Dynamic Management of Heuristics for Solving Structured 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 the problem of solving efficiently structured CSPs. It is well known that (hyper)tree-decompositions offer the best approaches from a theoretical viewpoint, but from the practical viewpoint, these methods do not offer efficient algorithms. Therefore, we introduce here a framework founded on coverings of CSP by acyclic hypergraphs. We study their properties and relations, and evaluate theoretically their interest with respect to the solving of structured problems. This framework does not define a new decomposition, but makes easier a dynamic management of the CSP structure during the search, and so an exploitation of dynamic variables ordering heuristics in the solving method. Thus, we provide a new complexity result which outperforms significantly the previous one given in the literature about heuristics for solving a decomposed problem. Finally, we present experimental results to assess the practical interest of these notions.

1 Introduction

In the past, the interest for the exploitation of structural properties of a problem was attested in various domains: for checking satisfiability in SAT [1–3], for solving CSP [4], in Bayesian or probabilistic networks [5, 6], in relational databases [7, 8], for constraint optimization [9, 10]. Complexity results based on topological properties of the network structure have been proposed. Generally, these results rely on the properties of a tree-decomposition [11] or a hypertree-decomposition [12] of the network, which can be considered as an acyclic hypergraph (a hypertree) covering the network. If we consider tree-decomposition, the time complexity of the best structural methods is $O(exp(w+1))$, with w the width of the used tree-decomposition. If we consider hypertree-decomposition, the time complexity is then $O(exp(k))$, with k the width of the used hypertree-decomposition. It has been shown that hypertree-decomposition is better than tree-decomposition [12], while it is recently outperformed by a generalized hypertree-decomposition [13, 14]. Note that these theoretical complexities can really outperform the classical one which is $O(exp(n))$ $(k < w < n)$ with n the number of variables of the considered CSP.

However, while several methods and theoretical results have been proposed, the practical interests of such approaches have not been proved yet, except in some recent works around CSPs [15] or Valued CSPs [16, 17, 10]. This is due to the fact that the good complexity bounds are often reached to the detriment of the practical efficiency. Precisely, approaches based on hypertree-decompositions and their generalizations assume that relations associated to constraints are expressed by tables, what is unrealistic, in practice, for many real life problems. Moreover, to ensure complexity bounds, the decompositions perform joins of relations. For solving CSPs, such an approach is generally unrealistic due to the size of the generated relations. Indeed, solving structured CSPs sometimes requires to manage joins of relations defined on several tens of attributes (i.e. variables), which is clearly impossible in practice. On the other hand, the methods that have shown their feasibility and their practical interest are based on assignments of variables. These methods can exploit the practical efficiency of backtracking-based algorithms with filtering, while they ensure complexity bounds without problems related to practical space complexity, contrary to approaches based on management of relations as hypertree-decomposition. Yet, some methods exploiting the structure of the problem, based on heuristics guaranteeing no good complexity bound, have shown the interest of such an approach [3].

In this paper, we propose to make a trade-off between good theoretical complexity bounds and the peremptory necessity to exploit efficient heuristics as often as possible. From this viewpoint, this work can be considered as an extension of the works recently presented in [17] in the field of AND/OR Branch-and-Bound Search in Constraint Optimization, or in [18] in the field of tree-decomposition methods for CSPs. Our contribution is more precisely an extension of the approach presented in [18]. Furthermore, we propose a framework different from the previous one, better theoretical results and a practical validation of this approach. Actually, we prefer here the more general and useful concept of covering acyclic hypergraph to the concept of tree-decomposition. Note that we do not aim here to define a new decomposition method for solving CSPs, nor to propose new bounds for complexity. We introduce a framework that makes possible to implement decomposition methods which can be more efficient than previously, because they can exploit efficient heuristics. Note that generally, decomposition methods define a decomposition of the constraint network, and then solve the associated CSP exploiting statically this structure. Here, our goal is to exploit dynamically sets of structures induced by the considered decomposition.

Given a hypergraph $H = (X, C)$ related to the graphical representation of the considered problem, we consider a covering of this hypergraph by an acyclic hypergraph $H_A = (X, E)$: the set of vertices is the same while for each edge $C_i \in C$, there is an edge $E_i \in E$ covering C_i ($C_i \subset E_i$). Now, given H_A, we can define various classes of acyclic hypergraphs which cover H_A. These classes are defined on the basis of criteria related to the nature of coverings and the relations existing with the solving methods: bounding the value of parameters such as tree-width, preservation of the separators in H_A, merging of neighboring hyperedges or ability to implement effective heuristics, in particular dynamic ones. First, these coverings are studied theoretically in order to determine their characteristics and properties. After, we show that they permit to preserve already known complexity results and also to improve some of them. Moreover, we indicate how they offer a framework for a dynamic management of the structure: during a search, we can take into account not only one acyclic hypergraph covering, but a set of coverings in order to manage heuristics dynamically. Thanks to this formal framework, we present a new algorithm (called BDH for "Backtracking on Dynamic covering by acyclic Hypergraphs") for which it is easy to extend heuristics. For example, for dynamic variable ordering, we can add dynamically a set of Δ variables for the choices. Moreover, we show how an implementation is made possible easily, allowing us to show the practical interest of this approach.

In the next section, we recall some results on structural decompositions. Section 3 introduces various classes of acyclic hypergraph coverings and show their relations. The fourth section describes how these classes can be exploited on the algorithmic level and gives the theoretical complexity they guarantee. Finally, we presents an experimental analysis before concluding.

2 Decompositions methods for solving CSPs efficiently

A *constraint satisfaction problem* (CSP) is defined by a tuple (X, D, C, R). X is a set of n variables which must be assigned in their respective finite domain from D, by satisfying a set C of constraints which are defined on a set of relations R. A solution is an assignment of every variable which satisfies all the constraints. Here a constraint $C_i \in C$ is defined by a subset of variables ($C_i \subset X$), while the associated relation R_i expresses a set of tuples of values defined on domains of variables belonging to C_i, all tuples in R_i satisfying the constraint C_i. The CSP structure can be represented by the hypergraph (X, C), called the *constraint hypergraph* (if all the constraints of a CSP are defined by pairs of variables, then we consider a *constraint graph*).

