SOLVING NON BINARY CONSTRAINT SATISFACTION PROBLEMS WITH DUAL BACKTRACKING ON HYPERTREE DECOMPOSITION

Zineb Habbas, Kamal Amroun
LITA, University Paul-Verlaine of Metz, Metz, France
Department of Computer Sciences, University of Bejaia, Bejaia, Algeria

Daniel Singer
LITA, University Paul-Verlaine of Metz, Metz, France

Keywords: CSP, Dual backtracking, Forward-checking, Hypertree decomposition, Structural decomposition, Ordering Heuristics.

Abstract: Solving a CSP (Constraint Satisfaction Problem) is NP-Complete in general. However, there are various classes of CSPs that can be solved in polynomial time. Some of them can be identified by analyzing their structure. It is theoretically well established that a tree (or hypertree) structured CSP can be solved in a backtrack-free way leading to tractability. Different methods exist for converting CSPs in a tree (or hypertree) structured representation. Among these methods *Hypertree Decomposition* has been proved to be the most general one for non-binary CSPs. Unfortunately, in spite of its good theoretical bound, the unique algorithm for solving CSP from its hypertree structure is inefficient in practice due to its memory explosion. To overcome this problem, we propose in this paper a new approach exploiting a *Generalized Hypertree Decomposition*. We present the so called HD_DBT algorithm (Dual BackTracking algorithm guided by an order induced by a generalized Hypertree Decomposition). Different heuristics and implementations are presented showing its practical interest.

1 INTRODUCTION

Many important real world problems can be formulated as Constraint Satisfaction Problems (CSPs). The most usual complete method for solving CSPs is based on a backtracking search. This approach has an exponential worst-case time complexity of $O(m.d^n)$ for a CSP instance with m the number of constraints, n the number of variables and d the largest size of variable domains. Solving a CSP is NP-Complete in general but there are various classes of CSPs that can be solved in polynomial time. Freuder (Freuder, 1982) proved that a tree-structured CSP can be solved efficiently and many efforts have been made to define tree-like decomposition methods that lead to tractability. Methods deriving from the Database area have been successfully used to characterize some new tractable classes (Jeavons et al., 1994; Gyssens et al., 1994; Gottlob et al., 2000; Gottlob et al., 2005). Their main common principle is to decompose the CSP into a number of subproblems organized in a tree or an hypertree structure. These subproblems are then solved independently and the solutions are propagated in a backtrack-free manner to solve the initial CSP as described in (Dechter and Pearl, 1989). Numerous other decomposition methods have been proposed, to cite some of the most important ones: biconnected components (Freuder, 1982), hinge decomposition combined with tree clustering (Gyssens et al., 1994) and spread-cut decomposition (David Cohen, 2005). A more recent work (Greco and Scarcello, 2010) proposed a deep study on different versions of structural decompositions deriving from binary representations of general (non-binary) CSPs. This study gives a theoretical comparison of structural decompositions on binary representations with direct non-binary ones. All these methods are characterized by their computational complexity in terms of the tree (resp. hypertree) width they generate. Among them Gottlob et al. (Gottlob et al., 2000) have shown the hypertree

Habbas Z., Amroun K. and Singer D..
SOLVING NON BINARY CONSTRAINT SATISFACTION PROBLEMS WITH DUAL BACKTRACKING ON HYPERTREE DECOMPOSITION.
DOI: 10.5220/0003184801460156
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 146-156
ISBN: 978-989-8425-40-9

decomposition dominates all the other structural decomposition methods excepted the methods recently introduced in (Grohe and Marx, 2006).

Most of works on structural decompositions are purely theoretical to define new general tractable classes. However, the main drawback of these approaches from a practical viewpoint is the memory space explosion due to the expression of relations associated to constraints by tables and the storage of all the solutions resulting from join operations as well. The representation of relations by tables is adequate for the Database processing using auxiliary memory disks to save all the relations, but is not realistic for solving a CSP which requires to save all the structure in RAM. This is the main reason why the first basic algorithm (Gottlob et al., 2001) proposed in the literature to solve CSP with an hypertree is inefficient in practice. To use more efficiently the structural decomposition in the search algorithm, the *join* and *semi-join operations* have to be discarded. This idea has been introduced by different researchers.

In (Pang and Goodwin, 1997; Pang and Goodwin, 2003) Pang et al. have proposed a *w-CBDT* algorithm combining both merits of Constraint-Directed Backtracking and structural decompositions, but the authors do not consider optimal structural decompositions. Jegou et al. (Jégou and Terrioux, 2003) have introduced the successfull method called BTD (Backtracking with Tree Decomposition) which is an enumerative search algorithm guided by some pre-established order induced by a tree decomposition. This paper goes forward in the same direction to solve non-binary CSPs directly using efficiently hypertree decomposition.

We propose an alternative approach to BTD called HD_DBT for Dual BackTracking algorithm guided by an order induced by an Hypertree Decomposition. The main idea of this approach is that search will be guided for the choice of the partial solution by the hypertree structure. HD_DBT is more adapted to solve non-binary CSPs represented as hypergraphs. Given a CSP and its hypergraph representation, HD_DBT computes one Hypertree Decomposition and it looks for a solution by using a dual backtracking search algorithm. We propose and compare different heuristics to achieve “the best” depth-first traversal of the hypertree decomposition wrt. CSP solving complexity. The hypertree decomposition properties make the generalisation of BTD to HD_DBT non obvious. Informally, HD_DBT is guided by an order on the *clusters of constraints* and not on the clusters of variables while (as BTD) the connectivity property concerns the clusters of variables. Moreover BTD is based on a tree decomposition which is complete while HD_DBT is based on hypertree decomposition which is not. Thus before solving a CSP using an hypertree decomposition we have to complete the resulting hypertree.

