HYPERTREE DECOMPOSITION FOR SOLVING CONSTRAINT SATISFACTION PROBLEMS

Abdelmalek Ait-Amokhtar, Kamal Amroun
University of Bejaia, Algeria

Zineb Habbas
LITA, University of Metz, France

Keywords: CSP, Structural decompositions, Hypertree decomposition, Acyclic Solving algorithm, Enumerative search.

Abstract: This paper deals with the structural decomposition methods and their use for solving Constraint Satisfaction problems (CSP). mong the numerous structural decomposition methods, hypertree decomposition has been shown to be the most general CSP decomposition. However so far the exact methods are not able to find optimal decomposition of realistic instances in a reasonable CPU time. We present Alea, a new heuristic to compute hypertree decomposition. Some experiments on a serial of benchmarks coming from the literature or the industry permit us to observe that Alea is in general better or comparable to BE (Bucket Elimination), the best well known heuristic, while it generally outperforms DBE (Dual Bucket Elimination), another successful heuristic. We also experiment an algorithm (acyclic solving algorithm) for solving an acyclic CSP obtained by using the heuristic Alea. The experimental results we obtain are promising comparing to those obtained by solving CSP using an enumerative search algorithm.

1 INTRODUCTION

Many important real world problems can be formulated as Constraint Satisfaction Problems (CSP)s. A CSP consists of a set V of variables each with a domain D of possible values and a set C of constraints on the allowed values for specified subsets of variables. A solution to CSP is the assignment of values to variables which satisfies all the constraints. The usual method for solving CSPs is based on backtracking search. This approach has an exponential theoretical time complexity in $O(e.d^n)$ for an instance of CSP for which e is the number of constraints, n is the number of variables and d is the cardinality of the largest domain of variables. Even if CSPs are NP-complete in general, some decomposition methods have been successfully used to characterize some tractable classes (Dechter and Pearl, 1989; Gyssens et al., 1994; Jeavons et al., 1994; Gottlob et al., 2000), biconnected components (Freuder, 1982), hinge decomposition (Gyssens et al., 1994), hinge decomposition combined with tree clustering (Gyssens et al., 1994), hypertree decomposition and generalized hypertree decompositions (Gottlob et al., 1999a; Gottlob et al., 2005) and more recently Spread Cut Decomposition (Cohen et al., 2005). The main principle of these methods is to decompose the CSP into subproblems that are organized in a tree or a hypertree structure. The sub-problems can be solved independently in order to solve the initial CSP. All these methods are characterized by their computational complexity and the width or hypertree-width of respectively the tree or the hypertree they generate. Among these numerous methods, Gottlob and al, (Gottlob et al., 1999a), have shown that hypertree decomposition dominates all the other structural decomposition methods excepted the recently introduced spread-cut decomposition method (Cohen et al., 2005). The first approach proposed to compute the hypertree decomposition is the exact algorithm (opt-k-decomp (Gottlob et al., 1999b)). For any parameter k, opt-k-decomp determines if a hypertree decomposition of width w bounded by k exists. If the response is yes, the algorithm computes this decomposition . Unfortunately, opt-k-decomp and all its improvements are not efficient in practice and run out of time and space for realistic instances. To overcome this limitation, some heuristics have been explored to find a hyper-

Ait-Amokhtar A., Amroun K. and Habbas Z. (2009).
HYPERTREE DECOMPOSITION FOR SOLVING CONSTRAINT SATISFACTION PROBLEMS.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 85-92
DOI: 10.5220/0001662400850092

