A Time-space Trade-off for Constraint Networks Decomposition

Philippe Jégou
LSIS - Université d'Aix-Marseille 3
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
philippe.jegou@univ.u-3mrs.fr

Cyril Terrioux
LSIS - Université d'Aix-Marseille 3
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
cyril.terrioux@univ.u-3mrs.fr

Abstract

We study here a CSP decomposition method introduced in [9] and called Cyclic-Clustering. While [9] only presents the principles of the method, this paper explains how this method can be made operational by exploiting good properties of triangulated induced subgraphs. After, we give formal results which show that Cyclic-Clustering proposes a time-space trade-off w.r.t. theoretical complexities. Finally, we present some preliminary experiments which show that Cyclic-Clustering may be efficient in practice.

1. Introduction

The CSP formalism (Constraint Satisfaction Problem) offers a powerful framework for representing and solving efficiently many problems. In particular, many academic or real problems can be formulated in this framework which allows the expression of NP-complete problems. Generally, CSPs are solved by different versions of backtrack search whose time complexity is exponential in the size of the instance. Nevertheless, theoretical results have shown that, for some particular instances, we can provide better complexity bounds. These new bounds are often obtained by applying decomposition methods which exploit some topological properties of constraint networks which are represented by graphs (or hypergraphs). For example, in [2, 5], two decomposition strategies have been proposed, the *Cycle-Cutset method* (CCM) and the *Tree-Clustering scheme* (TC). However, while their theoretical interest seems significant (see [7] for an analysis), their practical advantages have not been proved yet. Firstly [9] and recently [4] have proposed some trade-offs between time and space complexity to make this class of approaches usable. Moreover, in [10], an hybrid approach, which combines backtracking with structural decomposition has shown its practical interest for solving hard CSPs. So, it seems that trade-offs and hybrid approaches allow to propose realistic implementations of structural methods. This paper studies this direction by analyzing and trying to make usable the *Cyclic-Clustering* method (CC [9]). Note that [9] only describes the motivations and principles of this method. Our description of CC allows us to give complexity results, which prove that this method can actually make a compromise between TC and CCM. For example, we demonstrate that the theoretical time and space complexities of CC are located between ones of TC and CCM. Finally, we present an implementation based on BTD [10] which allows CC to obtain interesting practical results.

The paper is organized as follows. Section 2 introduces the main definitions about the CSP formalism and decomposition methods like TC and CCM. Section 3 presents theoretical foundations of CC and section 4 gives the formal comparisons w.r.t. TC and CCM. Section 5 deals with an efficient implementation of CC and presents some preliminary experimental results. Finally, we conclude in section 6. For lack of place in this paper, all the proofs of properties and theorems are given in [11].

2. CSPs and decomposition methods

A *constraint satisfaction problem* (CSP) also called a *constraint network*, consists of a set of variables which must be assigned with a value from finite domains such that each constraint is satisfied. Formally, a CSP is defined by a tuple (X, D, C, R). X is a set $\{x_1, \ldots, x_n\}$ of n variables. Each variable x_i takes its values in the finite domain D_i from D. Variables are subject to constraints from C. Each constraint C_i is defined as a set $\{x_{i_1}, \ldots, x_{i_{j_i}}\}$ of variables. A relation R_i (from R) is associated with each constraint C_i such that R_i represents the set of allowed tuples over

$D_{i_1} \times \cdots \times D_{i_{j_i}}$. A solution of a CSP is an assignment of a value to each variable which satisfies all the constraints. For a CSP $\mathcal{P}$, the hypergraph (X, C) is called the constraint hypergraph. A CSP is said binary if all the constraints are binary, i.e. they involve only two variables, so (X, C) is a graph (called the constraint graph) associated to (X, D, C, R). For a given CSP, the problem is either to find all solutions or one solution, or to know if there exists any solution. The decision problem (existence of solution) is NP-complete.

Generally, CSPs are solved by different versions of backtrack search whose time complexity is exponential in the size of the instance. Consequently, many works try to improve the search efficiency. They mainly deal with binary CSPs. In [6], Freuder gives a preprocessing procedure for selecting a good variable ordering prior to running the search. One of his main results is a sufficient condition for backtrack-free search. The most interesting property indicates that an arc-consistent binary CSP whose constraint graph is acyclic admits a solution and there is a backtrack-free search order (this property also holds for arbitrary CSPs with hypergraphs). This property shows that the tractability of CSPs is intimately connected to the topological structure of their underlying constraint graphs. This property has led authors to propose methods for solving cyclic CSPs, as CCM [2] and TC [5].