The paper is organized as follows: section 2 gives the preliminary and necessary notions on CSP and decomposition methods with a special focus on the hypertree decomposition which is the most general one. Section 3 presents the basic algorithm exploiting an hypertree decomposition to solve CSPs. Section 4 presents our new HD_DBT approach. Section 5 gives the first experimental results of HD_DBT compared with the basic algorithm proposed in (Gottlob et al., 2001). Then we present an improved HD_DBT+FC implementation together with different heuristics guiding the resolution. Finally, Section 6 gives a conclusion.

2 PRELIMINARIES

In this section we recall the basic definitions of constraint satisfaction problems, hypergraphs, hypertree and generalized hypertree decompositions.

2.1 Constraint Satisfaction Problems

The notion of Constraint Satisfaction Problems (CSP) was formally defined by U. Montanari (Montanari, 1974).

Definition 1 (Constraint Satisfaction Problem)**.** *A CSP is defined as a triple $P = < X, D, C >$ where :*
$X = \{x_1, x_2, ..., x_n\}$ is a set of n variables.
$D = \{d_1, d_2, ..., d_n\}$ is a set of finite domains; each variable x_i takes its value in its domain d_i.
$C = \{c_1, c_2, ..., c_m\}$ is a set of m constraints. Each constraint c_i is a pair $(S(c_i), R(c_i))$ where $S(c_i) \subseteq X$, is a subset of variables, called the scope of c_i and $R(c_i) \subseteq \prod_{x_k \in S(c_i)} d_k$ is the constraint relation, that specifies the legal combinations of values.

A solution to a CSP is an assignment of values to all the variables such that all the constraints are satisfied. Solving a CSP means to find a solution if it exists. Binary CSPs are those defined where each constraint involves only two variables $\forall i \in 1 \ldots m : |S(c_i)| = 2$.

In order to study the structural properties of a CSP, we need to present the following definitions. For more detailed descriptions see eg. (Dechter, 2003), (Gottlob et al., 2001) and (Gottlob et al., 2002).

Definition 2 (Hypergraph)**.** *A hypergraph is a structure $\mathcal{H} =< V,E >$ that consists of a set of vertices V and a set of hyperedges E where each hyperedge $h \in E$ is a subset of vertices of V. The hyperedges differ from edges of graphs in that they may connect more than two vertices.*

The structure of a CSP $P =< X,D,C >$ is entierely captured by its associated hypergraph $\mathcal{H} =< V,E >$ where the set of vertices V is the set of variables X and the set of hyperedges E corresponds to the set of constraints C. For any subset of hyperedges $K \subseteq E$ let be $vars(K) = \bigcup_{e \in K} e$ the set of the variables occuring in the edges of K. For any subset $L \subseteq V$ let be $edgevars(L) = vars(\{e | e \in E, e \cap L \neq \emptyset\})$ the set of all variables occuring in any edge intersecting L.

Definition 3 (Hypertree)**.** *Let $\mathcal{H} =< V,E >$ be a hypergraph. A hypertree for hypergraph $\mathcal{H}$ is a triple $< T,\chi,\lambda >$ where $T = (N,B)$ (the sets of nodes and branche-edges) is a rooted tree and χ and λ are two labelling functions on nodes of T. The functions χ and λ map each node $p \in N$ on two sets $\chi(p) \subseteq V$ and $\lambda(p) \subseteq E$.*

A tree is a pair $< T,\chi >$ where $T = (N,B)$ is a rooted tree and χ is a labelling function as previously defined. T_p denotes the subtree of T rooted at node p.

Definition 4 (Hypertree decomposition)**.** *A hypertree decomposition of a hypergraph $\mathcal{H} =< V,E >$, is a hypertree $< T,\chi,\lambda >$ which satisfies the following conditions:*

1. *For each (hyper)edge $h \in E$, there exists $p \in N$ such that $vars(h) \subseteq \chi(p)$. We say that p covers h.*
2. *For each vertex $v \in V$, the set $\{p \in N | v \in \chi(p)\}$ induces a (connected) subtree of T.*
3. *For each node $p \in N, \chi(p) \subseteq vars(\lambda(p))$.*
4. *For each node $p \in N$, $vars(\lambda(p)) \bigcap \chi(T_p) \subseteq \chi(p)$*

The width of a hypertree decomposition $< T,\chi,\lambda >$ is $max_{p \in N} |\lambda(p)|$. The hypertree-width $htw(\mathcal{H})$ of a hypergraph $\mathcal{H}$ is the minimum width over all its possible hypertree decompositions.

Definition 5 (Generalized hypertree decomposition)**.** *A generalized hypertree decomposition is a hypertree which satisfies the first three conditions of the hypertree decomposition (see Definition 4).*

The width of a generalized hypertree decomposition $< T,\chi,\lambda >$ is $max_{p \in N} |\lambda(p)|$. The generalized hypertree-width $ghw(\mathcal{H})$ of a hypergraph $\mathcal{H}$ is the minimum width over all its possible generalized hypertree decompositions.

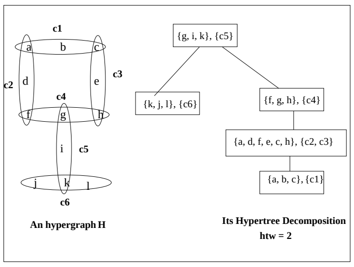

Figure 1: A hypergraph and one of its hypertree decomposition.

Definition 6. *A hyperedge h of a hypergraph $\mathcal{H} =< V,E >$ is strongly covered in $HD =< T,\chi,\lambda >$ if there exists $p \in N$ such that all the vertices in h are contained in $\chi(p)$ and $h \in \lambda(p)$.*

Definition 7. *A hypertree decomposition $< T,\chi,\lambda >$ of a hypergraph $\mathcal{H} =< V,E >$ is a complete hypertree decomposition if every hyperedge h of $\mathcal{H} =< V,E >$ is strongly covered in $HD =< T,\chi,\lambda >$.*

2.2 Computing an Hypertree Decomposition

In this section we briefly present the two main approaches proposed in the literature to compute hypertree decompositions: the exact methods and the heuristics ones.

