Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs

Philippe Jégou and **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 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 $O(exp(w+k+1))$, and finally to $O(exp(2(w+k+1)-s^-))$, with w the "tree-width" of a CSP and s^- the minimum size of its separators. 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 $x_1, x_2, \dots x_n$, 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 $O(n.d^{w+1})$ (denoted $O(exp(w+1))$). Different methods have been proposed to reach this bound like *Tree-Clustering* [Dechter and Pearl, 1989] (see [Gottlob *et al.*, 2000] for more details). 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. Except [Gottlob *et al.*, 2002; Jégou and Terrioux, 2003], no practical results have been provided. So, we study these approaches by concentrating us on the BTD method [Jégou and Terrioux, 2003] which seems to be the most effective method proposed until now within the framework of these structural methods.

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 [Jégou *et al.*, 2005]. 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. As presented in [Jégou and Terrioux, 2003], 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: $O(exp(w+k+1))$ and $O(exp(2(w+k+1)-s^-))$, where k is a constant to parameterize and s^- the minimum size of separators. 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 CSP 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. The next section provides basic notions about CSPs 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 is devoted to experimental results to assess the practical interest of our propositions. Finally, in section 6, we summarize this work 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 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 [Dechter and Pearl, 1989] 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 [Robertson and Seymour, 1986]. 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 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 $\{x, y\} \subseteq E_i$, 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.

Assume that we have a tree-decomposition of minimal width (w), the time complexity of Tree-Clustering is $O(exp(w + 1))$ 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 [Dechter and Fattah, 2001]. 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 [Jégou and Terrioux, 2003]. This method is called BTD (for Backtracking with Tree-Decomposition) and seems to provide among the best empirical results 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. 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 assignment 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, namely a separator (i.e. an intersection between two clusters), 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 complexity bounds, the variable ordering exploited in BTD is related to the cluster ordering. Formally, let us consider a tree-decomposition (E, T) of the CSP with $T = (I, F)$ a tree and assume that the elements of $E = \{E_i : i \in I\}$ are indexed w.r.t. a *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 the E_i 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 and nogoods allows not to explore twice subproblems if their consistency (inconsistency) with the current assignment is known. If we use a good to avoid visiting again a subtree, we known that the variables in it can be assigned consistently with the current assignment. So BTD does not assign them effectively, but they are considered done. For consistent problems, an additional work must be performed to assign these variables if we want to provide a solution [Jégou and Terrioux, 2004]. They are named *consistently assignable variables* thanks to a good. Thus the variables in E_j are assigned if the variables in E_i are either already assigned or consistently 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 [Jégou and Terrioux, 2003; 2004], the presented results were obtained without heuristics for the choice of the clusters and thus the choice of the variables. Only a traditional dynamic order was used inside the clusters. 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.

Let (X, D, C) be a CSP and (E, T) a tree-decomposition of the graph (X, C), we exploit an order σ_i on the subproblems $\mathcal{P}_{i_1}, \dots, \mathcal{P}_{i_k}$ rooted on the sons E_{i_j} of E_i and an order γ_i on the variables in E_i. We define recursively the following classes of orders. In the three next classes, we choose the first cluster to assign (the root): E_1 among all the clusters and we consider $\mathcal{P}_1$ the subproblem rooted on E_1 (i.e. the whole problem).

Definition 1 *We begin the search in E_1 and we try recursively to extend the current assignment on the subproblem rooted on E_i by assigning first the variables in E_i according to γ_i and then on $\mathcal{P}_{i_1}, \dots, \mathcal{P}_{i_k}$ according to σ_i.*

- ***Class 1.** σ_i **and** γ_i **are static**. We compute σ_i and γ_i statically (before starting the search).*
- ***Class 2.** σ_i **is static and** γ_i **is dynamic**. We compute statically σ_i, while γ_i is computed during the search.*
- ***Class 3.** σ_i **and** γ_i **are dynamic**. Both, σ_i and γ_i are computed during the search. σ_i is computed w.r.t. the current assignment as soon as all the variables in E_i are assigned.*
- ***Class ++. Enumerative dynamic order**. The variable ordering is completely dynamic. So, the assignment order is not necessarily a compatible enumeration order. There is no restriction due to the cluster tree.*

The defined classes form a hierarchy since we have: *Class 1* $\subset$ *Class 2* $\subset$ *Class 3* $\subset$ *Class ++*.