In this paper, we assume that the relations can be represented by tables, predicates, functions, or (in)equations. This remark is important in the field of decomposition methods. Indeed, several results assume that relations are represented by tables. From a theoretical viewpoint, this is possible since we consider finite domains, but from a practical viewpoint, this restriction can be unrealistic. Let us consider a constraint defined by the inequation $x_1 + x_2 + \dots x_c > c$, with domains $D_i = \{1, \dots 1000\}, i = 1, \dots, c$. The memory size of the table for representing the associated relation is then $c.(1000^c - 1)$, which is clearly unrealistic even for small values of c.

The basic concept which interests us here is the acyclicity of networks. Often, this concept is expressed by considering the tree-decomposition of constraint graphs, hypertree-decomposition of constraint hypergraphs, or more generally, coverings of variables and constraints by acyclic hypergraphs. We recall these different decompositions.

Tree-decomposition is based on graphs. Nevertheless, given a constraint hypergraph, we can exploit it by considering its primal graph. Let $H = (X, C)$ be a hypergraph, with X a finite set of vertices and $C = \{C_1, C_2, \dots C_m\}$ a set of edges (sometimes called hyperedges) which are nonempty subsets of X. Here we consider only reduced hypergraphs, that is hypergraphs such that for all edges C_i of H, C_i is not a proper subset of another edge of H. The primal graph of H is the graph $G = (X, A)$ where $A = \{\{x, y\} : \exists C_i \in C$ such that $\{x, y\} \subset C_i\}$.

Definition 1 *A tree-decomposition of a graph $G = (X, A)$ 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, s.t. 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 $\{x, y\} \in A$, *there exists* $i \in I$ *with* $\{x, y\} \subset E_i$, *and*
(iii) for all $i, j, k \in I$, *if* k *is in a path from* i *to* j *in* T, *then* $E_i \cap E_j \subset E_k$.

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

Assume that we have a tree-decomposition of width w for the constraint network. The best structural methods based on a tree-decomposition of a CSP have a time complexity in $O(exp(w + 1))$, while their space complexity can be reduced to $O(exp(s))$ where s is the size of the largest intersection $E_i \cap E_j$ (generally considered as minimal separators) between neighboring clusters of the tree-decomposition. Tree-Clustering (TC [4]) is based on this notion, but has never shown its efficiency for real life problems. This is due to the fact that TC runs in finding firstly all solutions of each subproblem induced by the variables in each E_i. An example of an efficient method exploiting tree-decomposition is BTD [15], which does not compute all solutions of subproblems. This method applies a backtracking driven by the tree-decomposition and preserves complexity bounds in $O(exp(w + 1))$. Note that this kind of methods can be considered as driven by the assignment of variables (or as "variables driven" methods). As a consequence, it does not need to represent relations in tables.

From a theoretical viewpoint, methods based on tree-decomposition are less interesting than those based on hypertree-decomposition and their generalizations [12].

Definition 2 *Given a hypergraph $H = (X, E)$, a hypertree for the hypergraph H is a triple (T, χ, λ) where $T = (N, F)$ is a rooted tree, and χ and λ are labelling functions which associate to each vertex $p \in N$ two sets $\chi(p) \subset X$ and $\lambda(p) \subset E$. If $T' = (N', F')$ is a subtree of T, we define $\chi(T') = \cup_{v \in N'} \chi(v)$. We denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.*

Definition 3 *Given a hypergraph $H = (X, E)$, a hypertree-decomposition of H is a hypertree $HD = (T, \chi, \lambda)$ for H which satisfies all the following conditions:*

(i) for each edge $E_i \in E, \exists p \in vertices(T)$ *such that* $E_i \subset \chi(p)$,
(ii) for each vertex $x \in X$, *the set* $\{p \in vertices(T) : x \in \chi(p)\}$ *induces a (connected) subtree of* T,

(iii) for each $p \in vertices(T), \chi(p) \subset \cup_{E_i \in \lambda(p)} E_i$,
(iv) for each $p \in vertices(T), \cup_{E_i \in \lambda(p)} E_i \cap \chi(T_p) \subset \chi(p)$.

An edge $E_i \in E$ is strongly covered in HD if there exists $p \in vertices(T)$ such that $E_i \subset \chi(p)$ and $E_i \in \lambda(p)$. A hypertree-decomposition HD is a complete decomposition of H if every edge of H is strongly covered in HD. The width k of a hypertree-decomposition $HD = (T, \chi, \lambda)$ is $max_{p \in vertices(T)} |\lambda(p)|$. The hypertree-width k^* of H is the minimum width over all its hypertree-decompositions.

Remark that acyclic hypergraphs are precisely those hypergraphs having a hypertree width one.

While tree-decomposition consists in grouping the vertices in clusters (i.e. variables in subproblems), hypertree-decomposition consists in grouping the constraints (and so the relations) in nodes of the hypertree. Given a hypertree-decomposition of a CSP, the method exploiting it consists in solving first each node of T, and then solving the acyclic induced problem. Thus its time complexity is $O(|\mathcal{P}|^{k+1} log(|\mathcal{P}|))$ where $|\mathcal{P}|$ denotes the size of the CSP $\mathcal{P}$ and k the width of the considered hypertree-decomposition of $H = (X, C)$. This complexity can be limited to $O(r^{k+1} log(r))$ where r is the maximum size for all tables representing relations in the considered CSP. This cost is related to the cost of computing each node of the hypertree, which is bounded by the cost of joins between k relations, which is r^k. Now, if we consider complexity space, the complexity is related to the size of the largest relation induced for each node of the hypertree, that is also $O(r^k)$. This kind of methods can be considered as "relations driven" approaches. Indeed, the time complexity bounds of these approaches outperform those guaranteed by methods as TC or BTD. Nevertheless, the relations associated to constraints must be represented by tables, or at least, the relations associated to nodes of the hypertree must be represented by tables. This restriction can make these approaches unusable in practice. Moreover, like TC, they must compute first all the solutions in each node of the hypertree by joining relations in it. Consequently, as for TC, these methods will require a too expensive amount of memory space in practice making them unusable again. Thus, they are generally unrealistic to solve real instances of CSPs, and then, at the present time, the proof of their practical efficiency has never been shown.