2.2.1 Exact Methods

Given a hypergraph $\mathcal{H} =< V,E >$, exact algorithms aim at finding a hypertree decomposition with width w less than or equal to a constant k, if such a decomposition exists. The first exact algorithm named *opt-k-decomp* for the generation of an optimal hypertree decomposition is due to Gottlob et al. (Gottlob et al., 1999). This algorithm builds a hypertree decomposition in two steps: it finds if a hypergraph $\mathcal{H} =< V,E >$ has a hypertree decomposition $HD =< T,\chi,\lambda >$ with width less than or equal to a constant k. If it is the case it finds a hypertree decomposition of smallest possible width. The algorithm *opt-k-decomp* runs in $O(m^{2k}V^2)$ where m is the number of hyperedges, V is the number of vertices and k is a constant.

Among the number of improvements to *opt-k-decomp* we can cite *Red-k-decomp* (Harvey and

Ghose, 2003) and the Subbarayan and Anderson algorithm (Sathiamoorthy and Andersen, 2007) which is a backtracking version of *opt-k-decomp*. However these exact methods have an important drawback which is the huge amount of memory space needed and the bad running time resulting in inefficiency in practice for large instances. To overcome these limitations some heuristics have been proposed to compute hypertree decompositions.

2.2.2 Heuristics

Heuristics aim at finding a hypertree decomposition with the smallest possible width (tree-width) but without any theoretical guaranty to succeed. There are several heuristics to compute a hypertree decomposition. Korimort (Korimort, 2003) proposed one heuristics based on the vertices connectivity of the given hypergraph. Samer (Samer, 2005) explored the use of branch decomposition for constructing hypertree decompositions. Dermaku et al. (Dermaku et al., 2005) proposed the following heuristics: BE (Bucket Elimination), DBE (Dual Bucket Elimination) and Hypergraph partitioning. Musliu and Schafhauser (Musliu and Schafhauser, 2005) explored the use of genetic algorithms for generalized hypertree decompositions.
We outline the two successful BE and DBE heuristics for generating hypertree decompositions.

The Bucket Elimination (BE) Heuristics (Dechter, 1999) was successfully used to compute a tree decomposition of a given graph (or a primal graph of a hypergraph). BE has been extended by Dermaku et al. (Dermaku et al., 2005) to compute a hypertree decomposition or more precisely a generalized hypertree decomposition. The simple idea behind this extension derives from the fact that a generalized hypertree decomposition satisfies the properties of a tree decomposition. Consequently, for computing a generalized hypertree decomposition, BE proceeds as follows. First, it builds a tree decomposition using (basic) BE. Then, it creates the $\lambda - labels$ for each node of this tree in order to satisfy the third condition of generalized hypertree decomposition according to the *Definition* 5. This is done greedily by attempting to cover the variables of each node by hyperedges. In practice the BE heuristics requires a good vertices ordering to be efficient.

The Dual Bucket Elimination (DBE) Heuristics was proposed by Dermaku et al. (Dermaku et al., 2005). DBE simply applies the BE heuristics on the dual graph of the hypergraph. The idea behind using the dual graph structure instead of the primal graph is that BE minimizes the $\chi - labels$ while the width of a hypertree decomposition is determined by the $\lambda - labels$. This is exactly what is done when applying BE to the dual graph of the hypergraph.

3 SOLVING CSP USING AN HYPERTREE DECOMPOSITION

To solve any CSP using a generalized hypertree decomposition, the first step consists in computing the generalized hypertree decomposition either with an exact or with an heuristic method. The next step transforms the generalized hypertree decomposition into a complete one in order to cover each constraint by at least one node of the hypertree. Consequently the complete generalized hypertree decomposition may be seen as a join tree of an equivalent (wrt. its solutions) acyclic CSP. Each node of the join tree represents a subproblem of the new acyclic CSP. The third step of the resolution is described in algorithm 1 due to Gottlob et al. (Gottlob et al., 2001). In algorithm 1, each subproblem is solved independently and it can be done by a parallel algorithm. The Acyclic solving algorithm is used for finding a complete consistent solution of the initial CSP.

Algoritm 1: Gottlob Algorithm (Gottlob et al., 2001).

```
Input: a complete generalized hypertree
       decomposition $< T,\chi,\lambda >$ associated to a
       given CSP.
Output: a solution $\mathcal{A}$ of the CSP if it is satisfiable
begin
    $\sigma = \{n_1, n_2, \dots, n_m\}$ a node ordering with $n_1$ the
    root of the hypertree and each node precedes all
    its sons in $\sigma$;
    foreach p a node in $\sigma$ do
        $R_p = (\bowtie_{C_j \in \lambda(p)} R_j)[\chi(p)]$ ;
    end
    for i = m to 2 do
        Let $v_j$ the father of $v_i$ in $\sigma$ ;
        $R_j = R_j \propto R_i$ ;
    end
    for i = 2 to m do
        Build a solution $\mathcal{A}$ by choosing a tuple $R_i$
        compatible with all the previous assignments
    end
    return $\mathcal{A}$ ;
end
```

Although this algorithm is theoretically interesting, its practical interest has unfortunately not yet been proved. The main drawback of this algorithm is its space complexity. Indeed a lot of memory is needed to save the intermediate results of join and semi-join operations. Saving all the intermediate results is useful only when we are looking for all the solutions. Therefore if we look for a single solution, this approach proves to be inefficient and much memory consuming. In this case, an enumerative approach should be more appropriate. To take into account both the advantages of decomposition methods and enumerative search techniques Jégou et al. proposed in (Jégou and Terrioux, 2003) an original and successful method called BTD (BackTrack algorithm guided by an order induced from a Tree Decomposition). In this work we propose a new approach deriving from the BTD idea to use a generalized hypertree decomposition. The new algorithm called HD_DBT (Dual Backtracking Algorithm guided by an order induced from a Hypertree Decomposition) is detailed in next Section.

