamic computing of the tree-decomposition changes the time complexity bound because sometimes it would be impossible to record goods and nogoods. Nevertheless, the use of the parameter k allows another time complexity bound.

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

The proof can be found in [6].

We have several approaches to choose variables to group. A good one consists in trying to reduce the value of the parameter s, by this way to enhance the space complexity bound. Grouping clusters with large separators permits to achieve a significant reduction of s.

5 Heuristics

In this section we define several heuristics that improve often in significant way the performances of BTD w.r.t. runtime. Firstly, we define the notion of expected number of solutions that appears in several heuristics.

5.1 Estimating the number of partial solutions

The assessment of the number of solutions by computing its mathematical expectation takes into account the problem density, the domain size and the constraint tightness. Hence, it allows us to make some choices to traverse the cluster tree by choosing first the clusters which have a minimum number of solutions. By so doing, we can expect the failures to occur earlier. The mathematical expectation of the number of solutions of a CSP has been introduced in [11]. Let us consider a CSP $P = (X, D, C)$ and, for $1 \leq i < j \leq n$, the tightness t_{ij} of the constraint between x_i and x_j. In the following formula, the tightness t_{ij} is defined as the ratio between the number of allowed tuples and the number of possible tuples, if the constraint exist, otherwise $t_{ij} = 0$. The mathematical expectation of the number of solutions of P is $E(P) = d^n \prod_{1 \leq i < j \leq n} (1 - t_{ij})$. If we consider a complete assignment $a = (a_1, \dots, a_n)$, a is a solution of P if $1 \leq i < j \leq n$, a_i and a_j are compatible w.r.t. the constraint c_{ij}. The probability of this event is $(1 - t_{ij})$. So the probability that a is a solution is $\prod_{1 \leq i < j \leq n} (1 - t_{ij})$. As the total number of assignments is d^n, the expected number of solutions is $E(P) = d^n \prod_{1 \leq i < j \leq n} (1 - t_{ij})$. This assessment provides a better estimation of the difficulty of the problem and the runtime than the density of the constraint graph or the number of constraints.

In our case, we are not interested in the expectation of the whole problem, but only in ones of the produced clusters. So, we consider the solutions of these subproblems, what explains the term of partial solutions.

5.2 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 (no)goods does not enforce another order. Indeed this early use of (no)goods improves a lot the method, by detecting earlier inconsistencies. 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 (no)good. For a good, we do not explore the subtree rooted on this son cluster since its consistency is known. For a nogood, it is known that the current assignment leads to a failure so we backtrack. In case the assignment is neither a good nor a nogood we try to extend it.

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

- $random$: the root cluster is chosen randomly.
- $minexp$: this heuristic is based on the expected number of partial solutions of clusters and on their size. Our experiments have shown that it is always better to choose as root cluster one with a large size, so the heuristic choose 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.

- $size$: 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.

- $random_s$: we compute the order on son clusters randomly.
- $minexp_s$: this heuristic is similar to $minexp$ and orders the son clusters according to the increasing value of their ratio.
- $minexp_{sd}$: we order the son clusters according to the increasing value of the expected number of solutions of the subproblem rooted on the cluster.
- $maxexp_{sd}$: we order the son clusters according to the decreasing value of the expected number of solutions of the subproblem rooted on the cluster.
- $minsize_{sd}$: the son clusters are ordered according to the increasing value of the size of the subproblem rooted on the cluster.
- $maxsize_{sd}$: the son clusters are ordered according to the decreasing value of the size of the subproblem rooted on the cluster.
- $minsep_s$: 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 to begin 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.

- $random_{sdyn}$: we choose randomly the next cluster visited.
- $minexp_{sdyn}$: 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 sons clusters.

5.3 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.

- $random_v$: we compute the order variables randomly.
- 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.

- $random_{vdyn}$: we choose randomly the next variable to assign.
- 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.

5.4 Heuristics for grouping variables in the *Class 3 k-extended*

Grouping variables allows more freedom for dynamic variable ordering heuristics which 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 3 k-extended*.

- 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 is not in the cluster parent. We merge a cluster with its parent if its number of proper variables is under the parameter.
- exp: this heuristic is based on the expected number of solutions. Two clusters are merged if the expected number of solutions of the grouped cluster is less than the expected number of solutions of the two clusters. Our goal is to compute cluster with many variables and a few solutions.