tree decomposition in a reasonable time, even if it is not optimal((Musliu and Schafhauser, 2007), (Korimort, 2003), (McMahan, 2003)). In this paper we consider a new heuristic to compute a hypertree without using the tree decomposition. To do this, we first consider a particular definition of hypertree decomposition and then we propose a generic algorithm which proceeds in two steps. In the first step, we look for a first separator of the hypergraph which will constitute the root of the output hypertree. A separator is a set of hyperedges whose removal either makes the problem smaller or decomposes a given problem into at least two components each of which is smaller than the orignal one. In the second step, this separator is used for partitioning the hypergraph by removing its vertices. Afterwards, each component is recursively partitioned, according to the connectedness condition to be satisfied by the resulting decomposition. This heuristic can be considerably influenced by the choice of this first separator. In order to compute the first separator we propose the heuristic Alea which considers the first separator as the first constraint introduced by the user or the first constraint in the file describing a given CSP instance. Some experiments on a serial of benchmarks coming from the literature or the industry (Ganzow et al., 2005) permit us to observe that Alea is in general better or comparable to BE(Bucket Elimination), the best well known heuristic as far as we know, while it generally outperforms DBE (Dual Bucket Elimination), another successful heuristic. We also experiment the resolution of constraint satisfaction problem based on a hypertree decomposition and the results we obtain are promising comparing to those obtained by solving CSP using nFC2-MRV, a forward-checking algorithm with a Minimum Remaing Value ordering algorithm (C.Bessière and Larossa, 2002).

2 PRELIMINARIES

The notion of Constraint Satisfaction Problems (CSP) was introduced by Montanari in (Montanari, 1974).

Definition 1 (Constraint Satisfaction Problem). *A constraint satisfaction problem is defined as a 3-uplet $P =< V,D,C >$ 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 values 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 $S(c_i) \subset X$, is a subset of variables, called **scope** of c_i and $R(c_i) \subset \prod_{x_k \in S(c_i)} d_k$ is the constraint relation, that specifies the allowed combinations of values.*

In order to study the structural properties of constraints satisfaction problems, we need to present the following definitions relative to the graphical representation of CSPs.

Definition 2 (Graph (Dechter, 2003)). *A graph $\mathcal{G} =< V,E >$ is a structure that consists of a finite set of vertices V and a set of edges E. Each edge is incident to an unordered pair of vertices $\{u,v\}$.*

Definition 3 (Hypergraph (Dechter, 2003)). *A hypergraph is a structure $\mathcal{H} =< V,E >$ that consists of a set of vertices V and a set of hyperedges E . Mainly a hyperedge $h \in E$ is a subset of vertices V. The hyperedges differ from regular edges in that they may connect more than two variables.*

The structure of a CSP $P =< V,D,C >$ is captured by a hypergraph $\mathcal{H} =< V,E >$ where the set of vertices V is the same as the set of variables V and the set of hyperedges E corresponds to the set of constraints C. For the set of hyperedges $K \subseteq E$ let the term $vars(K) = \bigcup_{e \in K} e$ i.e the set of the variables occuring in the edges K. For a set $L \subseteq V$ let the term $edgevars(L) = vars(\{e | e \in E, e \cap L \neq \emptyset\})$ i.e all the variables occuring in a set of edges where each edge in the set contains at least one variable in L.

Definition 4 (Hypertree (Gottlob et al., 2001)). *Let $\mathcal{H} =< V,E >$ a given hypergraph. A hypertree for hypergraph $\mathcal{H}$ is a triple $< T,\chi,\lambda >$ where $T = (N,E)$ is a rooted tree and χ and λ are two functions labelling each node of T. The functions χ and λ map each node $p \in N$ by two sets $\chi(p) \subseteq V$ and $\lambda(p) \subseteq E$*
A tree is a pair $< T,\chi >$ where $T = (N,E)$ is a rooted tree and χ is a labelled function as previously defined.

Definition 5 (Generalized Hypertree Decomposition (Gottlob et al., 2001)). *A generalized hypertree decomposition of a hypergraph $\mathcal{H} =< V,E >$ is a hypertree $HD =< T,\chi,\lambda >$ which satisfies the following conditions:*

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

The width of Generalized hypertree decomposition $< T,\chi,\lambda >$ is $max_{p \in vertices(T)} |\lambda(p)|$. The generalized hypertree width: $Ghw(\mathcal{H})$ of hypergraph is the minimum width over all its generalized hypertree decompositions

Definition 6 (Hypertree Decomposition (Gottlob et al., 2001)). *A hypertree decomposition is a generalized hypertree decomposition decomposition which*

additionally satisfies the following special condition:

For each vertex $p \in vertices(T)$, $var(\lambda(p)) \bigcap \chi(T_p) \subseteq \chi(p)$.