4 THE NEW METHOD HD_DBT

4.1 Informal Presentation

We mentioned in the previous section the main drawback of the basic algorithm proposed by Gottlob et al. (Gottlob et al., 2001) for solving CSP using an hypertree decomposition. The memory space explosion due to the join operations (on partial solutions) are unnecessary when only one solution is required. In this work, we propose a new approach for solving CSP exploiting the properties of Generalized hypertree decomposition together with the advantages of the enumerative search algorithms. Our approach called HD_DBT for Dual BackTrack using a Generalized Hypertree Decomposition, is an enumerative search algorithm guided by a partial order on the clusters of constraints derived from the Generalized hypertree decomposition. HD_DBT is called “Dual” because it works directly on tuples in the relations. In other words, HD_DBT looks for a solution by assigning simultaneously a set of variables instead of one single variable like in the classical algorithms. The assignment of this set of variables represents a partial solution of the whole problem and corresponds to a solution of one node of the hypertree. A solution of the CSP by HD_DBT can be expressed as $(\bowtie_{i=1\dots N} s_i)[x_1x_2\dots x_n]$ where the symbol $\bowtie$ corresponds to the join operator, n is the number of variables of the CSP, N is the number of nodes of the hypertree and s_i is a solution of a given node N_i. Clearly, the performance of HD_DBT is highly dependent on the number of nodes and on the order in which the nodes are explored, particularly the choice of the first root node.

4.2 Formal Presentation of HD_DBT

The HD_DBT approach is formally described by algorithm 2. It considers as input a complete hypertree decomposition according to Definition 7 and it consists of the following steps:

Algoritm 2: Generic Procedure HD_DBT.

```
Input: a complete hypertree decomposition
        $\mathcal{HD} = < T, \chi, \lambda >$ associated to a given CSP.
Output: a solution $\mathcal{A}$ of the CSP if it is satisfiable
begin
    Choose_Root ( $\mathcal{HD}$ , root ) ;
    $\sigma \longleftarrow$ Induced_Order ( $\mathcal{HD}$ , root ) ;
    cn $\longleftarrow$ root
    while $cn \neq \emptyset$ do
        consistent $\longleftarrow$ FALSE ;
        while $\neg$ consistent do
            cs $\leftarrow$ Resolution ( $\lambda$ (cn), $\chi$(cn));
            if Compatible(cs,sol(father(cn))) then
                $\mathcal{A} \leftarrow \mathcal{A} \cup \{x_i \leftarrow v_i : \forall x_i \in \chi(cn)\}$;
                consistent $\leftarrow$ TRUE ;
            end
        end
        if $\neg$ consistent then
            $\mathcal{A} \leftarrow \mathcal{A} - \{x_i \leftarrow v_i : \forall x_i \in \chi(cn)\}$ ;
            cn $\leftarrow$ $father$(cn) ;
        end
        else
            cn $\leftarrow$ $succ$(cn) ;
        end
    end
    return $\mathcal{A}$ ;
end
```

Step1 (Choice of the Root): the choice of the root made by the procedure **Choose_Root** is crucial for the performance of HD_DBT. As the resulting hypertree decomposition of a CSP is not a rooted tree, any node could be choosen as root. The hypertree decomposition quality depends both on the choice of the root and on the induced nodes ordering to be visited. In section 5 we present different heuristics for choosing the root.
Figure 2 shows two different hypertree decompositions (nodes orderings) of the same hypergraph given in Figure 1 induced by different choices of the root. The left one is the hypertree given by the BE heuristic with node n_1 as root and the the right one considers arbitrarily the node n_4 as root.

Step2 (The Nodes Order): the choice of the root node induces a partial ordering σ on the other nodes. This ordering built from the hypertree decomposition and the root using the procedure **Induced_Order** respects the connectivity

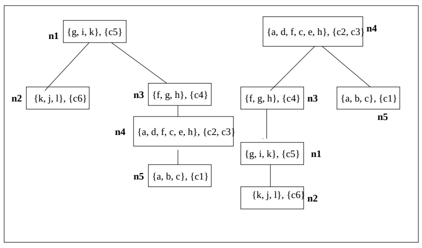

Figure 2: Two different hypertree decompositions (nodes orderings) for a same hypergraph.

property of the hypertree decomposition (*Definition* 4). For both hypertree decompositions given by Figure 2, we can associate the following depth-first orderings: $\sigma_1 = n_1 n_2 n_3 n_4 n_5$ for the left and $\sigma_2 = n_4 n_3 n_1 n_2 n_5$ for the right hypertrees.

Step3 (Looking for One Solution): this step is an enumerative search algorithm guided by the ordering σ. For each current node cn with label $(\lambda(cn), \chi(cn))$ in σ, HD_DBT looks for a partial solution by calling the procedure **Resolution**. The basic HD_DBT algorithm is the generic one using the generic BackTrack algorithm for the resolution but it can be easily generalized to FC, MAC, etc.

Proposition 1. *HD_DBT is correct, complete and it terminates .*

The proof is straightforward.