Property of the *Class 3*. Let Y be an assignment, $x \in E_j - (E_i \cap E_j)$ with E_i the parent of E_j: $x \in Y$ iff: i)$\forall v \in E_i$, $v \in Y$ ii) Let $E_{i_p} = E_j$, $\forall \mathcal{P}_{i_u}$ s.t. $\sigma_i(\mathcal{P}_{i_u}) \leq \sigma_i(\mathcal{P}_{i_p})$, $\forall v \in \mathcal{P}_{i_u}$, $v \in Y$ iii) $\forall v \in E_j$ s.t. $\gamma_j(v) \leq \gamma_j(x)$, $v \in Y$.

In [Jégou and Terrioux, 2003], the experiments use *Class 2* orders. Formally, only the orders of the *Class 1* are compatible. Nevertheless, for an order o_3 in the *Class 3* and a given assignment $\mathcal{A}$, one can find an order o_1 in the *Class 1* that instantiates the variables in $\mathcal{A}$ in the same way and the same order o_3 does. 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 ++* gives a complete freedom. Yet, it does not guarantee the time complexity bound because sometimes it is impossible to record nogoods. Indeed, let the cluster E_j be a son of the cluster E_i, we suppose that the enumeration 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 solving of the subtree rooted on E_j leads to a failure, it is impossible to record a nogood on $E_i \cap E_j$ (if it is consistently assigned) because we do not try the other values of x to prove that the assignment on $E_i \cap E_j$ cannot be consistently extended on this subtree. 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. Thus the time complexity bound is not guaranteed anymore. Meanwhile, the *Class 3* orders guarantee this bound.

Theorem 1 *Let the enumeration 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 all the variables in E_i are assigned and those in $E_j - (E_i \cap E_j)$ are not. Since the order belongs to 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 at the latest when the variables in E_i are assigned (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 an assignment on E_j compatible with $\mathcal{A}$. 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 among the sons of the current cluster. And in each cluster, the variable ordering is totally free. In section 4, 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 variable 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 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:*

- *$E_1 \subseteq E'_1$, E'_1 the root cluster of (E', T')*
- *$\forall E'_i \in E'$, $E'_i \subseteq E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$ and $E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_{R-1}} \subset E'_i$, with $E_{i_1} \ldots E_{i_R}$ a path in T*
- *$|E'_i| \leq w + k + 1$, where $w = max_{E_i \in E}|E_i| - 1$*

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

Definition 3 *Let (X, D, C) be a CSP, (E, T) a tree-decomposition of the graph (X,C) and k a positive integer. A variable order is in the **Class 4**, if this order is in the Class 3 for one directed k-covering tree-decomposition of (E, T).*

We derive a natural theorem:

Theorem 2 *Let the enumeration order be in the Class 4, the time complexity of BTD is $O(exp(w + k + 1))$.*

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 at most $w + k + 1$. □

A second extension is possible in exploiting during the search, a dynamic computing of the tree-decomposition (we can use several directed k-covering tree-decompositions during the search). Then, the time complexity bound changes because sometimes it would be impossible to record nogoods.