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

Definition 7 *A hyperedge h of a hypergraph* $\mathcal{H} =< V, E >$ *is* ***strongly covered*** *in* $HD =< T, \chi, \lambda >$ *if there exists* $p \in vertices(T)$ *such that the vertices in h are contained in* $\chi(p)$ *and* $h \in \lambda(p)$.

Definition 8 *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* $HD =< T, \chi, \lambda >$.

3 THE HEURISTIC ALEA

To compute hypertree decomposition, two approaches are proposed in the literature : the exact methods (Gottlob et al., 1999b; Harvey and Ghose, 2003) and the heuristic ones (Korimort, 2003; Musliu and Schafhauser, 2007; Dermaku et al., 2005). In this section, we introduce our new heuristic for computing hypertree decomposition.

3.1 Theoretical Consideration

First we consider a particular hypertree decomposition which satisfies the following conditions

1. For each edge $h \in E$, there exists $p \in vertices(T)$ such that $var(h) \subseteq \chi(p)$. We say that p covers h.
2. For each variable $v \in V$, the set $\{p \in vertices(T) | v \in \chi(p)\}$ induces a connected subtree of T.
3. For each vertex $p \in vertices(T), \chi(\mathbf{p}) = \mathbf{var}(\lambda(\mathbf{p}))$.

The first two conditions are the same as the first two conditions of the hypertree decomposition. The condition 3 is a particular case of the condition of the hypertree decomposition which forces the χ labels of each node p of T to be equal to $var(\lambda(p))$. In addition, this same condition 3 satisfies trivially the condition 4 of the hypertree decomposition.

3.2 Description of Alea

BE and DBE are both two extensions of the BE heuristic for tree decomposition. Unlike BE and DBE the heuristic Alea computes a hypertree without using the tree decomposition processing. We look for only the $\lambda - labels$ since the $\chi - labels$ are obtained for free (At each node p, the set $\chi(p)$ is the union of vertices of the each hyperedge c , for $c \in \lambda(p)$. Our goal is to obtain a hypertree decomposition with a small width, thus we have to minimize the size of the $\lambda - labels$.
The algorithm 1 Gen-decomposition gives the main generic procedure for computing a hypertree decomposition of a given hypergraph $\mathcal{H}$. This algorithm begins by determining *the first separator* or the root of the hypertree (Line 4 of the algorithm 1) for a given hypergraph $\mathcal{H}$. Once the first separator is defined a node root for the hypertree is created. In Line 8 of the algorithm 1, the procedure Decompose is called to decompose the resulting hypergraph.

Algorithm 1: Gen-decomposition($\mathcal{H}$).

1: **Input**: a given hypergraph $\mathcal{H}$
2: **Output**: its hypertree $\mathcal{HT}$
3: **Begin**
4: Find-first-separator($\mathcal{H}$, MinSep);
5: $create - node(root, \text{HT})$;
6: $\lambda(root) = \lambda(MinSep)$;
7: $\chi(root) = vars(\lambda(MinSep))$;
8: Decompose($\mathcal{H}$, MinSep,root);
9: **End.**

This algorithm is generic because it can use the same procedure Decompose and different procedures to find the first separator. In this paper we consider the heuristic Alea to determinate the first separator. This heuristic considers as a first separator, the first constraint introduced by the user or the first constraint in the file giving the description of a CSP instance. In the sequel, we describe the procedure Decompose.

3.2.1 The Procedure Decompose

The procedure **Decompose** (see algorithm 2) considers a given hypergraph or a subhypergraph Comp (set of hyperedges from E), the current separator Sep and the last node p in the hypertree as parameters. An appropriate separator for a component (sub-hypergraph) must be a minimal size separator composed of a set of hyperedges of the given component covering the variables of intersection between the variables of the last separator and those of the component. Decompose uses an auxiliary function and an auxiliary procedure named respectively separate and newsep.

The function Separate(Comp, sep): Given a component Comp and a set of hyperedges sep, separate removes the hyperedges of sep from the hyperedges

Algorithm 2: Decompose (Comp, Sep, p).