Among the methods based on a tree-decomposition of the constraint network, we have chosen, for sake of simplicity and without loss of generality, to describe TC. TC consists in building a new CSP by forming clusters of variables such that the interactions between the clusters are tree structured. The new CSP is equivalent to the first one (i.e. it has the same set of solutions), but the associated constraint hypergraph is acyclic and it can be solved efficiently. The time complexity of TC is $O(n.d^W.W.log(d))$ with d the size of the largest domain and W the arity of the largest constraint in the new CSP (equal to the size of the maximum clique). Its space complexity is $O(m.d^W)$ with m the number of clusters, but can be limited to $O(m.d^S)$ (like, for instance, Cluster Tree Elimination [3]) with S the size of the largest intersection between two clusters (i.e. the maximal size between minimal separators in the graph). Finally, note that the problem of finding the best tree decomposition, that is with the minimal value of W, is NP-hard. For more details, see [3, 5, 7]. In the sequel, we will call TC methods based on tree decomposition as TC [5], Join Tree Elimination [3] or Cluster Tree Elimination [3].

While these methods seem to be the most efficient methods to solve binary CSPs w.r.t. their time complexity (see [7]), their practical interest is really limited because the time computation is too expensive and the required memory size too large in practical cases. So, [10] has proposed an alternative approach, called BTD, which limits the size of the required memory and also avoids some redundancies in the search.

CCM [2] is based on the fact that assigned variables change the effective connectivity of the constraint graph. A cycle-cutset of a graph (resp. hypergraph) is a set of vertices such that the deletion of these vertices induces an acyclic graph (resp. acyclic hypergraph). So, as soon as all the variables of the cycle-cutset are assigned, all the cycles of the constraint graph are cut. For binary CSP, the time complexity of CCM is $O(e.d^{K+2})$ with e the number of constraints and K the size of cycle-cutset while for non-binary CSPs it is $O(e.r.log(r).d^K)$ where r is the size (number of tuples) of the largest relation associated to constraints in the CSP. The most important parameter in this complexity is K, but the problem of minimizing K is NP-hard. Note that for non-binary CSPs, after the assignment of the cycle-cutset, the induced acyclic CSP can be solved using the same kind of procedure than TC.

It has been shown that for optimal values of W and K, we have $W \leq K + 2$, and then, TC is theoretically better than CCM for time complexity. But, neither TC nor CCM has shown their practical interest. As indicated in [10], this is due to the practical space complexity for TC, while it is probably due to the time complexity for CCM.

3. Running cyclic-clustering (CC)

The alternative approach called *cyclic-clustering* (CC) has been introduced in [9] to avoid drawbacks of TC and CCM but [9] only presents the ideas of the approach without any indication on carrying out the method. CC runs in four steps. The first step consists in finding all maximal cliques of the initial constraint graph. The second step considers each maximal clique and then solves the associated subproblems. The result is a constraint hypergraph whose constraints correspond to the solved subproblems. The two last steps consist in finding a minimal cycle-cutset and then running CCM on this new CSP. Unfortunately, running CC is not easy because of some theoretical and practical problems. For example, it is well known that in an arbitrary graph, the number of maximal cliques can be exponential in the number of vertices and then achieving the first step is practically impossible. So, in this section, we introduce another way for running CC, which is based on good combinatorial and algorithmic properties of triangulated graphs. Figure 1 sum-

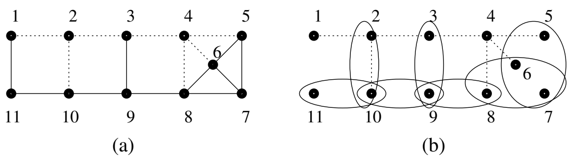

Figure 1. Example of CC decomposition.

marizes our approach. The first step of CC (a) identifies a Triangulated Induced Subgraph (TIS) of the graph (all the vertices and edges belong to the TIS, except vertices 2 and 4 and dotted edges). The second step generates an acyclic n-ary CSP based on the TIS (b). So the last step solves the initial CSP by applying the cycle-cutset method on the new problem, since the vertices which do not belong to the TIS define a cycle-cutset of the new problem (vertices 2 and 4 in this example). Theoretical justifications of this approach are based on properties of TIS.