Definition 4 *Let (X, D, C) be a CSP, (E, T) a tree-decomposition of the graph (X,C) and k a positive integer. A variable order o_5 is in the **Class 5**, if for a given assignment $\mathcal{A}$, one can find one directed k-covering tree-decomposition (E', T') of (E, T) such that $\forall E'_i \in E'$, $E'_i = E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$, with $E_{i_1} \ldots E_{i_R}$ a path in T and find an order o_3 on (E', T'), in the Class 3 that instantiates the variables in $\mathcal{A}$ in the same way and the same order o_5 does.*

This definition enforces to use directed k-covering tree-decompositions (E', T') of (E, T) that verify the additional condition: $\forall E'_i \in E'$, $E'_i = E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$. Hence, a separator in (E', T') is also a separator in (E, T). We denote by s^- the minimum size of separators in (E, T).

Theorem 3 *Let the enumeration order be in the Class 5, the time complexity of BTD is $O(exp(2(w + k + 1) - s^-))$.*

Proof Let (X, D, C) be a CSP, (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 $2(w+k+1)-s^-$ variables on a path of the tree T is computed only once. Let $\mathcal{A}$ be an assignment containing V. The order in which the variables of $\mathcal{A}$ were assigned is in the *Class 3* for a directed k-covering tree-decomposition (E', T') of (E, T) that verifies $\forall E'_i \in E'$, $E'_i = E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$, with $E_{i_1} \ldots E_{i_R}$ a path in T. The size of the clusters in (E', T') is bound by $w+k+1$, so the set V is covered by at least two clusters since s^- is the minimum size of the separators. Let $E'_{i_1} \ldots E'_{i_q}$ 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 (no)goods on the separators of these clusters. If E'_{i_1} is the root cluster of (E', T'), then V contains E_1. Thus $\mathcal{A}$ will not be computed again because it contains the first variables in the search. We suppose that E'_{i_1} is not the root cluster of (E', T'). Since $q \geq 2$, we record a (no)good on the separator of E'_{i_1} and its parent and at least an other on the separator of E'_{i_1} and E'_{i_2}. Let $\mathcal{B}$ be a new assignment that we try to extend on V with the same values in $\mathcal{A}$. One of the (no)goods will be computed first. Thus before all the variables in V are assigned, the search is stopped thanks to this (no)good. So $\mathcal{A}$ is not computed again. 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 Experimental study

Applying a structural method on an instance generally assumes that this instance presents some particular topological features. So, our study is first performed on instances having a structure which can be exploited by structural methods. 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, n_c, p), the first step consists in producing randomly a structured CSP according to the model described in [Jégou and Terrioux, 2003]. 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. Then, the second step removes randomly p% edges from the structured instance. Secondly, we experiment the proposed heuristics on benchmark instances of the CP'2005 solver competition[1]. 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 50. We limit the runtime to 30 minutes for random instances and to 10 minutes for CP'2005 benchmark instances. Above, the solver is stopped and the involved instance is considered as unsolved (what is denoted by the letter T in tables). 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 [Jégou *et al.*, 2005], 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 [Tarjan and Yannakakis, 1984] obtains the best results and is very easy to implement, we use it to compute tree-decompositions in this study. We do not provide the results obtained by classical enumerative algorithms like FC or MAC since they are often unable to solve several instances of each class within 30 minutes.

Here, for lack of place, we only present the more efficient heuristics:

- $minexp(A)$: this heuristic is based on the expected number of partial solutions of clusters [Smith, 1994] and on their size. It chooses as root cluster one which minimizes the ratio between the expected number of solutions and the size of the cluster.
- $size(B)$: we choose the cluster of maximum size as root cluster
- $minexp_s(C)$: this heuristic is similar to $minexp$ and orders the son clusters according to the increasing value of their ratio.
- $minsep_s(D)$: we order the son clusters according to the increasing size of their separator with their parent.
- $nv(E)$: we choose a dynamic variable ordering heuristic and we visit first the son cluster where appears the next variable in the heuristic order among the variables of the unvisited son clusters.
- $minexp_{sdyn}(F)$: 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.