So, while from a theoretical viewpoint, it appears that (generalized-)hypertree-decompositions should be considered, we prefer consider here "variables driven" decompositions. In this paper, we will refer to the covering of constraint networks by acyclic hypergraphs. Different definitions of acyclicity have been proposed. Here, we consider the classical definition called $\alpha - acyclicity$ in [7].

Definition 4 *Let $H = (X, C)$ be a hypergraph. A covering by an acyclic hypergraph (CAH) of the hypergraph H is an acyclic hypergraph $H_A = (X, E)$ such that for each edge $C_i \in C$, there exists $E_j \in E$ such that $C_i \subset E_j$. The width α of a CAH (X, E) is equal to $max_{E_i \in E} |E_i|$. The CAH-width α^* of H is the minimal width over all the CAHs of H_A. Finally, $\mathcal{CAH}(H)$ is the set of the CAHs of H.*

The notion of covering by acyclic hypergraph (called hypertree embedding in [19]) is very close to the one of tree-decomposition. Particularly, it is easy to see that for a tree-decomposition (E,T) of the primal graph of $H=(X,C)$, the pair (X,E) is a CAH of the hypergraph H. Moreover, the CAH-width α^* of H is equal to the *tree-width* of G plus one. However, the concept of CAH is less restrictive. Indeed, for a given (hyper)graph, it can exist a single CAH whose width is α, while it can exist several tree-decompositions of width w such that $\alpha = w+1$. An example is given in figure 1, where for a constraint network, we have one CAH (with $\alpha = 4$) and two possible tree-decompositions (with $w=3$). Given a CSP with a CAH of width α, the time complexity of better structural methods for solving it is $O(exp(\alpha))$ while its space complexity can be reduced to $O(exp(s))$ where s is the size of the largest intersection $E_i \cap E_j$ in H_A.

he next section presents different classes of covering acyclic hypergraphs.

3 Coverings by classes of acyclic hypergraphs

In the sequel, given a hypergraph $H=(X,C)$ and one of its CAHs $H_A=(X,E)$, we study several classes of acyclic coverings of H_A. These coverings correspond to coverings of hyperedges (elements of E) by other hyperedges (larger but less numerous), which belong to a hypergraph defined on the same set of vertices and which is acyclic. In all the cases, these extensions are defined with respect to a particular CAH H_A, called *CAH of reference*. Here, we do not introduce a new decomposition of hypergraph. We study different ways for covering and so to solve a CSP by decomposition. Our objective is to formalize different classes of acyclic coverings, to manage dynamically, during search, acyclic coverings of the considered CSP. We hope by this mean, to manage dynamic heuristics to optimize backtrack search while preserving complexity results.

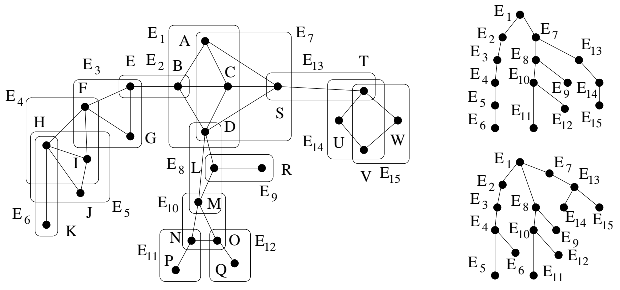

Fig. 1. A graph on 23 vertices under a covering by an acyclic hypergraph (of 15 edges) and 2 possible tree-decompositions.

Definition 5 *The set of coverings of a CAH $H_A=(X,E)$ of a hypergraph $H=(X,C)$ is defined by $\mathcal{CAH}_{H_A} = \{(X,E') \in \mathcal{CAH}(H) : \forall E_i \in E, \exists E'_j \in E' : E_i \subset E'_j\}$*

The following classes of coverings will be successive restrictions of this first class $\mathcal{CAH}_{H_A}$. But, let us define before that the notions of neighboring hyperedges in a hypergraph $H = (X, C)$.

Definition 6 *Let C_u and C_v be two hyperedges in H such that $C_u \cap C_v \neq \emptyset$. C_u and C_v are neighbours if $\nexists C_{i_1}, C_{i_2}, \ldots, C_{i_R}$ such that $R > 2, C_{i_1} = C_u, C_{i_R} = C_v$ and $C_u \cap C_v \subsetneq C_{i_j} \cap C_{i_{j+1}}$, with $j = 1, \ldots, R-1$.*
A path in H is a sequence of hyperedges $(C_{i_1}, \ldots C_{i_R})$ such that $\forall j, 1 \leq j < R, C_{i_j}$ and $C_{i_{j+1}}$ are neighbours. A cycle in H is a path $(C_{i_1}, C_{i_2}, \ldots C_{i_R})$ such that $R > 3$ and $C_{i_1} = C_{i_R}$. H is $\alpha-acyclic$ iff H contains no cycle.

The first restriction imposes that the edges E_i covered (even partially) by a same edge E'_j are connected in H_A, i.e. mutually accessible by paths. This class is called *set of connected-coverings of a CAH* $H_A = (X, E)$ and is denoted $\mathcal{CAH}_{H_A}[C^+]$. It is possible to restrict this class by restricting the nature of the set $\{E_{i_1}, E_{i_2}, \ldots E_{i_R}\}$. On the one hand, we can limit the considered set to paths (class of *path-coverings of a CAH* denoted $\mathcal{CAH}_{H_A}[P^+]$), and on the other hand by taking into account the maximum length of connection (class of *family-coverings of a CAH* denote $\mathcal{CAH}_{H_A}[F^+]$). We can also define a class (called *unique-coverings of a CAH* and denoted $\mathcal{CAH}_{H_A}[U^+]$) which imposes the covering of an edge E_i by a single edge of E'. Finally, it is possible to extend the class $\mathcal{CAH}_{H_A}$ in another direction (class of *close-coverings of a CAH* denoted $\mathcal{CAH}_{H_A}[B^+]$), ensuring neither connexity, nor unicity: we can cover edges with empty intersections but which have a common neighbor.