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 close to ideal structure. In practice, we assess here the proposed strategies on random partial structured CSPs in order to point up the best ones w.r.t. CSP solving. For building a random partial structured instance of a class (n, d, w, t, s, ns, p), the first step consists in producing randomly a structured CSP according to the model described in [7]. This structured instance consists of n variables having d values in their domain. Its constraint graph is a clique tree with ns cliques whose size is at most w and whose separator size does not exceed s. Each constraint forbids t tuples. Then, the second step removes randomly $p\%$ edges from the structured instance. The experimentations are performed on a Linux-based PC with a Pentium IV 3.2GHz and 1GB of memory. For each considered class, the presented results are the average on 50 instances. We limit the runtime to 30 minutes. Above, the solver is stopped and the involved instance is considered as unsolved. In the following tables, the symbol $>$ denotes that at least one instance cannot be solved within 30 minutes and so the mean runtime is greater than the provided value. The letter M means that at least one instance cannot be solved because it requires more than 1GB of memory.

In [5], 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 [12] obtains the best results, we use it to compute tree-decompositions in this study. Regarding the provided results, by

CSP (n, d, w, t, s, ns, p)	w^+	s	Class 1		Class 2		Class 3		
			$size$ $minsep_s$	$minexp$ $minexp_s$	$size$ $minsep_s$	$minexp$ $minexp_s$	$minexp$ $minexp_{sdyn}$	nv nv_{sdyn}	$size$ nv_{sdyn}
$(a)(150, 25, 15, 215, 5, 15, 10)$	13.00	12.22	9.31	28.12	3.41	2.52	2.45	6.85	5.34
$(b)(150, 25, 15, 237, 5, 15, 20)$	12.54	11.90	>45.99	5.61	>41.10	2.69	2.32	>40.07	>41.47
$(c)(150, 25, 15, 257, 5, 15, 30)$	12.16	11.40	13.36	27.82	3.38	5.06	4.97	5.67	3.55
$(d)(150, 25, 15, 285, 5, 15, 40)$	11.52	10.64	3.04	>44.77	1.12	>36.87	>37.27	0.95	1.16
$(e)(250, 20, 20, 107, 5, 20, 10)$	17.82	16.92	57.22	>129.70	16.03	>56.77	>56.44	>99.67	15.26
$(f)(250, 20, 20, 117, 5, 20, 20)$	17.24	16.56	55.87	78.44	23.25	14.03	13.04	>77.14	24.00
$(g)(250, 20, 20, 129, 5, 20, 30)$	16.80	15.80	>123.28	>93.28	92.52	64.87	>78.47	>81.60	>107.10
$(h)(250, 20, 20, 146, 5, 20, 40)$	15.92	15.24	44.60	30.17	26.24	3.91	4.51	10.61	17.99
$(i)(250, 25, 15, 211, 5, 25, 10)$	13.04	12.34	28.83	>74.75	15.16	43.41	44.66	>53.53	17.89
$(j)(250, 25, 15, 230, 5, 25, 20)$	12.86	11.98	20.09	>70.47	8.51	>43.12	>50.84	10.93	19.17
$(k)(250, 25, 15, 253, 5, 25, 30)$	12.38	11.82	>47.36	16.68	7.01	10.94	5.06	6.01	6.91
$(l)(250, 25, 15, 280, 5, 25, 40)$	11.80	11.16	7.84	33.57	3.82	16.88	18.13	>52.97	5.03
$(m)(250, 20, 20, 99, 10, 25, 10)$	17.92	17.02	M	M	M	M	M	M	M
$(n)(500, 20, 15, 123, 5, 50, 10)$	13.04	12.58	12.60	13.63	7.01	8.08	7.31	8.32	7.54
$(o)(500, 20, 15, 136, 5, 50, 20)$	12.94	12.10	47.16	19.22	25.54	23.49	27.01	7.26	15.11

Table 1. Parameters w^+ and s of the tree-decomposition and runtime (in s) on random partial structured CSPs with mdd for class 1 and mdd_{dyn} for classes 2 and 3.

CSP	w^+	s	$size$ $minsep_s$	$minexp$ $minexp_s$	$minexp$ $minexp_{sdyn}$	nv nv_{sdyn}	$size$ nv_{sdyn}
(a)	14.04	4.98	2.75	2.17	2.08	4.65	2.65
(b)	14.04	5.00	2.58	1.76	1.63	2.47	2.97
(c)	14.86	5.00	1.41	1.05	1.13	1.23	1.30
(d)	15.48	5.00	1.67	0.39	0.63	0.88	1.75
(e)	19.00	4.98	10.66	>52.14	>51.67	>50.96	10.92
(f)	19.00	5.00	10.05	8.81	8.39	45.18	10.34
(g)	19.82	5.00	33.93	4.61	4.41	41.92	34.20
(h)	20.44	5.00	11.38	3.17	3.17	7.58	10.63
(i)	14.00	5.00	5.86	7.71	6.65	8.86	6.44
(j)	14.40	5.00	4.19	3.94	3.36	4.99	6.81
(k)	15.02	5.00	2.80	3.71	3.52	4.49	3.06
(l)	17.90	5.00	4.03	1.40	1.26	14.78	3.55
(m)	57.28	4.88	66.94	63.15	62.99	74.32	66.33
(n)	14.02	5.00	5.48	4.50	4.41	5.02	5.86
(o)	14.44	5.00	4.86	4.92	3.94	5.54	5.24

