Extending to Soft and Preference Constraints a Framework for Solving Efficiently Structured Problems

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

Abstract

This paper deals with the problem of solving efficiently structured COPs (Constraints Optimization Problems). The formalism based on COPs allows to represent numerous real-life problems defined using constraints and to manage preferences and soft constraints. In spite of theoretical results, [15] has discarded (hyper)tree-decompositions for the benefit of coverings by acyclic hypergraphs in the CSP area. We extend here this work to constraint optimization. We first study these coverings from a theoretical viewpoint. Then we exploit them in a framework aiming not to define a new decomposition, but to make easier a dynamic management of the structure during the search (unlike most of structural methods which usually exploit the structure statically), and so the use of dynamic variable ordering heuristics. Thus, we provide a new complexity result which outperforms significantly the previous one given in the literature. Finally, we assess the practical interest of these notions.

1 Introduction

Preference handling, when preferences can be expressed by constraints as with COPs or VCSPs, defines hard problems from a theoretical viewpoint. So, algorithms to manage them must exploit all usable properties. For example, topological properties, i.e. structural properties of data must be exploited. In the past, the interest for the exploitation of structural properties of a problem was attested in various domains in AI: for checking satisfiability in SAT [20, 12, 18], in CSP [8], in Bayesian or probabilistic networks [6, 4], in relational databases [1, 10], in constraint optimization [23, 5]. Complexity results based on topological properties of the network structure have been proposed. A large part of these works has been realized on formalisms which can take into account preferences. Generally, they rely on the properties of a tree-decomposition [21] or a hypertree-decomposition [9] of the network, which can be considered as an acyclic hypergraph (a hypertree) covering the network.

On the one hand, 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, while their space complexity can generally be reduced to $O(exp(s))$ where s is the size of the largest intersection between two neighboring clusters of the tree-decomposition. An example of an efficient method exploiting tree-decomposition is BTD [16] which achieves a enumerative search driven by the tree-decomposition. Such a method can be seen as driven by the assignment of variables (or as a "variable driven" method).

On the other hand, from a theoretical viewpoint, methods based on hypertree-decomposition are more interesting than those based on tree-decomposition [9]. If we consider hypertree-decomposition, the time complexity of the best methods is in $O(exp(k))$, with k the width of the used hypertree-decomposition. We can consider them as "relation driven" approaches since they consist in grouping the constraints (and so the relations) in nodes of the hypertree and solve the problem by computing joins of relations. Recently, hypertree-decomposition has been outperformed by generalized hypertree-decomposition [3, 11].

These theoretical time complexities can really outperform the classical one which is $O(exp(n))$ ($k < w < n$) with n the number of variables of the considered problem. However, the practical interests of decomposition approaches have not been proved yet, except in some recent works around CSPs [16] or for managing preferences and soft constraints using Valued CSPs [17, 19, 5]. This kind of approaches seems to be the most efficient from a practical viewpoint. Indeed, the second international competition around MAX-CSP (a basic framework for preferences) has been won by "Toolbar-BTD" which exploits simultaneously decomposition with BTD and valued propagation techniques (http://www.cril.univ-artois.fr/CPAI06/round2/results/results.php?idev=7) [2]. We can note that the effective methods rely on the "variable driven" approach. A plausible explanation relies on the fact that "relation driven" methods need to compute joins which may involve many variables and so require a huge amount of memory. So, despite the theoretical results, we prefer exploit here "variable driven" decompositions.

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 [19, 14, 15] notably to optimization, preferences and soft constraints. Like in [15], we prefer exploit here the more general and useful concept of covering by acyclic hypergraph rather than one of tree-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 $H_A = (X, E)$ s.t. for each hyperedge $C_i \in C$, there is an hyperedge $E_i \in E$ covering C_i ($C_i \subset E_i$). From [15], given H_A, we can define various classes of acyclic hypergraphs which cover H_A. Here, we focus our study on a class of coverings which preserve the separators. We exploit it to propose a framework for a dynamic management of the structure: during the search, we can take into account not only one acyclic hypergraph covering, but a set of coverings in order to manage heuristics dynamically (while usually structural methods only exploit the structure statically). Thanks to this formal framework, we present a new algorithm (called BDH$_{val}$ 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. Finally, we provide theoretical and practical results showing that we can preserve already known complexity results and also improve some of them and the practical interest of this approach.