Definition 7 *Given a graph H and a CAH H_A of H:*

- $\mathcal{CAH}_{H_A}[C^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$ *with* $E_{i_j} \in E$ *and* $\forall E_{i_u}, E_{i_v}, 1 \leq u < v \leq R$, *there is a path in H between E_{i_u} and E_{i_v} defined on edges belonging to* $\{E_{i_1}, E_{i_2}, \ldots E_{i_R}\}\}$.
- $\mathcal{CAH}_{H_A}[P^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$ *with* $E_{i_j} \in E$ *and* $E_{i_1}, E_{i_2}, \ldots E_{i_R}$ *is a path in* $H\}$.
- $\mathcal{CAH}_{H_A}[F^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$ *with* $E_{i_j} \in E$ *and* $\exists E_{i_u}, 1 \leq u \leq R, \forall E_{i_v}, 1 \leq v \leq R$ *and* $v \neq u$, E_{i_u} *and* E_{i_v} *are neighbours* $\}$.
- $\mathcal{CAH}_{H_A}[U^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E_i \in E, \exists! E'_j \in E' : E_i \subset E'_j\}$.
- $\mathcal{CAH}_{H_A}[B^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$ *with* $E_{i_j} \in E$ *and* $\exists E_k \in E$ *such that* $\forall E_{i_v}, 1 \leq v \leq R$ $E_k \neq E_{i_v}$ *and* E_k *and* E_{i_v} *are neighbours* $\}$.

If $\forall E'_i \in E', E'_i = E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$, *these classes will be denoted* $\mathcal{CAH}_{H_A}[X]$ *for* $X = C, P, F, U$ *or* B.

The concept of separator is essential in the methods exploiting the structure, because space complexity directly depends on their size. This concept is introduced here to impose a new restriction. This one will make it possible to limit the separators to an existing subset of those in the hypergraph of reference:

Definition 8 *The set of separator-based coverings of a CAH $H_A = (X, E)$ is defined by $\mathcal{CAH}_{H_A}[S] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i, E'_j \in E', i \neq j, \exists E_k, E_l \in E, k \neq l : E'_i \cap E'_j = E_k \cap E_l\}$.*

In fact, this class imposes both the unicity and the connexity. It thus corresponds to restrictions of the two classes $[U]$ and $[C]$. One deduces from it that the unicity of covering and the connexity of the covered edges impose the conservation of the separators of H_A:

Theorem 1 $\mathcal{CAH}_{H_A}[S] = \mathcal{CAH}_{H_A}[U] \cap \mathcal{CAH}_{H_A}[C]$

By lack of place, we do not provide the proof of theorems 1-6.

Let us note that this classification is not exhaustive. We could consider additional classes but their interest and thus their study would probably be limited. Finally, the computation of an element of these various classes is easy in terms of complexity. For example, given H and H_A (H_A can be obtained as a tree-decomposition), we can compute $H'_A \in \mathcal{CAH}_{H_A}[S]$ by merging neighboring edges of H_A.

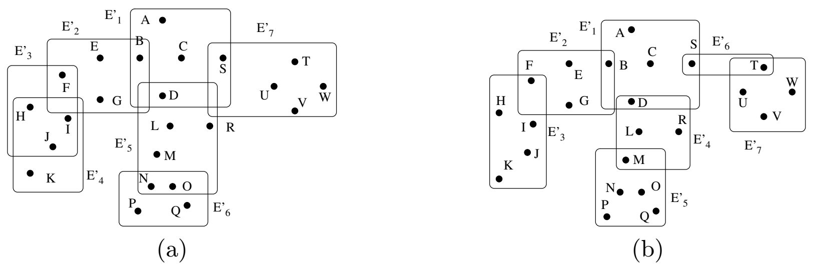

Fig. 2. (a) In this covering, neither connexity ($E'_6 = E_{11} \cup E_{12}$), nor unicity ($E_5 \subset E'_4 \cap E'_3$) and then separators (e.g. $E'_4 \cap E'_3$) are satisfied. This covering belongs to $\mathcal{CAH}_{H_A}[C^+]$. (b) This covering belongs to $\mathcal{CAH}_{H_A}[S]$.

In all the cases, one can observe that the value of the CAH-width increase inside these classes, and then is larger than α, the width associated to the hypergraph of reference H_A. In particular, for the classes of the type $\mathcal{CAH}_{H_A}[X^+]$, it is an additive increase:

Theorem 2 *$\forall H'_A \in \mathcal{CAH}_{H_A}[X^+]$ with $X = C, P, F, U$ or B, $\exists \Delta \geq 0$ such that $\alpha' \leq \alpha + \Delta$.*

Concerning the other classes, the increase is multiplicative. Indeed, in each case, covering is related to the merging of edges of (X, E):

Theorem 3 *$\forall H'_A \in \mathcal{CAH}_{H_A}[C], \exists \delta \geq 1$ such that $\alpha' \leq \delta(\alpha - s^-) + s^-$, where s^- is the minimum size of separators.*

For classes $\mathcal{CAH}_{H_A}[U]$ and $\mathcal{CAH}_{H_A}[B]$, edges can be deconnected and consequently with empty intersections. So, we cannot take into account the size of separators:

Theorem 4 $\forall H'_A \in \mathcal{CAH}_{H_A}[U] \cup \mathcal{CAH}_{H_A}[B], \exists \delta \geq 1$ *such that* $\alpha' \leq \delta.\alpha$.

These remarks are useful because they have consequences on the complexity of the algorithms which will exploit these coverings. They show in particular that classes $\mathcal{CAH}_{H_A}[C]$ (and then $[P]$, $[F]$ and $[S]$) must be privileged rather than classes $\mathcal{CAH}_{H_A}[U]$ or $\mathcal{CAH}_{H_A}[B]$. Concerning the size of the separators, one can observe that for the class $\mathcal{CAH}_{H_A}[S]$, the value s associated to H_A is an upper bound for any considered hypergraph H'_A. Formally:

Theorem 5 $\forall H'_A \in \mathcal{CAH}_{H_A}[S], s' \leq s$.