1: **Input**: **Comp** a component to be decompose , **Sep** the current separator to use for decomposing Comp, p the last node in the hypertree;
2: **Output**: a node p' son of p;
3: **Begin**
4: Components := separate(Comp,Sep);
5: **for** each component $C_i \in Components$ **do**
6: new-sep(C_i, Sep, NewSep);
7: create-node(p');
8: $\lambda(p') = \lambda(NewSep)$;
9: $\chi(p') = vars(\lambda(NewSep))$;
10: connect (p', p); // p' is the son of p in the hypertree;
11: **if** (isdecomposable) **then**
12: Decompose(C_i, NewSep,p');
13: **end if**
14: **end for**
15: **End.**

of comp in order to obtain some connected components. For each of these connected components, the procedure Decompose calls the procedure **new-sep** to determine the new separator to be used again for decomposing this current component (see algorithm 2, line 4-7). This new separator is computed by using the set cover techniques (Computing a minimal set of hyperedges covering the variables of intersection between the variables of the last node and the variables of the component) and by respecting the rules of hypertree decomposition as well. This procedure Decompose is recursively called. This process stops when all the components become not decomposable. A component is not decomposable if the minimal set of hyperedges required to cover the intersection of its variables with the variables of the last separator is equal to the number of all its hyperedges or if the size of the component is 1 (the component has only one hyperedge).

3.2.2 The Procedure New-sep

This procedure finds a set Y of all hyperedges in the component Comp that have at least one variable in the last node Sep, (see algorithm 3) and calls the procedure minrecovering-sep which mainly minimizes the size of Y.

3.2.3 The Procedure Minrecovering-sep

Given a set of variables X and a set of hyperedges E, this procedure looks for the minimal set of hyperedges c ($c \in E$)) , covering a maximum number

Algorithm 3: New-sep (Comp, Sep, Bestsep).

1: **Input**: Comp a component, Sep is the last separator used
2: **Output**: Bestsep a separator for decomposing Comp and isdecomposable a boolean;
3: **Begin**
4: $NSep := \phi$; // Nsep : set of hyperedges that have at least one variable in Sep
5: **for** each hyperedge $c \in hyperedges(Comp)$ **do**
6: **if** $(vars(Sep) \cap vars(c)) \neq \phi$) **then**
7: $NSep := NSep \cup c$;
8: **end if**
9: **end for**
10: **End.**
11: isdecomposable=true;
12: minrecovering-sep(Comp,vars(NSep),Bestsep, isdecomposable);
13: **End.**

of variables in X. Then it removes these covered variables from the set X, removes the hyperedge c from E and takes c as a member of the covering set. The same process is applied for the induced sets X and E until X becomes empty. The algorithm is deliberately omitted because it is classical. **This algorithm returns Bestsep and the boolean isdecomposable witch is true if the component is decomposable**. Notice that our heuristic is different from the one in (Korimort, 2003) in the sense that we do not add a special hyperedge to a component to find an appropriate separator for a component. In addition, our heuristic gives by construction a complete decomposition in which each hyperedge appears only once in node p of the resulting hypertree.

The figure 1(a) shows a given hypergraph and the figure 1(b)) gives its hypertree decomposition using Alea.

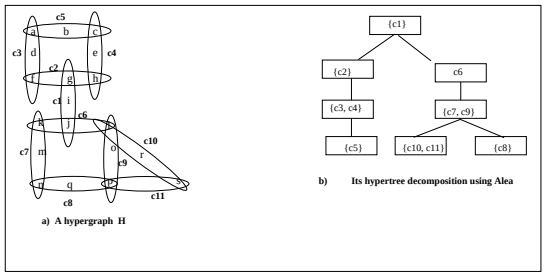

Figure 1: A hypergraph and its hypertree deccomposition.

Table 1: Grid2d family: comparing exact and heuristic methods.