In the following, we present our work by using the VCSP formalism [22], but any COP formalism could be used instead. A *valued CSP* (VCSP) is a tuple $\mathcal{P} = (X, D, C, E, \oplus, \preceq)$. X is a set of n variables which must be assigned in their respective finite domain from D. Each constraint of C is a function on a subset of variables which associates to each tuple a valuation from E. $\bot$ and $\top$ are respectively the minimum and maximum elements of E. $\oplus$ is an aggregation operator on elements of E. Given an instance, the problem generally consists in finding an assignment on X whose valuation is minimum, what is a NP-hard problem. The VCSP structure can be represented by the hypergraph (X, C), called the *constraint hypergraph*.

The next section deals with coverings by acyclic hypergraphs. The third one describes how these coverings can be exploited on the algorithmic level and gives some theoretical results. Then section 4 provides some variable ordering heuristics based on these coverings. Finally, we present practical results in section 5 before concluding in section 6.

2 Coverings by acyclic hypergraphs

The basic concept which interests us here is the acyclicity of networks. Often, it is expressed by considering tree-decompositions or hypertree-decompositions, or more generally, coverings of variables and constraints by acyclic hypergraphs. In this paper, we 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 [1] and we give an equivalent definition based on the notions of α-cycle in a hypergraph $H = (X, C)$.

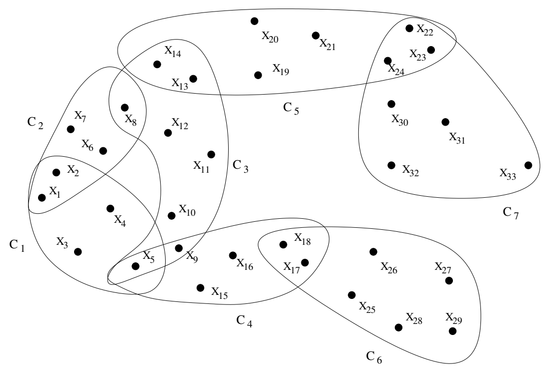

Figure 1. A hypergraph.

Definition 1 *Let C_u and C_v be hyperedges such that $C_u \cap C_v \neq \emptyset$. We call sequence of neighborhood connecting C_u and C_v, a sequence $(C_u = C_{i_1}, C_{i_2}, \ldots, C_{i_R} = C_v)$ such that $R > 2$ and $C_u \cap C_v \subsetneq C_{i_j} \cap C_{i_{j+1}}$, for $j = 1, \ldots, R-1$.*

Two hyperedges are α-neighbors if it does not exist another path (a sequence of neighborhood) joining them.

Definition 2 *Let C_u and C_v be two hyperedges of H. C_u and C_v are α-neighboring if it does not exist a sequence of neighborhood connecting them.*

For example, the hypergraph given in figure 1 contains two hyperedges C_1 and C_4 with a non empty intersection which are not α-neighbors because (C_1, C_3, C_4) is a sequence of neighborhood connecting C_1 and C_4.

Definition 3 *An α-path in H is a sequence of hyperedges $(C_{i_1}, \dots, C_{i_R})$ such that $\forall j, 1 \leq j < R, C_{i_j}$ and $C_{i_{j+1}}$ are α-neighboring.*

Definition 4 *An α-cycle in H is an α-path $(C_{i_1}, C_{i_2}, \dots, C_{i_R})$ such that $R > 3$, $C_{i_1} = C_{i_R}$, $\nexists 1 \leq a \neq b < R$, $C_{i_a} \cap C_{i_{a+1}} \subset C_{i_b} \cap C_{i_{b+1}}$.*

The next theorem shows the equivalence between the acyclicity of hypergraphs and the existence of α-cycle.

Theorem 1 *H is acyclic iff H does not contain an α-cycle.*

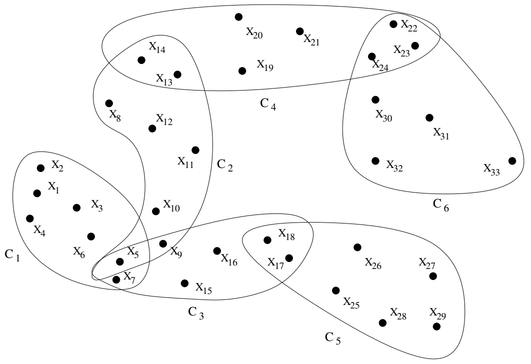

Figure 2. An acyclic hypergraph.

Hypergraph presented in figure 1 is not acyclic because (C_1, C_2, C_3, C_1) is a α-cycle. On the contrary, the hypergraph of figure 2 is acyclic.

Now, let us define acyclic covering.