Table 2. Parameters w^+ and s of the tree-decomposition and runtime (in s) on random partial structured CSPs for several orders of class 3-k extended based on mdd_d and the sep merge heuristic (the separator size is at most 5).

lack of place, we only report the heuristics giving the more interesting results for each class of orders. We do not provide the results obtained by classical enumerative algorithms like FC or MAC since these algorithms are often unable to solve several instances of each instance class within 30 minutes.

Table 1 shows the runtime of BTD with several heuristics of Classes 1, 2 and 3. Also it presents the width of the computed tree-decompositions and the maximum size of the separators. Clearly, we observe that Class 1 orders obtain poor results. This behaviour is not surprising since static variable orders are well known to be inefficient compared to dynamic ones. A dynamic strategy allows to make good choices by taking in account the modifications of the problem. Thus these choices are more justified than in a static case. That explains the good results of Classes 2 and 3 orders. The results show as well the crucial importance of the root cluster choice. For Classes 2 and 3, the symbol $>$ comes from a bad choice of root cluster for an unique problem causing a very long runtime. That induces the increasing of the mean runtime while for the other 49 instances the results are similar to the attempts. We note that the unsolved instances are not the same for $size$ and $minexp$ heuristics. The memory problems marked by M can be solved by using a Class 3 k-extended order with the sep heuristic for grouping variables. Table 2 gives the runtime of BTD for this class with a maximum separator size bounded by 5. The heuristics improve very significantly their results obtained for the Classes 2 and 3. The impact of the dynamicity is obvious. $minexp$ and nv heuristics solve all the instances except one due to a bad root cluster choice, $size$ solve all the instances. Except this unsolved instance, $minexp$ obtains very promising results. The son cluster ordering has a limited effect because the instances considered 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 best results are obtained by $minexp + minexp_{sdyn}$, but $size + minsep_s$ obtains often similar results and succeed in solving all instances in the *Class 3 k-extended*. The expected number of solution calculus supposed the problem constraints are independent, what is the case for the problems computed here. Thus, $size + minsep$ may outperform $minexp + minexp_{sdyn}$ on real world problems which have dependent constraints. In Table 3, we present the best values of the parameters for sep and pv heuristics. Generally, the best value is between 4 and 6 for sep, between 2 and 6 for pv. A relevant choice of a value for these parameters may lead to a significant improvement w.r.t. CSP solving. Yet, we are try-

CSP	sep			pv	
	Time	k	s_{max}	Time	k
(a)	2.07	2	6	2.06	4
(b)	1.54	37	4	1.83	6
(c)	1.13	11	5	1.02	7
(d)	0.36	35	4	0.43	5
(e)	12.71	43	4	10.29	10
(f)	7.61	44	4	7.30	2
(g)	4.40	8	6	2.85	3
(h)	3.17	18	5	4.84	4
(i)	6.65	1	5	5.14	5
(j)	3.36	6	5	3.24	2
(k)	3.25	3	6	3.95	3
(l)	1.26	20	5	5.45	3
(m)	35.83	30	9	44.92	2
(n)	4.37	2	6	4.43	3
(o)	3.41	63	4	3.64	5

Table 3. Best runtime (in s) and corresponding value of k for orders of class 3-k extended based on mdd_{dyn}+$minexp$+$minexp_{sdyn}$ and respectively the sep and pv merge heuristics. For sep, the maximum allowed separator size s_{max} ranges from 1 to 10 while, for pv, clusters having k proper variables or less are merged with k between 1 and 10. This table presents the results obtained for the best value (w.r.t. runtime) of s_{max} or k.

ing to find out criteria that would allow us to compute such relevant values and so the question is still open. Hence, we confirm empirically the intuition that dynamic strategies are more efficient since clever choices are made. When we compare Tables 1 and 2, we see the relevance of extending the dynamic order. Merging clusters with k less than 5 decrease the maximal size of separator and allow a more dynamic ordering of variables. That leads to an important reduction of the runtime. These experiments highlight the importance of dynamic orders and make us conclude that the Class 3 k-extended gives the best variable orders w.r.t CSP solving with a good value of k. Of course, this behaviour has been observed on random instances. The next step of our study will consist in assessing the proposed heuristics on other kinds of benchmarks (for instance, ones of the CP'2005 solver competition[2]).