Proposition 2. *The worst time complexity of HD_DBT is in $O(|r|^{w \times N})$ where r is the size of the largest relation, w is the hypertree width and N is the number of nodes in σ.*

Proof. In the worst case HD_DBT visits all the N nodes of the tree and at each node it checks in worst case $O(|r|^w)$ tuples. □

Proposition 3. *The space complexity of HD_DBT is in $O(N)$ where N is the number of nodes of the hypertree.*

Proof. Let t be the size of the largest scope, w the width of the hypertree decomposition and N the number of the nodes of the hypertree. Then we have to save in the worst case the number $t * w * N$ values leading to a linear complexity wrt. N.

□

4.3 How to Complete HTD

As already mentioned in the description of HD_DBT, the resulting hypertree decomposition obtained by BE or any other method is not necessarily complete. Before solving the CSP, the first step consists in completing the hypertree. In (Gottlob et al., 2001), Gottlob et al. proposed a procedure to complete one hypertree decomposition by adding for each not strongly covered constraint c_i a new node with label $\{c_i\}, \{var(c_i)\}$ as a son of a node n_j verifying the condition $var(c_i) \subseteq \chi(n_j)$. The idea behind this procedure is to not increase the hypertree width while the number of nodes of the hypertree obviously increases. We experimented another way to complete the hypertree decomposition. Instead of adding a new node, we add the constraint c_i not strongly covered in the λ of the node n_j satisfying the condition $var(c_i) \subseteq \chi(n_j)$. This makes the hypertree width increases, while the number of nodes of the hypertree remains the same.

Let be a CSP defined as the following set of constraints $C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$. Figure 3(a) shows one of its hypertree decompositions. The variables are deliberately omitted here because they are not useful for this example. This hypertree decomposition is not complete because the constraints c_4 and c_7 are not strongly covered. Figure 3(b) corresponds to its completion by using the procedure of Gottlob. Two new nodes n_5 and n_6 are created for strongly covering the constraints c_4 and c_7 respectively. Figure 3(c) illustrates its completion by using our procedure. No new node has been created but the nodes n_1 and n_3 are modified. In the node n_1 we add the constraint c_4 to its λ and in the node n_3 we add the constraint c_7 to its λ. We have obviously assumed that the variables of the non strongly covered constraints c_4 and c_7 are present in the $\chi(n_1)$ and $\chi(n_3)$ respectively. Figure 3(d) is another way to treat the non-covered constraints. As we can observe, we keep the incomplete hypertree decomposition in one hand and in another hand we consider a cluster of all non-covered constraints. This third idea has not been explored in this work.

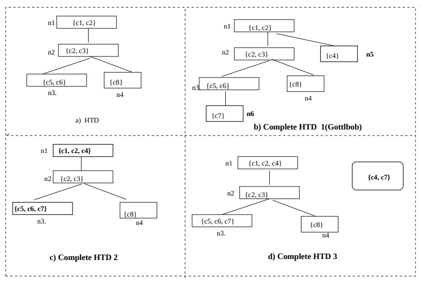

Figure 3: Different ways to complete a hypertree decomposition HTD.

5 EXPERIMENTAL RESULTS

In this section, we present and analyze some of the experiments we perform to validate our approach from a practical point of view.

5.1 The Experimental Considerations

We implemented the different versions of our approach using C++ language. The experiments were performed on a 1,7 GHZ PC with 2 GO of RAM running under Linux Fedora. Our tests have been executed on benchmarks downloaded from the following URL Benchmarks site[1]. We used the BE heuristics (Dermaku et al., 2005) to compute the generalized hypertree decomposition of any CSP. BE is well known to be the best one giving a nearly optimal generalized hypertree decomposition within a reasonable CPU time. In the sequel, we experiment and compare our approach in different ways:

- subsection 5.2 compares HD_DBT with the basic and unique resolution Algorithm 1 due to Gottlob et al. (Gottlob et al., 2001) using an hypertree decomposition (see Section 3).
- In subsection 5.3, we compare HD_DBT with (HD_DBT + FC) algorithm which is the Forward Checking version of HD_DBT, in order to measure the gain offered by the filtering operations.
- In subsection 5.4, we study the behavior of different heuristics for choosing the root.
- In subsection 5.5, we study different heuristics for choosing the next son node to be visited.
- Finally in subsection 5.6, we compare HD_DBT with BTD (Jégou and Terrioux, 2003).

In all the experiments, the resolution time includes the time for building the generalized hypertree decomposition using the BE heuristic.

5.2 Comparing HD_DBT with Gottlob et al. 2001 Algorithm

Table 1 presents the results of HD_DBT compared with the first proposed algorithm to solve CSP using an hypertree decomposition. HD_DBT outperforms the basic approach of Gottlob (Gottlob et al., 2001) for all the considered benchmarks (excepted $hanoi-6-ext$). The row Got. corresponds to the CPU time obtained with Gottlob algorithm. Unfortunately the naive preliminary version of our approach was also limited when large instances of CSP were considered.

[1] http://www.cril.univ-artois.fr/lecoutre/research/benchmarks

Table 1: HD_DBT vs. Gottlob approach.

Problem	Size		Time (in seconds)	
	$\|V\|$	$\|E\|$	HD_DBT	Got.
$Renault1$	101	134	2	3
$series-6-ext$	11	30	0,04	2,18
$series-7-ext$	12	41	0,1	/
$domino-100-100-ext$	100	100	0,12	2,59
$domino-100-200-ext$	100	100	0,30	18,37
$domino-100-300-ext$	100	100	0,4211	60
$hanoi-5-ext$	30	29	0,55	0,88
$hanoi-6-ext$	62	61	120	14
$hanoi-7-ext$	126	125	58	59
$Langford$	8	32	0,20	2,52
$geom-30a-4-ext$	30	81	0,1	0,1
$pigeons-7-ext$	7	21	2	26

5.3 Comparing HD_DBT with HD_DBT + FC

To improve the HD_DBT approach, we implemented the algorithm called (HD_DBT + FC). As the classical FC algorithm does, this algorithm consists in adding a filtering step at each node of the hypertree. When we are cheking a tuple t, solution of a given subproblem at node i, for each descendant node j of i we remove from each relation R_k corresponding to a constraint C_k (where $C_k \in \lambda(j)$) the tuples inconsistent with t. This filtering step is crucial thanks to the connectivity property of the hypertree decomposition.