Definition 5 *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 hyperedge $C_i \in C$, there exists $E_j \in E$ such that $C_i \subset E_j$. The width γ of a CAH (X, E) is equal to $max_{E_i \in E}|E_i|$. The CAH-width γ^* 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 [7]) is very close to one of tree-decomposition. Particularly, given a tree-decomposition we can easily compute a CAH. Moreover, the CAH-width γ^* is equal to the tree-width 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 γ, while it can exist several tree-decompositions of width w s.t. $\gamma = w + 1$. The best structural methods for solving a COP with a CAH of width γ have a time complexity in $O(exp(\gamma))$ while their space complexity can be reduced to $O(exp(s))$ with $s = \max_{E_i, E_j \in E} |E_i \cap E_j|$ in H_A.

In [15], given a hypergraph $H = (X, C)$ and one of its CAHs $H_A = (X, E)$, we have defined and studied 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 rely on a particular CAH H_A, called *CAH of reference*. [15] aims to study different classes of acyclic coverings, to manage dynamically, during the search, acyclic coverings of the considered CSP. By so doing, we hope to manage dynamic heuristics to optimize the search while preserving complexity results.

We first introduce the notion of set of covering:

Definition 6 *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 $\mathcal{CAH}_{H_A}$.

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 $\{E_{i_1}, E_{i_2}, \dots E_{i_R}\}$. 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:*

- *$\mathcal{CAH}_{H_A}[C^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \dots \cup E_{i_R}$ with $E_{i_j} \in E$ and $\forall E_{i_u}, E_{i_v}, 1 \leq u < v \leq R$, there is a α-path in H joining E_{i_u} and E_{i_v} defined on edges belonging to $\{E_{i_1}, E_{i_2}, \dots E_{i_R}\}\}$.*

- *$\mathcal{CAH}_{H_A}[P^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \dots \cup E_{i_R}$ with $E_{i_j} \in E$ and $E_{i_1}, E_{i_2}, \dots E_{i_R}$ is a α-path in $H\}$.*

- *$\mathcal{CAH}_{H_A}[F^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \dots \cup E_{i_R}$ 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 α-neighbors $\}$.*

- $\mathcal{CAH}_{H_A}[U^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E_i \in E, \exists! E'_j \in E' : E_i \subset E'_j\}$.
- $\mathcal{CAH}_{H_A}[B^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \ldots \cup E_{i_R}$ *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* α*-neighbors* $\}$.

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 their space complexity directly depends on their size. So we define the class S of coverings which makes 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* $\mathcal{CAH}_{H_A}[S] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i, E'_j \in E', i \neq j, \exists E_k, E_l \in E, k \neq l : E'_i \cap E'_j = E_k \cap E_l\}$.

This class presents several advantages. First, it preserves the connexity of H_A. Then, computing one of its elements 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 hyperedges of H_A. Moreover, according to theorem 2, this class may have some interesting consequences on the complexity of the algorithms which will exploit it. More precisely, it allows to make a time/space trade-off since, given a hypergraph H'_A of $\mathcal{CAH}_{H_A}[S]$, it leads to increase the width of H'_A w.r.t. H_A while the maximal size of separators in H'_A is bounded by one in H_A.

Theorem 2 $\forall H'_A \in \mathcal{CAH}_{H_A}[S]$, $\exists \Delta \geq 0$ *such that* $\gamma' \leq \gamma + \Delta$ *and* $s' \leq s$.

Other classes are defined in [15], but, from a theoretical and practical viewpoint, the class S seems to be the most promising and useful one.

In the sequel, we exploit these concepts at the algorithmic level. Each CAH is thus now equipped of a privileged hyperedge (the root) from which the search begins. So, the connections between hyperedges will be oriented.

3 Algorithmic Exploitation of CAHs

In this section, we introduce the method BDH$_{val}$ which is an extension of BDH [14] to the VCSP formalism and a generalization of BTD$_{val}$ based on CAH. BDH$_{val}$ relies on the Branch and Bound technique and a dynamic exploitation of the CAH. It makes it possible to use more dynamic variable ordering heuristics which are necessary to ensure an effective practical solving. Like in BDH, we will only consider hypergraphs in $\mathcal{CAH}_{H_A}[S]$, with H_A the reference hypergraph of the constraint hypergraph H of the given problem. This class allows to guarantee good space and time complexity bounds.

BTD$_{val}$ is based on a tree-decomposition that is a jointree on the acyclic hypergraph H_A. But, this jointree is not unique: it may exist another one more suitable w.r.t. the solving. Instead of choosing arbitrary one jointree of H_A, BDH$_{val}$ computes in a dynamic way a suitable one during the solving. Moreover, it is also possible not restricting ourself to one jointree but computing a suitable one at each stage of the solving.