7 Discussion and Conclusion

In this article, we have studied the CSP 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 variables orders allows us to define more dynamic heuristics without the loss of time complexity bound. So, we have defined classes of variables orders which allow a more and more dynamic ordering of variables and preserve the theoretical time complexity bound. This bound had been extended to enforce the dynamic aspect of orders that has an important impact on the efficiency of enumerative methods. Even though this new bound is 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 on importance for solving hard instances with suitable structural properties. For example, the instances we have used have a suitable structure and are seldom solved by backtracking methods like FC or MAC.

We have compared the classes of variable orders with relevant heuristics w.r.t. CSP solving. This comparison points up the importance of a dynamic variable ordering. The best results are obtained by *Class 3 k-extended* orders because they give more freedom to the variable ordering heuristic. We exploit the notion of expected number of partial solutions in order to guide the traversal of the cluster tree during the solving. This expectation of the number of solutions of a subproblem gives a more relevant criterion for exploiting the first-fail principle. The heuristics based on this criterion have led to significant improvements of BTD w.r.t CSP solving. Even though the other heuristics presented ($size + minsep$) are less efficient, often they obtain similar results. They are also more general what induces a stable behaviour. Then, for *Class 3 k-extended*, we aim to find out criteria that would permit to compute the best value of k by exploiting the problem features.

This study will be pursued on the Valued CSP problem [10] which is well known to be more difficult than the CSP problem. Thus, good heuristics would significantly improve the method performances. Yet, this work should not be easy since some of the heuristics proposed here are difficult to extend to VCSP. Like for the CSP problem, we must find out criteria that would permit to compute the best value of k for a VCSP by exploiting the problem features. We should also define other variable ordering classes that are more dynamic than the *Class 3* and that guarantee better time complexity bounds than the *Class 4* does. They should increase the freedom in the computing of more efficient heuristics.

Finally, for both CSP and VCSP problems, the behaviour of the proposed heuristics should be assessed on other kinds of benchmarks than random instances (e.g. ones of the CP'2005 solver competition).

[2] This competition held during the Second International Workshop on Constraint Propagation and Implementation of CP'2005.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] R. Dechter and Y. El Fattah, 'Topological Parameters for Time-Space Tradeoff', *Artificial Intelligence*, **125**, 93–118, (2001).
[2] R. Dechter and J. Pearl, 'Tree-Clustering for Constraint Networks', *Artificial Intelligence*, **38**, 353–366, (1989).
[3] G. Gottlob, M. Hutle, and F. Wotawa, 'Combining hypertree, bicomp and hinge decomposition', in *Proc. of ECAI*, pp. 161–165, (2002).
[4] G. Gottlob, N. Leone, and F. Scarcello, 'A Comparison of Structural CSP Decomposition Methods', *Artificial Intelligence*, **124**, 343–282, (2000).
[5] P. Jégou, S. N. Ndiaye, and C. Terrioux, 'Computing and exploiting tree-decompositions for solving constraint networks', in *Proc. of CP*, pp. 777–781, (2005).
[6] P. Jégou, S. N. Ndiaye, and C. Terrioux, 'Heuristiques pour la recherche énumérative bornée : Vers une libération de l'ordre', Technical Report LSIS.RR.2006.004, Laboratoire des Sciences de l'Information et des Systèmes (LSIS), (2006). In french.
[7] P. Jégou and C. Terrioux, 'Hybrid backtracking bounded by tree-decomposition of constraint networks', *Artificial Intelligence*, **146**, 43–75, (2003).
[8] P. Jégou and C. Terrioux, 'Decomposition and good recording for solving Max-CSPs', in *Proc. of ECAI*, pp. 196–200, (2004).
[9] N. Robertson and P.D. Seymour, 'Graph minors II: Algorithmic aspects of tree-width', *Algorithms*, **7**, 309–322, (1986).
[10] T. Schiex, H. Fargier, and G. Verfaillie, 'Valued Constraint Satisfaction Problems: hard and easy problems', in *Proc. of IJCAI*, pp. 631–637, (1995).
[11] B. Smith, 'The Phase Transition and the Mushy Region in Constraint Satisfaction Problems', in *Proc. of ECAI*, pp. 100–104, (1994).
[12] 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).