Table 2 shows the gain obtained by (HD_DBT + FC) over HD_DBT. Clearly the filtering always improves considerably HD_DBT. Moreover HD_DBT + FC can find a solution when HD_DBT fails. Thus in the following experiments we will consider only this improved version.

Table 2: HD_DBT vs. HD_DBT + FC.

Problems	Size		Time(s)	
	$\|V\|$	$\|E\|$	HD_DBT	HD_DBT + FC
$series-6-ext$	11	30	1,54	0,09
$series-7-ext$	12	41	2,02	0,08
$domino-100-100-ext$	100	100	0,20	0,125
$domino-100-200-ext$	100	100	5,90	0,24
$domino-100-300-ext$	100	100	12,77	0,35
$langford-2-4-ext$	8	32	0,54	0,03
$geom-30a-4-ext$	30	81	>20	0,03
$pigeons-7-ext$	7	21	>20	4,34
$haystacks-06-ext$	36	96	>20	3,33
$Renault1$	101	134	3,40	3,38
$Renault2$	101	113	/	8,87
$Renault-Modified-6$	111	147	/	**32,07**
$Renault-Modified-24$	111	159	/	**21,25**
$Renault-Modified-30$	111	154	/	**24,28**
$Renault-Modified-33$	111	154	/	**25,42**
$Renault-Modified-47$	108	149	/	**19,64**

Notice here that we implemented two methods to complete the hypertree decomposition (the Gottlob approach and our method). Not surprinsingly, we observed that our approach of completion is better than Gottlob's one when we consider the HD_DBT algorithm (without filtering), but when we used (HD_DBT +FC) the two approaches of completion are equivalent.

5.4 Heuristics to Choose the Root Node

To improve further the performance of HD_DBT we introduce different heuristics based on the way the nodes of the hypertree are explored. The nodes are traversed in a depth-first order from one given root. But the choice of the root is not unique, and a rational choice of the root may considerably improve the overall performance of the resolution. For choosing the root node, the three following methods may be distinguished: structural, semantic and hybrid heuristics.

Structural Heuristics depend on the size of the clusters at each node. We will consider the following structural heuristics:

- LC (Largest Cluster): the node with the largest number of constraints.
- SC (Smallest Cluster): the node with the smallest number of constraints.
- LNS (Largest Number of Sons): the node with the largest number of sons.
- LS (Largest Separator): the node with the largest separator with one of its sons.

Semantic Heuristics exploit the data properties (the size of relations, the density of nodes, the hardness of constraints ...). Here we consider that we have already computed the number of solutions of any node i denoted by $Nbsol(i)$. This step is not expensive thanks to the small number of variables at each node. We will consider the following semantic heuristics:

- MCN (Most Constrained Node): the node with the smallest number of solutions.
- LCN (Least Constrained Node): the node with the largest number of solutions.
- HN (Hardest Node): the "hardest node". Hardness of a node corresponds to $\frac{Nbsol}{NbMax}$ where Nbsol is the exact number of solutions and NbMax is the number of possible tuples of a node (cartesian product of the variables domains).

Hybrid Heuristics combine in different ways structural and semantic heuristics eg. HN & LNS , LG & HN , HN & LC heuristics.

Table 3 shows the results for the different structural heuristics for the choice of the root. Recall this choice induces an order σ on the nodes and the row BE corresponds to the order induced by BE. We remark that the best orders (for the hardest problems of the test) are the one returned by BE and SC.

Table 3: Comparing different structural orders.

Problems	Order				
	BE	LC	SC	LNS	LS
$series-6-ext$	0,09	0,07	0,43	0,07	0,12
$series-7-ext$	0,08	3,48	3,18	3,5	0,96
$domino-100-100-ext$	0,125	0,14	0,49	0,13	0,22
$domino-100-200-ext$	0,24	0,23	0,27	0,23	0,28
$domino-100-300-ext$	0,35	0,36	0,69	0,34	0,51
$Langford$	0,03	0,31	0,91	8,73	1,02
$geom-30a-4-ext$	0,03	0,03	5	0,23	1,02
$pigeons-7-ext$	4,34	4,19	3,5	12	4,23
$haystacks-06-ext$	3,33	3,35	3,33	3,22	3,21
$Renault1$	3,38	3,02	4,41	3,03	3,31
$Renault2$	8,87	9,02	8,88	/	/
$Renault-Modified-6$	32,07	/	36,09	/	/
$Renault-Modified-24$	21,25	/	18,32	/	/
$Renault-Modified-30$	24,28	/	22,05	/	/
$Renault-Modified-33$	25,42	/	26,01	/	/
$Renault-Modified-47$	19,64	/	21,02	/	/

Table 4 shows the results for the different semantic heuristics for the choice of the root. We remark that the better choice is given by the MCN heuristic.

Table 4: Comparing different semantic orders.