At each stage, BDH$_{val}$ uses a jointree T_c of H_A, computed incrementally. At the beginning, the current subtree T_{c_0} of T_c is empty. BDH$_{val}$ chooses a root hyperedge E_1 where the search begins and computes the neighbors of E_1 in T_c (among its α-neighbors in H_A). Then it adds E_1 and its neighbors to T_{c_0} and obtains the next subtree T_{c_1} of T_c. After this, BDH$_{val}$ chooses incrementally among the neighboring hyperedges those which will be merged with E_1. Let E_i be the first of these. BDH$_{val}$ computes first the neighbors of E_i, adds them to T_{c_1} and merges E_i and E_1. The sons of this new hyperedge is the union of the sons of E_1 and ones of E_i. The same operation is repeated on the new hyperedge. Let E'_1 be the hyperedge obtained and T_{cm_1} the resulting subtree. BDH$_{val}$ assigns all the variables in E'_1 and recursively solves the next subproblem among those rooted on its sons in T_{cm_1}.

Definition 9 *Let* T_c *be a jointree of* H_A*. A jointree,* T_{cm}*, related to* T_c *is one of a hypergraph* H'_A *obtained by merging some neighboring hyperedges in* T_c*. Moreover,* $H'_A \in \mathcal{CAH}_{H_A}[S]$.

$Father(E_i)$ denotes the father node of E_i and $Sons(E_i)$ its son set. The descent of E_i (denoted $Desc(E_i)$) is the set of variables in the hyperedges contained in the subtree rooted on E_i. The subproblem rooted on E_i is the subproblem induced by the variables in $Desc(E_i)$. BDH has 7 inputs: $\mathcal{A}$ the current assignment, E'_i the current hyperedge, $V_{E'_i}$ the set of unassigned variables in E'_i, $ub_{E'_i}$ the current upper bound for the subproblem $\mathcal{P}_{\mathcal{A},Father(E'_i)/E'_i}$ induced by $Desc(E'_i)$ and the assignment $\mathcal{A}[E'_i \cap Father(E'_i)]$, $lb_{E'_i}$ the lower bound of the current assignment in $Desc(E'_i)$, H'_A the current hypergraph and T_{cm_b} the current subtree. BDH$_{val}$ solves recursively the subproblem $\mathcal{P}_{\mathcal{A},Father(E'_i)/E'_i}$ and returns its optimal valuation. At the first call, the assignment $\mathcal{A}$ is empty, the subproblem rooted on E_1 corresponds to the whole problem. While $V_{E'_i}$ is not empty and the lower bound is less than the upper bound, BDH$_{val}$ chooses a variable x in $V_{E'_i}$ (line 12) and a value in its domain (line 14) (if not empty) and updates the lower bound. If the lower bound is greater or

Algorithm 1: BDH$_{val}$ $(\mathcal{A}, E'_i, V_{E'_i}, ub_{E'_i}, lb_{E'_i}, H_A, T_{cm_b})$

```
1  if $V_{E'_i} = \emptyset$ then
2      $F \leftarrow Sons(E'_i)$; $lb \leftarrow lb_{E'_i}$
3      while $F \neq \emptyset$ and $lb \prec ub_{E'_i}$ do
4          $E_j \leftarrow$ Choose-hyperedge($F$); $F \leftarrow F \backslash \{E_j\}$
5          $E'_{j'} \leftarrow Compute(T_{cm_b}, H_A, E_j)$
6          if $(\mathcal{A}[E'_{j'} \cap E'_i], o)$ is a good then $lb \leftarrow lb \oplus o$
7          else
8              $o \leftarrow$BDH$_{val}$ $(\mathcal{A}, E'_{j'}, E'_{j'} \backslash (E'_{j'} \cap E'_i), \top, \bot, H_A, T_{cm_{b+1}})$
9              $lb \leftarrow lb \oplus o$; Record the good $(\mathcal{A}[E'_{j'} \cap E'_i], o)$
10     return $lb$
11 else
12     $x \leftarrow$ Choose-var($V_{E'_i}$); $d_x \leftarrow D_x$
13     while $d_x \neq \emptyset$ and $lb_{E'_i} \prec ub_{E'_i}$ do
14         $v \leftarrow$ Choose-val($d_x$); $d_x \leftarrow d_x \backslash \{v\}$
15         $L \leftarrow \{c \in E_{\mathcal{P},E'_i} | X_c \cap V_{E'_i} = \{x\}\}$; $lb_v \leftarrow \bot$
16         while $L \neq \emptyset$ and $lb_{E'_i} \oplus lb_v \prec ub_{E'_i}$ do
17             Choose $c \in L$; $L \leftarrow L \backslash \{c\}$
18             $lb_v \leftarrow lb_v \oplus c(\mathcal{A} \cup \{x \leftarrow v\})$
19         if $lb_{E'_i} \oplus lb_v \prec ub_{E'_i}$ then
20             $ub_{E'_i} \leftarrow min(ub_{E'_i}$,BDH$_{val}(\mathcal{A} \cup \{x \leftarrow v\}, E'_i,$
                   $V_{E'_i} \backslash \{x\}, ub_{E'_i}, lb_{E'_i} \oplus lb_v, H_A, T_{cm_b})$
21     return $ub_{E'_i}$
```

