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

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 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, (Jégou, Ndiaye, & Terrioux 2007b) 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.

Introduction

Preference handling, when preferences can be expressed by constraints as with COPs or VCSPs, define hard problems from a theoretical viewpoint. So, algorithms to manage them must exploit all usable properties. For example, topological properties, ie. structural properties of datas 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 (Rish & Dechter 2000; Huang & Darwiche 2003; Li & van Beek 2004), in CSP (Dechter & Pearl 1989), in Bayesian or probabilistic networks (Dechter 1999; Darwiche 2001), in relational databases (Beeri *et al.* 1983; Gottlob, Leone, & Scarcello 2002), in constraint optimization (Terrioux & Jégou 2003; de Givry, Schiex, & Verfaillie 2006). 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 (Robertson & Seymour 1986) or a hypertree-decomposition (Gottlob, Leone, & Scarcello 2000) 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 (Jégou & Terrioux 2003) 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 (Gottlob, Leone, & Scarcello 2000). 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 (Cohen, Jeavons, & Gyssens 2005; Grohe & Marx 2006).

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 (Jégou & Terrioux 2003) or for manage preferences and soft constraints using Valued CSPs (Jégou & Terrioux 2004; Marinescu & Dechter 2006; de Givry, Schiex, & Verfaillie 2006). 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) (Bouveret *et al.*). 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 (Marinescu & Dechter 2006; Jégou, Ndiaye, & Terrioux 2007a; 2007b) notably to optimisation, preferences and soft constraints. Like in (Jégou, Ndiaye, & Terrioux 2007b), we prefer exploit here the more general and useful concept of covering by acyclic hypergraph rather than the 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 (Jégou, Ndiaye, & Terrioux 2007b), 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. First, this class is studied theoretically in order to determine its features. Then, 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 (Schiex, Fargier, & Verfaillie 1995), 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 and practical results before concluding.

Coverings by acyclic hypergraphs

The basic concept which interests us here is the acyclicity of networks. Often, it is expressed by considering the tree-decomposition or hypertree-decomposition, 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 (Beeri *et al.* 1983).

Definition 1 *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 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 (Dechter 2003)) is very close to one of tree-decomposition. Particularly, given a tree-decomposition we can easily compute a CAH. Moreover, the CAH-width α^* 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 α, while it can exist several tree-decompositions of width w s.t. $\alpha = w + 1$. The best structural methods for solving a COP with a CAH of width α have a time complexity in $O(exp(\alpha))$ 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 (Jégou, Ndiaye, & Terrioux 2007b), 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*. (Jégou, Ndiaye, & Terrioux 2007b) 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 2 *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 3 *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 $\{E_{i_1}, E_{i_2}, \dots E_{i_R}\}$. 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 4 *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 path in H between 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 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 neighbours* $\}$.
- $\mathcal{CAH}_{H_A}[U^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E_i \in E, \exists! E'_j \in E' : E_i \subset E'_j\}$.
- $\mathcal{CAH}_{H_A}[B^+] = \{(X, E') \in \mathcal{CAH}_{H_A} : \forall E'_i \in E', E'_i \subset E_{i_1} \cup E_{i_2} \cup \dots \cup E_{i_R}$ *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 $\forall E'_i \in E', E'_i = E_{i_1} \cup E_{i_2} \cup \dots \cup E_{i_R}$, 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 5 *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 1, 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 1 $\forall H'_A \in \mathcal{CAH}_{H_A}[S]$, $\exists \Delta \geq 0$ *such that* $\alpha' \leq \alpha + \Delta$ *and* $s' \leq s$.

Other classes are defined in (Jégou, Ndiaye, & Terrioux 2007b), 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.

Algorithmic Exploiting of the CAHs

In this section, we introduce the method BDH$_{val}$ which is an extension of BDH (Jégou, Ndiaye, & Terrioux 2007a) 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 H the constraint hypergraph of the given problem. This class allow 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. 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}.

$Father(E_i)$ 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.