This study indicates us the most promising classes. From a theoretical viewpoint, it seems that a class as $\mathcal{CAH}_{H_A}[S]$ could be the most useful, on condition that we limit the size of the induced width.

In the sequel, we exploit these concepts at the algorithmic level. So, each CAH is thus now equipped of a privileged edge - the root - from which the search begins. Consequently, the connections between edges of the hypergraph will be oriented. Thus, certain concepts introduced before will be now expressed with words such as "hyperedge father", "hyperedge son" or "hyperedge brother" like for trees.

4 Algorithmic Exploiting of the CAHs

Several methods have been proposed to exploit properties related to the acyclicity of constraint networks. We have chosen to extend BTD. This method has shown its practical interest and it is easy to extend it to other formalisms [9, 20, 10]. It relies on the concept of tree-decomposition of graph, but its adaptation to acyclic hypergraphs is easy. BTD explores the search space by assigning the variables according to an order induced by a tree-decomposition (E, T). BTD initially chooses a node E_1 as the root of T. Thus, T is now directed. For a node E_i, $Father(E_i)$ denotes its father node, and $Sons(E_i)$ the set of its sons. BTD carries out a backtracking search, by using an induced order and maintaining an assignment $\mathcal{A}$ of the variables. During the search, BTD assigns the variables V_{E_i} of the current cluster E_i. Assume that it succeeds to extend the current assignment on these variables. If E_i is a leaf of T, the search continues on another part of the problem. If E_i has sons, BTD will continue the search on a son $E_j \in Sons(E_i)$. Note that the assignment $\mathcal{A}[E_i \cap E_j]$ allows in fact to disconnect the problem in two independent subproblems. One independent subproblem is the one located below E_i which is rooted in E_j. Then two possibilities arise. Either, $\mathcal{A}[E_i \cap E_j]$ has never been computed until now and then the search continues on the subproblem rooted in E_j; or, $\mathcal{A}[E_i \cap E_j]$ has been computed before. In case the subproblem rooted in E_j has a consistent extension of $\mathcal{A}[E_i \cap E_j]$, this information has been recorded as a *structural good of E_i with respect to E_j* (i.e. a consistent assignment on the separator $E_i \cap E_j$ which can be consistently extended on the subproblem rooted on E_j). Otherwise, the subproblem rooted in E_j has no consistent extension of $\mathcal{A}[E_i \cap E_j]$. This information has been recorded as *structural nogood of E_i with respect to E_j* (i.e. a consistent assignment on $E_i \cap E_j$ which cannot be extended consistently on the subproblem rooted on E_j). In these two cases, the search will be immediately stopped on this part of the problem, either by a success (good), or by a failure (nogood). The efficiency of BTD is based on these principles. The time complexity of BTD is $O(exp(w+1))$. Its space complexity can be limited to $O(exp(s))$, although this complexity is never reached in practice.

We propose an extension of BTD, called *BDH* (for "Backtracking on Dynamic covering by acyclic Hypergraphs"), and based on dynamic exploitation of the CAH. This approach will make it possible to effectively integrate more dynamic variable ordering heuristics. Such heuristics are necessary to ensure an effective practical solving. In order to make the implementation easier and to guarantee interesting time and space complexity bounds, we will only consider hypergraphs in $\mathcal{CAH}_{H_A}[S]$, with H the constraint hypergraph of the given problem and H_A its reference hypergraph.

Firstly, we must exploit an orientation of the hypergraph by considering edges as nodes and distinguishing an edge E_1 as the root. The neighbouring edges of E_1 are its sons and recursively the neighbouring edges of an edge E_i are its sons except the one on the path from the root to E_i, actually its father. Let E_i be a node, $Father(E_i)$ denotes its father node and $Sons(E_i)$ its son set. The descent of E_i is the set of variables in the edges contained in the acyclic sub-hypergraph rooted on E_i. The subproblem rooted on E_i is the subproblem induced by the variables in the descent in E_i.

BDH has 4 inputs: $\mathcal{A}$ the current assignment, E'_i the current edge, $V_{E'_i}$ the set of unassigned variables in E'_i and H'_A the current hypergraph. This hypergraph is computed recursively. The choice (heuristic) of a new covering hypergraph H''_A in $\mathcal{CAH}_{H_A}[S]$ is made before the beginning of the search on a subproblem. BDH solves recursively the subproblems rooted on E'_i and returns *true* if $\mathcal{A}$ can be consistently extended on this subproblem and *false* otherwise. At the first call, the assignment $\mathcal{A}$ is empty, the subproblem rooted on E_1 corresponds to the whole problem and H_A is the hypergraph of reference. Like in BTD, the order according to which variables are assigned is partially induced by the current hypergraph H'_A. During the search, the covering hypergraph is modified to take into account the evolutions of the problem. While $V_{E'_i}$ is not empty, BDH chooses a variable x in $V_{E'_i}$ (line 16) and a value in its domain (line 18) (if not empty) that verifies all the constraints (included those induced by the nogoods). Then $BDH(\mathcal{A} \cup \{x \leftarrow v\}, E'_i, V_{E'_i} \backslash \{x\}, H'_A)$ is called in the rest of the edge (line 19). When all the variables in E'_i are assigned, the algorithm chooses a son E'_j of the current edge (line 4) (if exists). If $\mathcal{A}[E'_i \cap E'_j]$ is a good (line 5), we know that $\mathcal{A}$ can be consistently extended on $Desc(E'_j)$ (descent of E'_j). Likewise, if E'_i contains a separator $s_k = E_{k_u} \cap E_{k_{u'}}$ of H_A, with $E_{k_{u'}}$ a son of E_{k_u}, such that