equal to the upper bound, BDH$_{val}$ chooses another value or performs a backtrack. Otherwise, BDH$_{val}$ is called in the rest of the hyperedge (line 20). When all the variables in E'_i are assigned, the algorithm chooses a son E_j of E'_i (line 4) (if exists). The function $Compute$ extends the construction of T_{cmb} by computing a new hyperedge $E'_{j'}$ covering E_j. If $\mathcal{A}[E'_i \cap E'_{j'}]$ is a good (line 6), the optimal valuation on $Desc(E'_{j'})$ is added directly to the lower bound and the search continues on the rest of the problem. If $\mathcal{A}[E'_i \cap E'_{j'}]$ is not a good, then the solving continues on $Desc(E'_{j'})$. As soon as, the optimal valuation on $Desc(E'_{j'})$ is computed, we record it with the assignment $\mathcal{A}[E'_i \cap E'_{j'}]$ as a good and return it as the result (line 10).

Theorem 3 *BDH$_{val}$ is sound, complete and terminates.*

Like BDH, BDH$_{val}$ uses a subset of hypergraphs in $\mathcal{CAH}_{H_A}[S]$ for which there exists $\Delta \geq 0$ such that for all H'_A in this subset, $\gamma' \leq \gamma + \Delta$. The value of Δ can be parametrized to only consider covering hypergraphs in $\mathcal{CAH}_{H_A}[S]$ whose width is bounded by $\gamma+\Delta$. Anyway, the time complexity of BDH$_{val}$ is given by the following theorem while the space complexity remains in $(O(exp(s)))$ since the search relies on the same set of separators as H_A.

Theorem 4 *The time complexity of BDH$_{val}$ is $O(N(T_c).(\gamma + \Delta).exp(\gamma + \Delta + 1))$, with $N(T_c)$ the number of jointrees used by BDH$_{val}$.*

Proof: Let $\mathcal{P} = (X, D, C, E, \oplus, \preceq)$ be a VCSP, H_A the CAH of reference of $H = (X, C)$. Let us consider a jointree T_c built by BDH$_{val}$.

As for the proof for time complexity of BDH ([15]), it is possible to cover H_A (T_c) by sets V_a of $\gamma + \Delta + 1$ variables verifying that each assignment of their variables will not be generated by BDH$_{val}$ at most $number_{Sep}(V_a)$ times, with $number_{Sep}(V_a)$ the number of separators of H_A contained in V_a. The definition of sets V_a is exactly the same than given in the proof of BDH.

Let V_a be a set of $\gamma + \Delta + 1$ variables such that $\exists(E_{u_1}, \ldots, E_{u_r})$ a path taken in T_c, $V_a \subset E_{u_1} \cup \ldots \cup E_{u_r}$ ($r \geq 2$ since $|V_a| = \gamma + \Delta + 1$ and γ is the maximal size of hyperedges of H_A) and $E_{u_2} \cup \ldots \cup E_{u_{r-1}} \subsetneq V_a$ (respectively $E_{u_1} \cap E_{u_2} \subset V_a$) if $r \geq 3$ (respectively $r = 2$).

V_a contains $r - 1$ separators which are the intersections between hyperedges that cover it. Indeed, $(E_{u_1}, \ldots, E_{u_r})$ being a path and none hyperedge being included in another, the separators are located only between two consecutive elements in the path.

During the search, it is possible to cover V_a in different ways with the jointrees T_{cm} related to T_c. Nevertheless, at least one separator of V_a will be an intersection between two hyperedges in each T_{cm}. Let T_{cm} be a jointree related to T_c such that s_1 is an intersection between two hyperedges. The search based on this jointree will generate an assignment on V_a and record on s_1 a valued good. If s_1 is also an intersection between two hyperedges in a jointree T'_{cm} related to T_c, used during a new attempt for an assignment of variables of V_a with the same values, the valued good will allow to stop the assignment. Conversely, if s_1 is not an intersection, the location of the good can lead to produce again totally the assignment but another valued good will be recorded on another separator s_2 of V. Henceforth, if s_1 or s_2 is an intersection between hyperedges of a jointree related to T_c used during the search, the assignment will not be reproduced. Thus, an assignment on V_a can be reproduced as many times as it is possible to decompose it by its separators : thus the number of separators.