Instances	Size			BE		DBE		Alea		dkd	
	$\|V\|$	$\|E\|$	htw	w	t	w	t	w	t	w	t
grid2d_10	50	50	4	5	0	6	0	6	0	4	0
grid2d_15	113	112	6	8	0	9	0	9	0	6	3
grid2d_20	200	200	7	12	0	11	28	**10**	2	**7**	3140
grid2d_25	313	312	9	15	3	15	43	21	0	10	**200**
grid2d_30	450	450	11	19	7	20	8	**14**	1,6	**13**	1566
grid2d_35	613	612	12	23	15	23	73	26	2	15	1905
grid2d_40	800	800	14	26	28	25	31	**19**	6	**17**	253
grid2d_45	1013	1012	16	31	51	30	56	**30**	11	21	26
grid2d_50	1250	1250	17	33	86	32	94	**21**	39	24	2786
grid2d_60	1800	1800	21	41	204	34	178	**33**	72	31	3940

3.3 Correctness of Our Heuristic

In this section, we prove that our heuristic computes a hypertree decomposition.

Proof 1 *1. The condition 1 is satisfied due to the fact that in our approch, all hyperedges appear in the hypertree and since $\chi(p) = vars(\lambda(p))$ for each node p in the hypertree.*

2. *To prove the condition 2, two cases are to consider.*
   - *case1: p_1 is the parent of p_2 and p_2 is the parent of p_3. We have to prove that the proposition $x \in \chi(p_1), x \in \chi(p_3)$ and $x \notin chi(p_2)$ is not possible. In fact if $x \in \chi(p_1)$, according to the procedure Decompose either $x \in \chi(p_2)$ and so mutch the better. Either $x \notin \chi(p_2)$ and then according to the procedure Decompose $x \notin \chi(p_3)$.*
   - *case2: p_2 and p_3 are the sons of p_1. We have to prove that the proposition $x \in \chi(p_2)$ and $x \in \chi(p_3)$ and $x \notin \chi(p_1)$ is not possible. Let be $x \in p_2$ then either $x \in chi(p_1)$ and so mutch the better. Either $x \notin \chi(p_2)$ an then $x \notin \chi(p_3)$*
3. *The condition 3 is trivially satisfied.*
4. *The condition 4 of the definition 5 is trivially satisfied because for each vertex $p \in vertices(T)$, $var(\lambda(p)) \bigcap \chi(T_p) \subseteq \chi(p) = var(\lambda(p))$. ($because \chi(p) = var(\lambda(p))$).*

4 EXPERIMENTS

In this section we report the computational results obtained with implementation of our heuristic Alea when we applied it to some different classes of hypergraphs (Ganzow et al., 2005). We compare our results with the results obtained from evaluations of BE and DBE. The tests for BE and DBE have been done with three variable ordering heuristics MCS (Maximum Cardinality Search), MIW (Minimum Induced Width) and MF(Min Fill). But we report here only the best obtained results in each case. Nevertheless, MIW is in the general the best variable ordering. All the experiments were performed on PENTIUM 4,2.4 GHZ, 512 MO of RAM and running under OPENSUSE Linux 10.2.

- **Dimensional Grids Benchmarks:** The first class of benchmarks are hypergraphs extracted from two dimensional grids. The results obtained are reported in the table 1, where the first column contains the name of the example. The columns 2 and 3 contain respectively the number of variables and the number of hyperedges. The column 4 contains the hypertree width if this is known. Then there are two columns for each heuristic (BE, DBE and Alea). The first one contains the minimal width and the second one gives the runtime in seconds.

  For this class of hypergraphs, in (Gottlob and Samer, 2007) the authors observe that even if the optimal decompositions can be constructed by hand, their algorithm det-k-decomp seems suffer to compute a hypertree decomposition. They observe also that opt-k-decomp cannot solve any of them within a timeout of one hour. Notice that Alea Solves all the problems with a width close to those obtained with det-k-decomp. In addition, the CPU time is smaller with Alea. These results are reported in the table 1. The results for det-k-decomp and opt-k-decomp are taken from (Gottlob and Samer, 2007).
- **The Dubois Benchmarks:** In the table 2 we report the results obtained with the Dubois benchmarks which is a particular benchmark of SAT problems. Alea is equivalent to BE and DBE.

Table 2: Dubois family: Comparing Alea, BE and DBE.