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} = ∅ then
2      F ← Sons(E'_i); lb ← lb_{E'_i}
3      while F ≠ ∅ and lb ≺ ub_{E'_i} do
4          E_j ← Choose-hyperedge(F); F ← F\{E_j}
5          E'_{j'} ← Compute(T_{cm_b}, H_A, E_j)
6          if (A[E'_{j'} ∩ E'_i], o) is a good then lb ← lb ⊕ o
7          else
8              o ← BDH_val (A, E'_{j'}, E'_{j'}\(E'_{j'} ∩ E'_i), ⊤, ⊥, H_A,
               T_{cm_{b+1}})
9              lb ← lb ⊕ o; Record the good (A[E'_{j'} ∩ E'_i], o)
10     return lb
11 else
12     x ← Choose-var(V_{E'_i}); d_x ← D_x
13     while d_x ≠ ∅ and lb_{E'_i} ≺ ub_{E'_i} do
14         v ← Choose-val(d_x); d_x ← d_x\{v}
15         L ← {c ∈ E_{P,E'_i} | X_c = {x, y}, with y ∉ V_{E'_i}}; lb_v ← ⊥
16         while L ≠ ∅ and lb_{E'_i} ⊕ lb_v ≺ ub_{E'_i} do
17             Choose c ∈ L; L ← L\{c}
18             lb_v ← lb_v ⊕ c(A ∪ {x ← v})
19         if lb_{E'_i} ⊕ lb_v ≺ ub_{E'_i} then ub_{E'_i} ← min(ub_{E'_i}, BDH_val
           (A ∪ {x ← v}, E'_i, V_{E'_i}\{x}, ub_{E'_i}, lb_{E'_i} ⊕ lb_v, H_A, T_{cm_b})
20     return ub_{E'_i}
```

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 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 19). 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 9).

Theorem 2 *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, $\alpha' \leq \alpha + \Delta$. The value of Δ can be parametrized to only consider covering hypergraphs in $\mathcal{CAH}_{H_A}[S]$ whose width is bounded by $\alpha + \Delta$. 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 3 *The time complexity of BDH$_{val}$ is $O(N(T_c).(\alpha + \Delta).exp(\alpha + \Delta + 1))$, with $N(T_c)$ the number of jointrees used by BDH$_{val}$.*

Proof: Let $\mathcal{P} = (X, D, C, S)$ be a VCSP, H_A the CAH of reference of $H = (X, C)$.

Consider a tree-decomposition (E, T_c) of H the tree of which T_c has been built by BDH$_{val}$.

As for the proof for time complexity of BDH ((Jégou, Ndiaye, & Terrioux 2007b)), it's 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 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 ant none hyperedge being included in another, the separators are located only between two consecutive elements in the path.

During search, it's possible to cover V_a in different ways with the trees T_{cf} associated to T_c. Nevertheless, at least one separator of V_a will be an intersection between two clusters in each T_{cf}. Let T_{cf} be a tree associated to T_c such that s_1 be an intersection between two clusters. The search based on this tree will generate an assignment on V_a and record on s_1 a valued good. If s_1 is also an intersection between two clusters in a tree T'_{cf} associated 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 isn't an intersection, the location of the good can drive 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 clusters of a tree associated 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's 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 $number_{T_c}.number_{V_a}.(\gamma+\Delta).d^{\gamma+\Delta+1}$, 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 is then $O(number_{T_c}.(\gamma + \Delta).exp(\gamma + \Delta + 1))$. □

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 tree 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$^{+}_{val}$ (de Givry, Schiex, & Verfaillie 2006) 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 (de Givry, Schiex, & Verfaillie 2006). 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 cluster 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 4 *The Time complexity of LC-BDH-val^{+} is $O(\alpha^{*}.(\gamma+\Delta).number_{T_c}.exp(\gamma+\Delta+1))$, with $number_{T_c}$ the number of trees T_c used by LC-BDH-val^{+} and α^{*} the optimal valuation of the VCSP.*

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: The jointree is computed statically, no covering hypergraph is used and the hyperedge order is static as well as the variable order in each hyperedge.

Class 2: Like the Class 1 but the variable order is dynamic.

Class 3: Like the Class 1 excepted the hyperedge and variable orders which are dynamic.

Class 4: The jointree is always computed statically, one covering hypergraph is used and computed statically, the hyperedge and variable orders remain dynamic.

Class 5: Like Class 4 excepted the covering hypergraphs which are computed dynamically.

Class 6: The jointrees and the covering hypergraphs are computed dynamically and the hyperedge and variable orders are dynamic.

We run experiments with BDH$_{val}$ on benchmarks (structured random VCSPs) presented in (Jégou, Ndiaye, & Terrioux 2006). The reference hypergraph is computed thanks to the triangulation of the constraint graph H performed in (Jégou, Ndiaye, & Terrioux 2007a). We present only the best results obtained by the heuristics given in this paper, using the same empirical protocol and PC. The table shows the runtime of BDH$_{val}$ with the heuristic $card + minsep$ (class 1) and $minexp$ (classes 2,3 and 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 memory space required. 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 which is equivalent to BDH$_{val}$ with an order of the Class 1.

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.

Conclusion

In this paper, we have proposed an extension to VCSP (preferences and soft constraints) of the method BDH (Jégou, Ndiaye, & Terrioux 2007a) 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 $O(N(T_c).(\alpha + \Delta).exp(\alpha + \Delta + 1))$. 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.

References

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

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

Cohen, D.; Jeavons, P.; and Gyssens, M. 2005. A Unified Theory of Structural Tractability for Constraint Satisfaction and Spread Cut Decomposition. In *IJCAI'05*, 72–77.

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

de Givry, S.; Schiex, T.; and Verfaillie, G. 2006. Dcomposition arborescente et cohérence locale souple dans les CSP pondérés. In *Proceedings of JFPC'06*.

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

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

Dechter, R. 2003. *Constraint processing*. Morgan Kaufmann Publishers.

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

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

Grohe, M., and Marx, D. 2006. Constraint solving via fractional edge covers. In *SODA*, 289–298.

Huang, J., and Darwiche, A. 2003. A Structure-Based Variable Ordering Heuristic for SAT. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 1167–1172.

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

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

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

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

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

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

Marinescu, R., and Dechter, R. 2006. Dynamic Orderings for AND/OR Branch-and-Bound Search in Graphical Models. In *Proc. of ECAI*, 138–142.

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

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

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 631–637.

Terrioux, C., and Jégou, P. 2003. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of CP*, 709–723.