The maximum number of separators $(r - 1)$ of a V_a is bounded by $\gamma + \Delta$ because the number of elements of the path $(E_{u_1}, \ldots, E_{u_r})$ is bounded by $\gamma + \Delta + 1$.

We have proved that each assignment on V is generated at most $\gamma + \Delta$ times.

On each V_a covering T_c an assignment is produced at most $\gamma + \Delta$ times. The number of possible assignments on V_a is bounded by $d^{\gamma+\Delta+1}$. So, the number of possible assignments on the set of variables of the problem is bounded by $N(T_c).number_{V_a}.(\gamma+\Delta).d^{\gamma+\Delta+1}$, with $number_{V_a}$ the number of sets V_a covering T_c. The number of V_a being bounded by the number of hyperedges of H_A, the complexity of BDH$_{val}$ is then $O(N(T_c).(\gamma + \Delta).exp(\gamma + \Delta + 1))$. □

As for BDH, this complexity is bounded by $O(exp(h))$

(the complexity of the method without good learning), with h the maximum number of variables in a path of a jointree T_c.

Like BTD$_{val}$, BDH$_{val}$ can be improved applying valued local consistency techniques. We can defined LC-BDH$_{val}$, an extension of BDH$_{val}$ with LC a valued local consistency technique. Even though, we obtain very good results in our experiments, using FC-BDH$_{val}$, we should do the same with LC-BDH$^{+}_{val}$. This method is similar to LC-BTD$^{+}_{val}$ [5] and the motivations are identical, i.e. to improve the pruning by using a better upper bound than $\top$ at the beginning of the search. Thus, we use in this method, as an upper bound, the difference between the valuation of the best solution so far and the local valuation of the current assignment as in [5]. In this case, recorded informations are not necessarily valued goods. It is possible that this upper bound be less than the optimal valuation of the sub-problem. So, LC-BTD$^{+}_{val}$ has not an optimal valuation, but a lower bound of this one. Nevertheless, this information is recorded, modifying the definition of structural valued good. A valued good is then defined by an assignment $\mathcal{A}[E_i \cap E_j]$ of an intersection between a hyperedge E_i and one of its sons E_j, and by a valuation which is a lower bound of the optimal valuation of the problem rooted in E_j or this optimal valuation. Finally, we have:

Theorem 5 *The time complexity of LC-BDH-val^{+} is $O(\alpha^*.(\gamma + \Delta).N(T_c).exp(\gamma + \Delta + 1))$, with $N(T_c)$ the number of jointrees T_c used by LC-BDH-val^{+} and α^* the optimal valuation of the VCSP.*

4 Variable ordering Heuristics based on CAH

4.1 Classes of variable orders

As for BDH, many variable orders can be used in BDH$_{val}$. We give below a hierarchy of classes of these orders more and more dynamic.

- **Class 1:** *Static variable orders compatible with a jointree T_c of H_A.*
  In this class, we use a single jointree T_c all along the search and computed at the beginning. Then, we derive a static compatible order on the hyperedges of H_A and on the variables in each hyperedge. The main drawback of this class is the inefficiency of the static variable orders.

- **Class 2:** *Static hyperedge orders compatible with a jointree T_c of H_A, dynamic variable ordering in each hyperedge.*
  Likewise in the Class 1, we have only one jointree T_c of H_A, computed statically. Hyperedge ordering is also static and compatible with T_c, while variable ordering in the hyperedges is dynamic.

- **Class 3:** *Dynamic hyperedge orders compatible with a jointree T_c of H_A, dynamic variable ordering in each hyperedge.*
  This class provides more freedom than the previous ones. The jointree is always computed statically and stays unchanged during the search. Yet, besides a dynamic variable ordering in each hyperedge, the order on hyperedges is also dynamic and compatible with the jointree.

- **Class 4:** *Class 3 orders on a jointree T_{cm} related to a jointree T_c of H_A.*
  The jointree T_c of H_A is always unique. However, we can modify the decomposition during the search by merging (covering) hyperedges in T_c and obtain, by so doing, a new jointree T_{cm} of a hypergraph in $\mathcal{CAH}_{H_A}[S]$. Finally, an order of this class is a *Class 3* order based on T_{cm}.