Instances	$\|V\|$	$\|E\|$	BE	DBE	Alea
Dubois20	60	40	2	2	2
Dubois21	63	42	2	2	2
Dubois22	66	44	2	2	2
Dubois23	69	46	2	2	2
Dubois24	72	192	2	2	2
Dubois25	75	200	2	2	2
Dubois26	78	208	2	2	2
Dubois27	81	216	2	2	2
Dubois28	84	224	2	2	2
Dubois29	87	232	2	2	2
Dubois30	90	240	2	2	2
Dubois50	150	400	2	2	2
Dubois100	300	800	2	2	2

5 SOLVING ACYCLIC CSP

In order to show the practical interest of our approach, we experiment here the resolution of a given CSP after its transformation into a hypertree. Given a CSP and its structutal representation into an hypergraph $\mathcal{G} =< V,E >$ we first compute its hypertree $< T,\chi,\lambda >$ using Alea. Then, the hypertree is transformed into an equivalent acyclic CSP . Finally, the resulting CSP is solved using the Acyclic Solving algorithm (Dechter et al., 1988)

5.1 Building and Equivalent Acyclic CSP

Given a CSP $\mathcal{P} = (X,D,C,R)$ and its hypertree decomposition $< T,\chi,\lambda >$, the equivalent acyclic CSP $\mathcal{P} = (X','D,C',R')$ is obtained as follows: $X' = X$, $D' = D$, $C' = \{\chi(p)|p \in vertices(T)\}$ and $R' = \{\bowtie (\lambda(p)[\chi(p)]|p \in vertices(T)\}$ where $\bowtie$ represents the join operation and $[X]$ represents the projection operator on the set X of variables.

5.2 Acycling Solving Algorithm

The first algorithm proposed is the tree solving(Dechter et al., 1988) which solves acyclic binary CSPs in polynomial time The generalization of this algorithm for solving non binary CSP gave rise to the Acyclic Solving algorithm (Dechter, 2003) . This algorithm begins by ordering the nodes of the hypertree such that every node parent precedes its sons(Ligne 4 of 4). Then, it treats successively all the nodes coming from the leaves to the root. For each node p of T, it realizes a semi jointure on itself and its parent in order to filter the parent relation (Lignes 6-12 4) . When the root is treated without making empty any relation, the solution of the CSP is determinated in a backtrack free manner (Lignes 14-16 of 4).

Algorithm 4: Acyclic Solving (Comp, Last-sep, Sep, p).

1: **Entres**: $\mathcal{P} = (X,D,C,R)$, $R = \{R_1,R_2,\ldots,R_t$ is an hypertree $< T,\chi,\lambda >$
2: **Sortie** : Verifies the consistency and looks for in the positive case.
3: **Dbut**
4: $d = (R_1,R_2,\ldots,R_t)$ an ordering where each relation preceeds its sons in the rooted hypertree T. .
5: **for** i= t to 2 **do**
6: **for** each edge (j, k) , k ¡ j, in the hypertree **do**
7: $R_k = (R_k \propto R_j)[C_k]$
8: **if** $R_k = \emptyset$ **then**
9: Exit; The problem is inconsistent.
10: **end if**
11: **end for**
12: **end for**
13: /* The CSP is consistent, look for its solution
14: Select a tuple from R1.
15: **for** i = 2 t **do**
16: Select a tuple from Ri consistent withe previous choosen tuples.
17: **end for**

Table 3: Dubois family: Comparing Acyclic Solving and nFC2-MRV.

Instances	$\|V\|$	$\|E\|$	Alea	BE	nFC2-MRV
Dubois20	60	40	0,23	0,31	419
Dubois21	63	42	0,24	0,34	896
Dubois22	66	44	0,24	0,34	1916
Dubois23	69	46	0,29	0,33	15026
Dubois24	72	48	0,29	0,37	-
Dubois25	75	50	0,27	0,34	-
Dubois26	78	52	0,29	0,36	-
Dubois27	81	54	0,31	0,39	-
Dubois28	84	56	0,36	0,41	-
Dubois29	87	58	0,32	0,42	-
Dubois30	90	60	0,35	0,43	-

5.3 Experimental Evaluation

Our approach has been tested on 11 SAT problems from the Dubois familly. For each problem, we compare the enumerative algorithm nFC2-mrv (C.Bessière and Larossa, 2002) with the Acyclic Solving algorithm 4 applied to the hypertree decompositions derived from BE and Alea heuristics. Our experiments are reported in the table 3. We clearly observe that Acyclic Solving algorithm outperforms the

CPU time. The symbol – signifies that nFC2-MRV has been interruped after 5 hours of execution time. We also observe that the results obtained by Alea are slightly better that those obtained by BE. This is due to the fact that no jointure operations has been done by using Alea.