Problems	Size		Orders		
	$\|V\|$	$\|E\|$	MCN	LCS	HN
$series-6-ext$	11	30	0,10	0,38	0,43
$series-7-ext$	12	41	0,55	3,48	0,13
$domino-100-100-ext$	100	100	0,12	0,70	0,49
$domino-100-200-ext$	100	100	0,23	2,32	0,76
$domino-100-300-ext$	100	100	0,36	5,14	0,75
$Langford$	8	32	0,14	0,36	0,09
$geom-30a-4-ext$	30	81	7	1,23	0,03
$pigeons-7-ext$	7	21	11	4,23	4,2
$haystacks-06\_xt$	36	96	6,96	3,32	3,31
$Renault1$	101	134	3,03	3,02	3,04
$Renault2$	101	113	7,90	8,36	8,34
$Renault-Modified-6$	111	147	22,07	/	/
$Renault-Modified-24$	111	159	19,04	/	/
$Renault-Modified-30$	111	154	28,02	/	/
$Renault-Modified-33$	111	154	27,03	/	/
$Renault-Modified-47$	108	149	12,02	/	/

Finally, we combine the best structural heuristic with the best semantic one resulting in BE with MCN hybrid heuristic. The results are given in Table 5 showing that execution times are always better with an hybridation of heuristics.

5.5 Heuristics for Choosing Son Nodes

In this section, we experiment different heuristics to choose the best successor node (function *Succ* of Algorithm 2) to be solved from the sons of the current node. We suppose here the root has been already chosen and heuristics have only the choice for the first son, and it may be considered statically or dynamically.

- **Static Choice:** the first son to be chosen is the one with the largest hypertree-width.
- **Dynamic Choice:** since the best strategy from the static viewpoint is the one with the minimum of number of tuples in its relations, we evaluate here the impact of choosing the next son with the minimum number of tuples dynamically. Thus to be

Table 5: Combining structural and semantic heuristics.

Problems	Size		Method
	$\|V\|$	$\|E\|$	BE & MCN
$series-6-ext$	11	30	0,07
$series-7-ext$	12	41	0,06
$domino-100-100-ext$	100	100	0,10
$domino-100-200-ext$	100	100	0,20
$domino-100-300-ext$	100	100	0,30
$Langford$	8	32	0,04
$geom-30a-4-ext$	30	81	0,04
$pigeons-7-ext$	7	21	3,34
$haystacks-06-ext$	36	96	2,59
$Renault1$	101	134	3,38
$Renault2$	101	113	8,87
$Renault-Modified-6$	111	147	22,07
$Renault-Modified-24$	111	159	15,2
$Renault-Modified-30$	111	154	18,28
$Renault-Modified-33$	111	154	19,42
$Renault-Modified-47$	108	149	16,64

Table 7: Dynamic choice of the sons.

Problems	Size		Method
	$\|V\|$	$\|E\|$	Dynamic MCN
$series-6-ext$	11	30	0,1
$series-7-ext$	12	41	0,86
$domino-100-100-ext$	100	100	0,10
$domino-100-200-ext$	100	100	0,20
$domino-100-300-ext$	100	100	0,30
$Langford$	8	32	0,9
$geom-30a-4-ext$	30	81	1,04
$pigeons-7-ext$	7	21	4,34
$haystacks-06-ext$	36	96	3,59
$Renault1$	101	134	3,38
$Renault2$	101	113	8,87
$Renault-Modified-6$	111	147	13,07
$Renault-Modified-24$	111	159	13,2
$Renault-Modified-30$	111	154	17,28
$Renault-Modified-33$	111	154	15,42
$Renault-Modified-47$	108	149	14,64

consistent with this general strategy we adopt the same heuristic for choosing the root.

Table 6 gives the results obtained with a static choice of the successor node and Table 7 gives the results obtained with a dynamic choice. We observe here that dynamic choice are better thant the static one.

Table 6: Static choice of the sons.

Problems	Time(s)				
	BE	LC	SC	LNS	LS
$series-6-ext$	0,09	0,07	0,06	0,07	1,01
$series-7-ext$	0,10	12,48	3,18	11,50	9,80
$domino-100-100-ext$	0,12	0,14	0,49	0,132	0,09
$domino-100-200-ext$	0,24	0,23	0,27	0,23	0,27
$domino-100-300-ext$	0,35	0,36	0,69	0,34	0,34
$haystacks6$	3,33	3,35	4,01	3,22	3,29
$Langford2-4$	0,03	0,51	0,05	9,73	1,02
$geom-30a-4-ext$	0,03	0,125	5	0,15	0,19
$pigeons-7-ext$	2,54	4,46	3,5	9,71	5,22
$Renault1$	3,26	3,63	3,35	3,25	3,32
$Renault2$	8,71	20	9,25	7,39	> 30
$Renault-Modified-6$	21	18,07	/	/	/
$Renault-Modified-24$	17,09	15,2	/	/	/
$Renault-Modified-30$	22,11	28,02	/	/	/
$Renault-Modified-33$	24,12	27,03	/	/	/
$Renault-Modified-47$	13,01	12,02	/	/	/

5.6 Comparing with the BTD Algorithm (Jégou et al., 2009)

This section compares our results with the one obtained by BTD for some benchmarks of the Modified Renault family in Table 8. Due to the fact that two different machines are used to experiment BTD and HD_DBT (PC Pentium IV 3,4 Ghz for BTD, laptop HP Compact 6720 s, 1,7 Ghz for HD_DBT) the reported computational times in the row HD_DBT are normalized according to two benchmarks (Spe, ; GeB,). According to these benchmarks, the pentium IV is 4,5 times faster than the HP Compact 6720s. The reported results for BTD are the best ones found in (Jégou et al., 2009) and HD_DBT present nearly comparable results. But we have to mention this last test is not very significative since the detailed implementation of our approach could be much improved by using more adequate data structures and general implementation optimization techniques such the one used in BTD which is more mature. Notice that the enumerative algorithms to solve CSP suffer to solve the modified renault family of benchmarks[2] because these problems are much structured. Notice that the symbol / in the table 8 means that we have no result for the considered instance in (Jégou et al., 2009).

Table 8: BTD_HMIN(HD) vs. HD_DBT.