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

CSP	w^+	s	Class 1		Class 2		Class 3		Class 4			
			B	A	B	A	A	B	B	A	A	B
(n, d, w, t, s, n_c, p)			D	C	D	C	F	G	D	C	F	G
$(150, 25, 15, 215, 5, 15, 10)$	13.00	12.22	9.31	28.12	3.41	2.52	2.45	5.34	2.75	2.17	2.08	2.65
$(150, 25, 15, 237, 5, 15, 20)$	12.54	11.90	9.99	5.27	5.10	2.47	1.99	5.47	2.58	1.76	1.63	2.97
$(150, 25, 15, 257, 5, 15, 30)$	12.16	11.40	13.36	27.82	3.38	5.06	4.97	3.55	1.41	1.05	1.13	1.30
$(150, 25, 15, 285, 5, 15, 40)$	11.52	10.64	3.07	8.77	1.13	0.87	1.27	1.17	1.67	0.39	0.63	1.75
$(250, 20, 20, 107, 5, 20, 10)$	17.82	16.92	54.59	57.75	15.92	12.39	12.14	14.93	10.18	7.75	7.34	10.26
$(250, 20, 20, 117, 5, 20, 20)$	17.24	16.56	55.39	79.80	23.38	14.26	13.25	24.14	10.05	8.81	8.39	10.34
$(250, 20, 20, 129, 5, 20, 30)$	16.80	15.80	26.21	21.14	7.23	5.51	6.19	7.84	33.93	4.61	4.41	34.20
$(250, 20, 20, 146, 5, 20, 40)$	15.92	15.24	44.60	30.17	26.24	3.91	4.51	17.99	11.38	3.17	3.17	10.63
$(250, 25, 15, 211, 5, 25, 10)$	13.04	12.34	28.86	38.75	15.33	11.67	13.37	18.12	5.86	7.71	6.65	6.44
$(250, 25, 15, 230, 5, 25, 20)$	12.86	11.98	20.21	34.47	8.60	7.12	14.84	19.47	4.19	3.94	3.36	6.81
$(250, 25, 15, 253, 5, 25, 30)$	12.38	11.82	11.36	16.91	5.18	11.13	5.14	5.26	2.80	3.71	3.52	3.06
$(250, 25, 15, 280, 5, 25, 40)$	11.80	11.16	7.56	32.74	3.67	16.32	17.49	4.91	4.03	1.40	1.26	3.55
$(250, 20, 20, 99, 10, 25, 10)$	17.92	17.02	M	M	M	M	M	M	66.94	63.15	62.99	66.33
$(500, 20, 15, 123, 5, 50, 10)$	13.04	12.58	12.60	13.63	7.01	8.08	7.31	7.54	5.48	4.50	4.41	5.86
$(500, 20, 15, 136, 5, 50, 20)$	12.94	12.10	47.16	19.22	25.54	23.49	27.01	15.11	4.86	4.92	3.94	5.24

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, 3 and 4.

CSP Instance	n	e	w^+	s	Class 1		Class 2		Class 3		
					B	A	B	A	A	E	B
					D	D	C	C	F	G	G
qa_5	26	309	25	9	32.01	132.82	2.67	77.82	84.19	80.86	2.62
qcp-10-64-45-QWH-10	100	900	91	82	T	M	69.83	T	T	70.10	69.82
qcp-15-120-278-QWH-15	225	3,150	211	197	T	T	120.69	10.47	10.92	122.26	122.47
qwh-15-106-392-QWH-15	225	3,150	211	197	T	M	0.89	169.49	181.18	0.86	0.89
qwh-15-106-974-QWH-15	225	3,150	211	197	T	94.58	2.55	75.77	79.46	2.67	2.54

Table 2: Parameters w^+ and s of the tree-decomposition and runtime (in s) on some instances from the CP'2005 solver competition with mdd for class 1 and mdd_{dyn} for classes 2 and 3.

- $nv_{sdyn}(G)$: it is similar to nv. We visit first the son cluster where appears the next variable in the heuristic order among the variables of the unvisited son clusters.