- **Class 5:** *Class 3 orders on a set of jointrees T_{cm} related to a jointree T_c of H_A.*
  The main difference with the previous class is that the hyperedge merging is dynamic. We have an unique jointree T_c and at each step of the search, we choose dynamically the next hyperedge (w.r.t. a compatible order) and we merge it possibly with some other hyperedges in the sub-problem rooted on it. Thus, we can use during the solving different jointrees T_{cm} (related to T_c) of hypergraphs in $\mathcal{CAH}_{H_A}[S]$. The orders of this class are *Class 3* orders based on this set of jointrees.

- **Class 6:** *Class 5 orders on a set of jointrees T_c of H_A.*
  This class allow using several jointrees T_c during the search.

- **Class ++:** *Dynamic variable ordering.*
  There is no restrictions on the variable orders. They are totally dynamic. This class offers no theoretical guarantees.

These classes forms an hierarchy :
Class 1 $\subset$ *Class 2* $\subset$ *Class 3* $\subset$ *Class 4* $\subset$ *Class 5* $\subset$ *Class 6* $\subset$ *Class ++*.

Theorem 6 *With a Class 1, Class 2 or Class 3 order, the time complexity of BDH$_{val}$ is $O(exp(\gamma))$.*

Theorem 7 *With a Class 4 order, the time complexity of BDH$_{val}$ is $O(exp(\gamma + \Delta))$, with $\gamma + \Delta$ the maximum size of hyperedges in T_{cm}.*

Theorem 8 *With a Class 5 order, the time complexity of BDH$_{val}$ is $O((\gamma + \Delta).exp(\gamma + \Delta + 1))$, with $\gamma + \Delta$ the maximum size of the hyperedges of the set of T_{cm} used.*

Theorem 9 *With a Class 6 order, the time complexity of BDH$_{val}$ is $O(N(T_c).(\gamma + \Delta).exp(\gamma + \Delta + 1))$, with $\gamma + \Delta$ the maximum size of the hyperedges of all the sets of T_{cm} used and $N(T_c)$ the number of jointrees T_c of H_A used.*

4.2 Heuristics

We present here some heuristics we use to run experiments presented in the following section. Since, these experiments are restricted to classes 1, 2, 3 and 4, we define heuristics for static merging of hyperedges, hyperedge ordering and variable ordering in the hyperedges.

4.2.1 Merging strategies

First, we define some static merging heuristics. They choose among the sons of the current hyperedge those that will be merged with it.

- sep: This heuristic has a parameter which defines the maximum size of the separators authorized. It merges every pair of hyperedges <father, son> whose intersection size is greater than this value.
- vp: Its parameter is the mimimum number of proper variables (variables not in the father hyperedge) that must contain a hyperedge. It merges with its father any hyperedge whose number of proper variables is less than this value.

4.2.2 Hyperedge ordering

Hyperedge ordering heuristics allow to choose efficiently the next hyperedge to consider among the sons of the current one. There is two steps in this ordering: the choice of the root hyperedge (the first to assign) and the ordering of the sons of a given hyperedge. While the order on the sons can be dynamic, the choice of the root is always static since it must be done before the search begins.

Root choice:

- $rand$: chooses randomly.
- $minexp$: chooses a hyperedge minimizing the ratio (exp) between its expected number of solutions and its size.
- $card$: chooses a hyperedge maximizing the number of variables.
- $size$: chooses a hyperedge maximizing the size (product of the variable domain sizes in hyperedges).
- $bary$: chooses barycentre hyperedge. The notion of barycentre uses one of distance between two hyperedges in the hypergraph which is the shortest path joining them. The barycentre hyperedge c is one minimizing $\Sigma_{c' \in C} dist(c, c')$.
- nv: chooses a hyperedge containing the first variable w.r.t. a dynamic variable ordering heuristic. We try, by this way, to derive benefit from the efficiency of these heuristics.

Static Son orders:

- $rand_s$: orders randomly.
- $minexp_s$: orders according to the increasing value of the ratio exp.
- $minsep_s$: orders according to the increasing value of the size of the intersection with the father hyperedge.

Dynamic Son orders:

- $rand_{sdyn}$: chooses randomly.
- nv_s: chooses a hyperedge containing the next variable w.r.t. a dynamic variable ordering heuristic, among the variables in the unassigned son hyperedges.
- $minexp_{sdyn}$: chooses a hyperedge minimizing the ratio exp which is computed after the father is totally assigned (it takes in account the induced filtering of domains).

4.2.3 Variable ordering

About the variable orders in the hyperedges, we use one of the best heuristics: minimum domain on degree (mdd) static (class 1) and dynamic (class 2, 3, 4) versions. This heuristic chooses as the next variable one minimizing the ratio between the size of its domain and its degree.