Finally the figures figures 2 and 3 show the practical interest of a polynomial algorithm comparing to an exponential one.

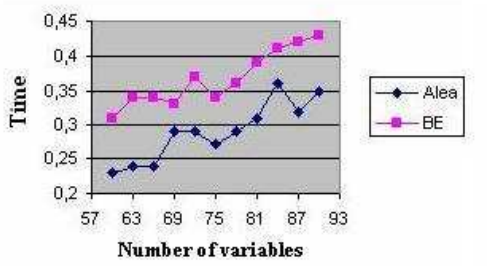

Figure 2: Solving with Acyclic Solving, BE and Alea.

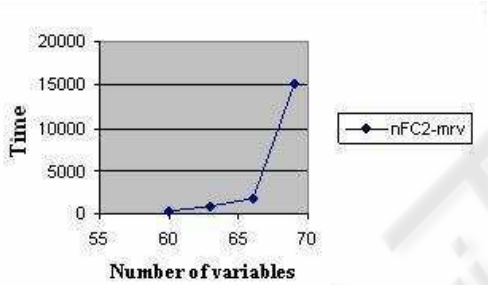

Figure 3: Solving with nFC2-MRV.

6 CONCLUSIONS

We have studied and tested an approach to solve CSP by decomposition.

First, we proposed the heuristic Alea to compute the hypertree decomposition. Some experiments have shown that Alea is comparable and sometimes better on certain problems than BE (the most successful heuristic and DBE). Then we have studied and implemented an acyclic solving (Dechter, 2003) algorithm to solve a given CSP after it was transformed into a hypertree decomposition. Our heuristic "Alea" is particular as it does not require the projection operations. As a consequence, execution time to solve a CSP using Alea is slightly lower than with BE.

We have shown the interest of this approach on a particular family of problems. A work currently in progress consists in the development of a parallel acyclic solving algorithm with a judicious strategy for the saving of data. Indeed besides the CPU time, one of the major disadvantages of the Acyclic Solving is the use of a large space memory.

REFERENCES

C.Bessière, P. Meseguer, C. F. and Larossa, J. (2002). On forward checking for non binary constraint satisfaction. *Artificial Intelligence*, 141:205–224.

Cohen, D., Jeavons, P., and Gyssens, M. (2005). A unified theory of structural tractability for constraint satisfaction problems and spread cut decomposition. In *Proceedings of IJCAI 2005*, pages 72–77.

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

Dechter, R., Dechter, A., and Pearl, J. (1988). Optimization in constraint-networks. In *Proceedings of the Conference on Influence Diagrams for Decision Analysis: Inference and prediction*, Berkeley.

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.

Ganzow, T., Gottlob, G .and Musliu, N., and Samer, M. (2005). A csp hypergraph library. Technical report, DBAI-TR-2005-50, Technische Universitt Wien.

Gottlob, G., Grohe, 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. (1999a). A comparison of structural csp decomposition methods. In *Proceedings of IJCAI'99*, pages 394–399.

Gottlob, G., Leone, N., and Scarcello, F. (1999b). 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. and Samer, M. (2007). A backtraching based algorithm for computing hypertree decompositions. *arXivics.DS 0701083vl 14 Jan 2007*.

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 *Proceding 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.

Korimort, T. (April 2003). *Heuristic decomposition Hypertree Decomposition*. PhD thesis, Vienna University of Technology.

McMahan, B. (2003). Bucket elimination and hypertree decompositions. Technical report, Implementation report, institute of information systems (DBAI), TU, Vienna.

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

Musliu, N. and Schafhauser, W. (2007). Genetic algorithms for generalized hypertree decompositions. *European Journal of Industrial Engineering*, 1(3):317–340.