Algorithm 1: BDH($\mathcal{A}, E'_i, V_{E'_i}, H'_A$)

```
1  if $V_{E'_i} = \emptyset$ then
2      $Cons \leftarrow true$ ; $F \leftarrow Sons(E'_i)$
3      while $F \neq \emptyset$ and $Cons$ do
4          Choose $E'_j$ in $F$ ; $F \leftarrow F \backslash \{E'_j\}$
5          if $\mathcal{A}[E'_j \cap E'_i]$ is a good then $Cons \leftarrow true$
6          else
7              if $\exists s_k = E_{k_u} \cap E_{k_{u'}}$ in $H_A$ s.t.
               $E_{k_{u'}} \in Sons(E_{k_u}), s_k \subset E'_i, E'_j \subset Desc(E_{k_{u'}})$ and $\mathcal{A}[s_k]$ is a good then
               $Cons \leftarrow true$
8              else
9                  Choose $H''_A$ induced by $H'_A$ and $E'_j$ with root $E''_1$
10                 $Cons \leftarrow BDH(\mathcal{A}, E''_1, E''_1 \backslash (E''_1 \cap E'_i), H''_A)$
11                 if $Cons$ then Record the good $\mathcal{A}[E''_1 \cap E'_i]$
12                 else Record the nogood $\mathcal{A}[E''_1 \cap E'_i]$

13     if $Cons$ then $\forall E_u \cap E_v \subset E'_i$, record the good $\mathcal{A}[E_u \cap E_v]$
14     return $Cons$
15 else
16     Choose $x \in V_{E'_i}$ ; $d_x \leftarrow D_x$ ; $Cons \leftarrow false$
17     while $d_x \neq \emptyset$ and $\neg Cons$ do
18         Choose $v$ in $d_x$ ; $d_x \leftarrow d_x \backslash \{v\}$
19         if $\mathcal{A} \cup \{x \leftarrow v\}$ satisfies constraints and nogoods then
           $Cons \leftarrow BDH(\mathcal{A} \cup \{x \leftarrow v\}, E'_i, V_{E'_i} \backslash \{x\}, H'_A)$
20     return $Cons$
```

the subproblem rooted on E'_j is included in the one rooted on $E_{k_{u'}}$ and if $\mathcal{A}[s_k]$ is a good (line 7), we know that $\mathcal{A}$ can be consistently extended on $Desc(E_{k_{u'}})$. So $\mathcal{A}$ can also be consistently extended on $Desc(E'_j)$. Thus a forward-jump is performed and the algorithm keeps on the search on the rest of the problem. Since the nogoods are used like constraints, the assignment $\mathcal{A}$ is stopped in E'_i if it contains a nogood. If E'_i contains no (no)good, then the search continues on the subproblem rooted on E'_j. If $\mathcal{A}$ admits a consistent extension on this subproblem, $\mathcal{A}[E'_i \cap E'_j]$ is recorded as a good (line 11) and $true$ is returned, else $\mathcal{A}[E'_i \cap E'_j]$ is recorded as a nogood (line 12) and $false$ is returned. If BDH fails to consistently extend $\mathcal{A}$ on E'_i then it returns $false$.

This extension of BTD brings several advantages. It makes it possible to propose a large number of heuristics since we are freed from the initial structure H_A (e.g. the next variable to assign is not necessarily chosen in one single edge E_j). Then BDH can exploit all the (no)goods recorded on all the separators of the reference CAH included separators which are currently included in a larger edge in the current CAH, while BTD with a class 5 order only exploits (no)goods on the intersection of two clusters of the current covering. The space complexity is not modified ($O(exp(s))$) because the search is based on the same set of separators that those of H_A. Finally, its implementation is straightforward because only hypergraphs of $\mathcal{CAH}_{H_A}[S]$ will be considered and these hypergraphs will be obtained by a simple merging of neighboring edges in H_A.

Theorem 6 *BDH is sound, complete and terminates.*

BDH uses a subset of hypergraphs in $\mathcal{CAH}_{H_A}[S]$. The theorem 2 states that there exists $\Delta \geq 0$ such that for all H'_A in this subset, $\alpha' \leq \alpha + \Delta$. The time complexity of the method depends on Δ. Besides, Δ can be parametrized : it is possible to bound the value of Δ and only consider covering hypergraphs in $\mathcal{CAH}_{H_A}[S]$ whose width is bounded by $\alpha + \Delta$. Anyway, the time complexity of BDH is given by the following theorem.

Theorem 7 *The time complexity of BDH is $O(exp(\alpha + \Delta + 1))$.*

Proof Let (X, D, C, R) be a CSP, H_A the reference CAH of $H = (X, C)$. Let V be a set of $\alpha + \Delta + 1$ variables such that $\exists E_{u_1}, \ldots, E_{u_r}$, a path in H_A (with $r \geq 2$ since $|V| = \alpha + \Delta + 1$ and α is the maximum size of the edges of H_A), $V \subset E_{u_1} \cup \ldots \cup E_{u_r}$ and $E_{u_2} \cup \ldots \cup E_{u_{r-1}} \subsetneq V$ (resp. $E_{u_1} \cap E_{u_2} \subsetneq V$) if $r \geq 3$ (resp. $r = 2$). We will prove first that any assignment on V is computed only twice. $\forall H'_A \in \mathcal{CAH}_{H_A}[S], \exists E'_{i_1}, \ldots, E'_{i_{r'}}$, a path, (with $r' \geq 2$ since the maximum size of the edges in H'_A is $\alpha + \Delta$) such that $V \subset E'_{i_1} \cup \ldots \cup E'_{i_{r'}}$ and $E'_{i_2} \cup \ldots \cup E'_{i_{r'-1}} \subsetneq V$ (resp. $E'_{i_1} \cap E'_{i_2} \subsetneq V$) if $r' \geq 3$ (resp. $r' = 2$). Let $\mathcal{A}$ be an assignment to extend on V. The order in which the variables of $\mathcal{A}$ will be assigned is induced by $H'_A \in \mathcal{CAH}_{H_A}[S]$. We suppose in the following that the edges covering V are ordered w.r.t. the order they are assigned: E'_{i_j} is visited before $E'_{i_{j'}}$ if $j < j'$. Thus E_{u_1} is the first assigned edge among those of the path in H_A covering V and $s_1 = E_{u_1} \cap E_{u_2}$ is included in E'_{i_1} because $E_{u_1} \subset E'_{i_1}$.