5 Experimental results

We run experiments with BDH$_{val}$ on benchmarks (structured random VCSPs) presented in [13]. The reference hypergraph is computed thanks to the triangulation of the constraint graph H performed in [14]. We present only the best results obtained by the heuristics given in this paper, using the same empirical protocol and PC as in [13]. The table shows the runtime of BDH$_{val}$ with the heuristic $card + minsep$ (class 1) and $minexp$ (classes 2,3 and 4),

VCSP (n,d,w,t,s,ns,p)	(a)	(b)	(c)	(d)
(75,10,15,30,5,8,10)	Mem	Mem	Mem	8.69
(75,10,15,30,5,8,20)	3.27	6.13	6.24	1.56
(75,10,15,33,3,8,10)	8.30	7.90	7.87	5.26
(75,10,15,34,3,8,20)	2.75	3.42	3.52	1.48
(75,10,10,40,3,10,10)	11.81	3.02	4.73	0.58
(75,10,10,42,3,10,20)	1.02	0.76	0.83	0.51
(75,15,10,102,3,10,10)	11.76	12.10	12.09	5.41
(100,5,15,13,5,10,10)	Mem	Mem	Mem	9.60

Table 1. Runtime (in s) on random partial structured VCSPs: (a) Class 1, (b) Class 2, (c) Class 3 and (d) Class 4.

with a hypergraph whose maximum size of hyperedge intersections is bounded by 5 for the Class 4. The Class 4 obtains the best results, it succeeds in solving all the instances while the other classes fails in solving a problem in the classes $(75, 10, 15, 30, 5, 8, 10)$ and $(100, 5, 15, 13, 5, 10, 10)$ because of a too large required memory space. Merging hyperedges with a too large intersection for the class 4 reduces the space complexity and increases the dynamicity of the variable ordering heuristic. This leads to significant improvements of the practical results of BDH$_{val}$ as well as those of the first version of BTD$_{val}$ which is equivalent to BDH$_{val}$ with an order of the Class 1.

6 Conclusion

In this paper, we have proposed an extension to VCSP (preferences and soft constraints) of the method BDH [14] defined in the CSP framework and based on coverings of a problem by acyclic hypergraphs. This approach gives more freedom to the variable ordering heuristic. We obtain a new theoretical time complexity bound in $O(N(T_c).(\gamma + \Delta).exp(\gamma + \Delta + 1))$. The dynamic exploitation of the problem structure induced by this method leads to very significant improvements. We must continue the experiments on the classes 5 and 6 for which more important enhancements are expected.

Acknowledgments This work is supported by an ANR grant (STAL-DEC-OPT project).

References

[1] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30:479–513, 1983.

[2] S. Bouveret, S. de Givry, F. Heras, J. Larrosa, E. Rollon, M. Sanchez, T. Schiex, G. Verfaillie, and M. Zytnicki. Max-CSP Competition 2006: toolbar/toulbar2 solver brief description. Technical report.

[3] D. Cohen, P. 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.

[4] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126:5–41, 2001.

[5] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proc. of AAAI*, pages 22–27, 2006.

[6] R. Dechter. Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[7] R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

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

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

[10] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[11] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proc of SODA*, pages 289–298, 2006.

[12] J. Huang and A. Darwiche. A structure-based variable ordering heuristic for SAT. In *Proc. of IJCAI*, pages 1167–1172, 2003.

[13] P. Jégou, S. Ndiaye, and C. Terrioux. Dynamic heuristics for branch and bound search on tree-decomposition of Weighted CSPs. In *Proc. of the International Workshop on Preferences and Soft Constraints (Soft-2006)*, pages 63–77, 2006.

[14] P. Jégou, S. Ndiaye, and C. Terrioux. 'Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs. In *Proc. of IJCAI*, pages 112–117, 2007.

[15] P. Jégou, S. Ndiaye, and C. Terrioux. Dynamic Management of Heuristics for Solving Structured CSPs. In *Proc. of 13th International Conference on Principles and Practice of Constraint Programming (CP-2007)*, pages 364–378, 2007.

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

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

[18] W. Li and P. van Beek. Guiding Real-World SAT Solving with Dynamic Hypergraph Separator Decomposition. In *Proc. of ICTAI*, pages 542–548, 2004.

[19] 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.

[20] I. Rish and R. Dechter. Resolution versus Search: Two Strategies for SAT. *Journal of Automated Reasoning*, 24:225–275, 2000.

[21] N. Robertson and P. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.

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

[23] C. Terrioux and P. Jégou. Bounded backtracking for the valued constraint satisfaction problems. In *Proc. of CP*, pages 709–723, 2003.