Problems	Size		Time(s)	
	$\|V\|$	$\|E\|$	HD_DBT	BTD HMIN(HD)
$Renault-Modified-4$	111	147	16	/
$Renault-Modified-6$	111	147	4,5	2,70
$Renault-Modified-9$	111	147	10	/
$Renault-Modified-13$	111	149	13	/
$Renault-Modified-17$	111	149	22	3.41
$Renault-Modified-24$	111	159	3, 7	7,67
$Renault-Modified-30$	111	154	7, 2	8,25
$Renault-Modified-33$	111	154	6, 5	/
$Renault-Modified-42$	108	149	7	2,50
$Renault-Modified-47$	108	149	3	80,25

6 CONCLUSIONS

In this paper we have presented HD_DBT a new algorithm exploiting Hypertree Decomposition to solve non-binary Constraint Satisfaction Problems. This algorithm clearly improves the first and basic Algorithm 1 due to Gottlob et al. (Gottlob et al., 2001).

[2]http://www.cril.univ-artois.fr/lecoutre/research/benchmarks

This last one suffers from the memory explosion problem and it is consequently inefficient face to large instances of CSP. Even if HD_DBT improves the basic approach, it is also limited to small structured instances. To improve HD_DBT we propose its Forward Checking version (HD_DBT+FC) which clearly improves the previous one as FC does for BT. Moreover we proposed and compared several strategies to achieve the best depth-first traversal of the Hypertree Decomposition. These strategies concern the choice of the root, that is the first node and the order between the sons of a given node to be visited by the algorithm. The experimental results show that:

- the best choice of the root corresponds to the hardest node, meaning the node which mimizes the ratio between the number of solutions and the maximal number of solutions;
- the best way to visit the sons of a given node corresponds to the dynamic heuristic based on the remaining tuples in their associated relations;
- combining the best heuristic for choosing the root with a dynamic heuristic for choosing the next son of a given node leads to appreciable results.

Finally we compare our approach with the BTD algorithm (Jégou et al., 2009). From a theoretical point of view the two approaches are similar in the sense they are both enumerative algorithms exploiting some structural decomposition. Our approach differs by exploring in a dual manner an hypertree decomposition instead of a tree decomposition. We compare our HD_DBT with the best BTD version on different benchmarks showing HD_DBT and BTD have comparable performance. To conclude notice that our approach can gained over with several improved data structures and different technical optimizations that we have not yet implemented.
As main perspectives, it would be interesting to apply HD_DBT approach to other structural decomposition methods and to improve the structural decomposition by taking into account semantic properties of the CSPs.

REFERENCES

GeekBench benchmarks. http://www.primatelabs.ca/geekbenchs.

Standard Performance evaluation Corporation (SPEC). http://www.spec.org.

David Cohen, Peter Jeavons, M. G. (2005). A unified theory of structural tractability for constraint satisfaction problems. In *Proceedings of IJCAI ' 05*.

Dechter, R. (1999). unifying framework for reasoning. *Artificial Intelligence*, 113.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Dechter, R. and Pearl, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366.

Dermaku, A., Ganzow, T., Gottlob, G., McMahan, B., Musliu, N., and Samer, M. (2005). Heuristic methods for hypertree decompositions. Technical report, DBAI-R.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *Journal of the Association for Computing Machinery*, 29:24–32.

Gottlob, G., Crohe, M., and Musliu, N. (2005). Hypertree decomposition: structure, algorithms and applications. In *Proceeding of 31 st International workshop WG*, Metz.

Gottlob, G., Leone, N., and Scarcello, F. (1999). On tractable queries and constraints. In *Proceedings of DEXA'99*.

Gottlob, G., Leone, N., and Scarcello, F. (2000). A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124:243–282.

Gottlob, G., Leone, N., and Scarcello, F. (2001). Hypertree decompositions: A survey. In *Proceedings of MFCS '01*, pages 37–57.

Gottlob, G., Leone, N., and Scarcello, F. (2002). Robbers, marshals and guards : Theoretic and logical characterizations of hypertree width. *Journal of the ACM*.

Greco, G. and Scarcello, F. (2010). On the power of structural decompositions of graph-based representations of constraint problems. *Artificial Intelligence*, 174:382–409.

Grohe, M. and Marx, D. (2006). Constraint solving via fractional edge covers. *ACM 2006*, C-30(2):101–106.

Gyssens, M., Jeavons, P. G., and Cohen, D. A. (1994). Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66:57–89.

Harvey, P. and Ghose, A. (2003). Reducing redundancy in the hypertree decomposition scheme. In *Proceeding of ICTAI'03*, pages 548–555, Montreal.

Jeavons, P. G., A, C. D., and Gyssens, M. (1994). A structural decomposition for hypergraphs. *Contemporary Mathematics*, 178:161–177.

Jégou, P., Ndiaye, S. N., and Terrioux, C. (2009). Combined strategies for decomposition-based methods for solving csps. In *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2009)*, pages 184–192.

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

Korimort, T. (2003). Heuristic hypertree decomposition. *AURORA TR 2003-18*.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to pictures processing. *Information Sciences*, 7:95–132.

Musliu, N. and Schafhauser, W. (2005). Genetic algorithms for generalized hypertree decompositions. *European Journal of Industrial Engineering*, 1(3):317–340.

Pang, W. and Goodwin, S. D. (1997). Constraint-directed backtracking. In *1Oth Australian Joint Conference on AI*, pages 47–56, Perth, Western Australia.

Pang, W. and Goodwin, S. D. (2003). A graph based backtracking algorithm for solving general csps. In *Lecture Notes in Computer Sciences of AI 2003*, pages 114–128, Halifax.

Samer, M. (2005). Hypertree-decomposition via branch-decomposition. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 1535–1536, Edinburgh, Scotland.

Sathiamoorthy, S. and Andersen, H. R. (2007). Backtracking procedures for hypertree, hyperspread and connected hypertree decomposition of csps. In *Proceedings of the IJCAI-07*.