If $E'_{i_1} \cap E'_{i_2} = s_1$, the solving of the subproblem rooted on E'_{i_2} with the assignment $\mathcal{A}$ leads to the recording of one (no)good on the separator s_1: $\mathcal{A}[s_1]$. Let $\mathcal{B}$ be a new assignment that we try to extend on V with the same values in $\mathcal{A}[V]$ and $H''_A \in \mathcal{CAH}_{H_A}[S]$ induce the order in which the variables of $\mathcal{B}$ were assigned. $\exists E''_{j_1}, \ldots, E''_{j_{r''}}, r'' \geq 2$, a path in H''_A such that $V \subset E''_{j_1} \cup \ldots \cup E''_{j_{r''}}$ and $E''_{j_2} \cup \ldots \cup E''_{j_{r''-1}} \subsetneq V$ (resp. $E''_{j_1} \cap E''_{j_2} \subsetneq V$) if $r'' \geq 3$ (resp. $r' = 2$). Since $s_1 \subset E''_{j_1}$, as soon as E''_{j_1} is totally assigned, $\mathcal{A}[s_1]$ stops the assignment on V. Thus $\mathcal{A}[V]$ is computed twice only on $\alpha + \Delta$ variables of V at worst.

Now we suppose $E'_{i_1} \cap E'_{i_2} \neq s_1$. If $\mathcal{A}[E'_{i_1} \cap E'_{i_2}]$ can be consistently extended on the subproblem rooted on E'_{i_2} then $\mathcal{A}[E'_{i_1} \cap E'_{i_2}]$ is recorded as a good. Since $s_1 \subset E'_{i_1}$, if this current assignment $\mathcal{A}$ can be consistently extended on the unassigned variables in the subproblem rooted on E'_{i_1} then $\mathcal{A}[s_1]$ is recorded as a good. Otherwise, this assignment cannot be consistently extended on at least one subproblem rooted on $E'_{t'}$ an edge of H'_A such that $E'_{t'} \neq E'_{i_l}, l = 1, \ldots, r'$ and $E'_{t'} \cap E'_{i_1} \neq \emptyset$. Thus $\mathcal{A}[E'_{i_1} \cap E'_{t'}]$ is recorded as a nogood. When we try to extend $\mathcal{B}$ on V, if $\mathcal{A}[s_1]$ is a good then for the same reasons as previously $\mathcal{A}[V]$ is computed twice only on $\alpha + \Delta$ variables of V at worst.

If $\mathcal{A}[s_1]$ is not a good then $\mathcal{A}[E'_{i_1} \cap E'_{t'}]$ is recorded as a nogood. Nevertheless, it is possible to compute $\mathcal{A}[V]$ twice if $(E'_{i_1} \cap E'_{i_2}) \cup (E'_{i_1} \cap E'_{t'}) \subset E''_{j_{r''}}$. If $\mathcal{B}[E''_{j_1} \cap E''_{j_2}]$ can be consistently extended on the subproblem rooted on E''_{j_2} then $\mathcal{B}[E''_{j_1} \cap E''_{j_2}]$ is recorded as a good. Since $s_1 \subset E''_{j_1}$, either $\mathcal{B}[s_1]$ is recorded as a good or $\mathcal{A}[E''_{j_1} \cap E''_{t''}]$ is recorded as a nogood, with $E''_{t''}$ an edge of H''_A such that $E''_{t''} \neq E''_{jl}, l = 1, \ldots, r''$ and $(E''_{t''} \cap E''_{j_1}) \neq \emptyset$. If $\mathcal{B}[s_1]$ is a good then $\mathcal{B}[V]$ is computed again only on $\alpha + \Delta$ variables of V at worst.

If $\mathcal{B}[E''_{j_1} \cap E''_{t''}]$ is a nogood: two nogoods are recorded on two separators in V. As soon as the first one of these separators is totally assigned, the nogood related to this separator stops the assignment on V. Thus $\mathcal{A}[V]$ is not computed again.

If $\mathcal{B}[E''_{j_1} \cap E''_{j_2}]$ cannot be consistently extended on the subproblem rooted on E''_{j_2} then $\mathcal{B}[E''_{j_1} \cap E''_{j_2}]$ is recorded as a nogood. Since two nogoods are recorded, $\mathcal{A}[V]$ is not computed again.

If $\mathcal{A}[E'_{i_1} \cap E'_{i_2}]$ cannot be consistently extended on the subproblem rooted on E'_{i_2} then we use the same reasoning as in the first part of this proof to demonstrate that $\mathcal{A}[V]$ is computed only twice at worst.

e prove that any assignment on V is computed only twice at worst. Now, we suppose that H is covered by a set of V_i which verifies that V_i contains $\alpha + \Delta + 1$ variables such that $\exists E_{u_1}, \ldots, E_{u_r}$, a branch in H (with $r \geq 2$), $V \subset E_{u_1} \cup \ldots \cup E_{u_r}$ and $E_{u_2} \cup \ldots \cup E_{u_{r-1}} \subsetneq V$ (resp. $E_{u_1} \cap E_{u_2} \subsetneq V$) if $r \geq 3$ (resp. $r = 2$). It is sufficient to cover each branch in H by some V_i (they can intersect). On each V_i covering H an assignment is computed twice at worst. The number of possible assignments on V_i is $d^{\alpha+\Delta+1}$. Thus the number of possible assignments on the variables of the problem is bounded by $number_{V_i}.d^{\alpha+\Delta+1}$, with $number_{V_i}$ the number of sets V_i covering H. Since the number of V_i is bounded, then the time complexity of BDH is in $O(exp(\alpha + \Delta + 1))$. $\square$

5 Experimental study

In this section, we assess the efficiency of BDH on benchmarks (structured random CSPs) presented in [18] with the same empirical protocol and PC. The reference hypergraph is computed thanks to the triangulation of the constraint graph H performed in [18]. We also use the best heuristics given in this paper for an efficient traversal of a hypergraph: *minexp*. Now, we define some heuristics for computing the covering hypergraphs dynamically. In order to reduce the space complexity, we merge edges in the following heuristics only if their intersection size is greater than 5 unless the size of the resulting edge is greater than 1.5α (Δ is bounded by 0.5α). According to the order edges are considered, different covering hypergraphs are computed. We define an order on edges based on the traversal heuristic *minexp*. *Br*: the current edge is merged with its first son (w.r.t. *minexp*), the first son of this one and so on, if the size of their separator is greater than 5 and while the size of the resulting edge is less than 1.5α. $Br + vp$: first, the reference hypergraph is modified such that edges with less than 3 variables not included in their parent are merged to it. Then the heuristic *Br* is used to compute dynamically covering hypergraphs. Table 1 shows the runtime of BDH with heuristics Br, $Br + vp$, as well as with heuristics based on *minexp* in the Classes 3 and 4 defined in [18] (with a hypergraph with maximum size of edge intersections bounded by 5 for the Class 4). The Class 4 obtains better results than the Class 3. The heuristic *Br* has very good results, close to Class 4 ones, and succeeds in solving all the instances in $(250, 20, 20, 99, 10, 25, 10)$ while two are not solved in the Class 4 within 1,800 s. Having a better reference hyper-