Inside a cluster, the heuristic used for choosing the next variable is min domain/degree (static version mdd_s for class 1 and dynamic mdd_{dyn} for the other classes).

Table 1 shows the runtime of BTD with several heuristics of Classes 1, 2, 3 and 4. For Class 5, we cannot get good results and then, the results are not presented. Also it presents the width of the computed tree-decomposition 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 during search. 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 since each heuristic of the Classes 2 and 3 fails to solve an average of 4 instances over all instances of all classes because of a bad choice of root cluster. 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 4* order with the sep heuristic for grouping variables (we merge cluster whose intersection is greater than a value s_{max}). Table 1 gives the runtime of BTD for this class with $s_{max} = 5$.

When we analyze the value of the parameter k, we observe that its value is generally limited (between 1 to 6). Nevertheless, for the CSPs $(250, 20, 20, 99, 10, 25, 10)$, the value of k is near 40, but this high value allows to solve these instances.

The heuristics for the Classes 2 and 3 improve very significantly the results obtained. 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 the 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 4*. The calculus of the expected number of solution assumes that the problem constraints are independent, what is the case for the problems considered here. Thus, $size + minsep$ may outperform $minexp + minexp_{sdyn}$ on real-world problems which have dependent constraints.

When we observe the results in table 1, we see the relevance of extending the dynamic order. Merging clusters with

k less than 6 decreases the maximal size of separators and allows 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 4 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 consists in assessing the proposed heuristics on benchmark instances of the CP'2005 solver competition. At the beginning of the section, we reminded that structural methods like BTD assume the CSP has interesting topological features. This is not the case for a great part of instances of this competition. Since the space complexity of BTD is in $O(n.s.d^s)$, if the size of the cluster separators of the computed tree-decomposition of CSP is too large, the method needs more than 1GB of memory. Many instances have tree-decompositions with very large separators and clusters. Reducing the required space memory leads to group all the variables in one cluster and so to perform like the FC algorithm. In table 2, which provides results on these instances, we do not present the results for instances with very bad topological features for which BTD has a behavior close to FC one.

On these instances, the $size$ heuristic outperforms $minexp$ except on qcp instance. To resume, we can say that the dynamicity improves significantly the method and the expected number of solutions provides an important improvement on random CSPs, while the $size + minsep$ heuristic outperforms the others on real-world instances.

6 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 variable orders allows us to define more dynamic heuristics without loosing the time complexity bound. 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 on importance for solving hard instances with suitable structural properties. For example, the structured instances used here are seldom solved by enumerative 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. Indeed the best results are obtained by *Class 4* orders because they give more freedom to the variable ordering heuristic while their time complexity is $O(exp(w+k+1))$ where k is a constant to parameterize. Note that for the most dynamic class (the Class 5), we get a time complexity in $O(exp(2(w + k + 1) - s^-))$. It seems that this bound should be too large to expect a significant practical improvement.

Concerning heuristics, we exploit the notion of expected number of partial solutions in order to guide the traversal of the cluster tree during the 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 stabler behaviour. So, on real-world problems with dependent constraints, they may outperform the expected number of solutions based heuristics. 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.

This study will be pursued on the Valued CSPs [Schiex *et al.*, 1995] which are well known to be more difficult.

Acknowledgments

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

References

[Dechter and Fattah, 2001] R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.

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

[Gottlob *et al.*, 2000] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.

[Gottlob *et al.*, 2002] G. Gottlob, M. Hutle, and F. Wotawa. Combining hypertree, bicomp and hinge decomposition. In *Proceedings of ECAI*, pages 161–165, 2002.

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

[Jégou and Terrioux, 2004] P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proceedings of ECAI*, pages 196–200, 2004.

[Jégou *et al.*, 2005] 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.

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

[Schiex *et al.*, 1995] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proceedings of IJCAI*, pages 631–637, 1995.

[Smith, 1994] B. Smith. The Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings of ECAI*, pages 100–104, 1994.

[Tarjan and Yannakakis, 1984] 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.