Table 1. Runtime (in s) on random partial structured CSPs: (a) Class 3, (b) Class Br, (c) Class $Br + vp$ and (d) Class 4.

CSP (n, d, w, t, s, ns, p)	(a)	(b)	(c)	(d)
$(150, 25, 15, 215, 5, 15, 10)$	2.04	1.78	1.93	1.76
$(150, 25, 15, 237, 5, 15, 20)$	8.84	1.23	1.12	1.10
$(150, 25, 15, 257, 5, 15, 30)$	4.20	1.40	1.30	3.19
$(150, 25, 15, 285, 5, 15, 40)$	3.08	1.17	0.27	0.23
$(250, 20, 20, 107, 5, 20, 10)$	14.87	10.49	12.89	13.14
$(250, 20, 20, 117, 5, 20, 20)$	11.43	6.97	13.18	5.98
$(250, 20, 20, 129, 5, 20, 30)$	29.24	3.62	2.80	2.87
$(250, 20, 20, 146, 5, 20, 40)$	12.26	6.99	3.97	7.89
$(250, 25, 15, 211, 5, 25, 10)$	7.18	4.19	3.79	4.64
$(250, 25, 15, 230, 5, 25, 20)$	3.86	2.76	1.69	2.36
$(250, 25, 15, 253, 5, 25, 30)$	3.86	3.73	2.61	2.08
$(250, 25, 15, 280, 5, 25, 40)$	56.66	13.87	11.88	12.33
$(250, 20, 20, 99, 10, 25, 10)$	112.88	82.07	78.16	144.20
$(500, 20, 15, 123, 5, 50, 10)$	7.11	2.23	2.00	3.12
$(500, 20, 15, 136, 5, 50, 20)$	6.11	3.01	3.23	4.13

graph w.r.t. the solving allows the heuristic $Br + vp$ to obtain generally the best results. These experiments highlight the great importance of a good reference hypergraph (w.r.t. the solving) for the search. Furthermore a dynamic exploitation of this hypergraph structure leads to significant improvements. Let us note that BDH with heuristics in Classes 3 and 4 is identical to BTD so it obtains similar results to BTD's ones in [18] for these classes. Moreover, the methods FC and MAC fail in solving most of these instances within 1,800 s. Likewise, TC and so the hypertree-decomposition method do not succeed in solving most of these instances due to an expensive time and space consumption.

6 Conclusion

In this paper, we have proposed a framework based on an old concept: the covering of a problem by acyclic hypergraphs. We have shown that such an approach seems to be more useful than the tree-decomposition based one. In particular, this approach offers more flexibility to the variable ordering heuristic and obtains better theoretical and practical results. Moreover, it turns to be operational in practice, what is not the case for the hypertree-decomposition method and its generalizations.

Firstly, these coverings have been studied theoretically in order to determine their characteristics and properties. After, we have focused our study on a particular class of coverings that preserves time and space complexity results and also improves some of them. One of the major results presented in this paper is the theorem indicating that the addition of Δ variables to offer freedom in a dynamic variable ordering, induces a limited time complexity in $O(exp(\alpha + \Delta + 1))$, which outperforms previous results significantly (namely $O(exp(2(w + \Delta + 1) - s^-))$

with s^- the minimum size of separators [18] and $\alpha = w + 1$). Furthermore, we have shown that our approach makes easier the implementation and permits to improve practical efficiency showing thus the practical interest of this approach. This work, introduced within the framework of CSPs, must now be extended to other formalisms like SAT, MAX-SAT, VCSP or probabilistic networks.

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

References

1. I. Rish and R. Dechter. Resolution versus Search: Two Strategies for SAT. *Journal of Automated Reasoning*, 24:225–275, 2000.
2. J. Huang and A. Darwiche. A structure-based variable ordering heuristic for SAT. In *Proceedings of IJCAI*, pages 1167–1172, 2003.
3. W. Li and P. van Beek. Guiding Real-World SAT Solving with Dynamic Hypergraph Separator Decomposition. In *Proceedings of ICTAI*, pages 542–548, 2004.
4. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
5. R. Dechter. Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.
6. A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126:5–41, 2001.
7. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30:479–513, 1983.
8. G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.
9. C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of CP*, pages 709–723, 2003.
10. S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proceedings of AAAI*, pages 22–27, 2006.
11. N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.
12. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
13. D. Cohen, Peter Jeavons, and M. Gyssens. A Unified Theory of Structural Tractability for Constraint Satisfaction and Spread Cut Decomposition. In *Proc. of IJCAI*, pages 72–77, 2005.
14. M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proc. of SODA*, pages 289–298, 2006.
15. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.
16. P. Jégou and C. Terrioux. Decomposition and good recording for solving Max-CSPs. In *Proc. of ECAI*, pages 196–200, 2004.
17. R. Marinescu and R. Dechter. Dynamic Orderings for AND/OR Branch-and-Bound Search in Graphical Models. In *Proc. of ECAI*, pages 138–142, 2006.
18. P. Jégou, S.N. Ndiaye, and C. Terrioux. ‘Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs. In *Proc. of IJCAI*, pages 112–117, 2007.
19. R. Dechter. *Constraint processing.* Morgan Kaufmann Publishers, 2003.
20. M. Sachenbacher and B. C. Williams. Bounded Search and Symbolic Inference for Constraint Optimization. In *Proc. of IJCAI*, pages 286–291, 2005.