We need now to recall some definitions on graph and hypergraph theory: The *primal graph* of a hypergraph (X, C) is the graph (X, E) with $E = \{\{X_i, X_j\} \mid \exists C_k \in C \ s.t. \ \{X_i, X_j\} \subseteq C_k\}$. A graph is *triangulated* if every cycle of length at least four has a chord, i.e. an edge joining two non-consecutive vertices along the cycle. Several equivalent definitions have been proposed for acyclic hypergraphs in the literature. Here we consider one related to triangulated graphs. A hypergraph is *conformal* if every maximal clique of its primal graph corresponds to an edge in the original hypergraph (an original constraint). A hypergraph is *acyclic* if it is conformal and its primal graph is triangulated. Given a graph $G = (X, C)$, if T is a subset of X, $G(T) = (T, C(T))$ is the subgraph of G induced by vertices of T, that is $C(T) = \{\{x_i, x_j\} \in C : x_i, x_j \in T\}$. The graph $G(T)$ is a *Triangulated Induced Subgraph* (TIS) of G if and only if $G(T)$ is triangulated. $G(T)$ is a *Maximal TIS* (MTIS) if T is maximal for inclusion, i.e. if $\nexists T'$ such that $T \not\subseteq T' \subseteq X$ and $G(T')$ is triangulated.

In [1], Balas and Yu defined an efficient algorithm called `TRIANG` to find a MTIS; its time complexity is $O(n + e)$. Note that the computed MTIS is not necessary a maximum size induced subgraph w.r.t. the number of vertices, but it is always a maximal subgraph w.r.t. the inclusion in the set of vertices (finding such a maximum subgraph is a NP-hard problem).

To simplify, and without loss of generality, we only present our approach on constraint graphs. To extend this method to non-binary CSPs, it is sufficient to consider the primal graph used in TC. Given the initial constraint graph, the first step consists in finding a TIS. The second step generates a hypergraph which has two kinds of edges. The first kind of edges corresponds to the maximal cliques of the TIS and the second one to the edges of the initial graph that do not belong to the subgraph. The primal graph of this hypergraph is the initial constraint graph and we know a cycle-cutset of this hypergraph which corresponds to the vertices that do not appear in the TIS. From this hypergraph, CC produces a new CSP. Some edges of the new CSP correspond to new constraints. For these new constraints, we must compute all their consistent tuples (w.r.t. the original constraints). The last step consists in applying CCM.

Given a graph $G = (X, C)$, $H_{max}(G) = (X, \mathcal{C}')$ denotes the hypergraph induced by maximal cliques of G, that is $\mathcal{C}'$ is the set of maximal cliques in G. Given a subset Y of X, $E_G(Y) = \{c \in C : c \cap Y \neq \emptyset\}$, i.e. $E_G(Y)$ is the set of edges which contain at least one vertex in Y. Given a hypergraph $H = (X, C)$ and a subset Y of X, $H(Y) = (Y, C')$ denotes the hypergraph induced by the set of vertices Y, where $C' = \{c' \subseteq Y : \exists c \in C, c' \subseteq c$ and c' is maximal $\}$. If Y is such that $H(X \setminus Y)$ is acyclic, then Y is a cycle-cutset of H.

Property 1 *Given a graph $G = (X, C)$ and a subset T of X such that $G(T)$ is triangulated, then the hypergraph $H_{max}(G(T))$ is acyclic.*

Property 2 *Given a graph $G = (X, C)$, T a subset of X and $\mathcal{C}'$ the set of maximal cliques in $G(T)$, that is $H_{max}(G(T)) = (T, \mathcal{C}')$; if $G(T)$ is triangulated, then $X \setminus T$ is a cycle-cutset of the hypergraph $H_T = (X, \mathcal{C}' \cup E_G(X \setminus T))$.*

Property 3 *Given a graph $G = (X, C)$ and T a subset of X; if $G(T)$ is a maximal TIS of G, then $Y = X \setminus T$ is a minimal cycle-cutset of $H_T = (X, \mathcal{C}' \cup E_G(Y))$ where $\mathcal{C}'$ is the set of maximal cliques in $G(T)$.*

Given a binary CSP $\mathcal{P}_G = (X, D, C, R)$ with a graph $G = (X, C)$, and T a subset of X such that $G(T)$ is a triangulated graph, the CSP induced by T on $\mathcal{P}_G$ is $\mathcal{P}_{H_T} = (X, D, C_T, R_T)$ defined by: $C_T = \mathcal{C}' \cup E_G(Y)$ where $\mathcal{C}'$ is the set of maximal cliques of $G(T)$, that is $H_{max}(G(T)) = (T, \mathcal{C}')$ and $Y = X \setminus T$, and $R_T = \{R_i \in R : C_i \in E_G(Y)\} \cup \{R'_i : \mathcal{C}'_i \in \mathcal{C}'$ and $R'_i = \bowtie_{C_j \subseteq \mathcal{C}'_i} R_j\}$ where the symbol $\bowtie$ denotes the join operator of relational databases theory. Moreover, $H_T = (X, C_T)$. If $G(T)$ is a maximal TIS of G, then $\mathcal{P}_{H_T}$ is called a maximal CSP induced by T on $\mathcal{P}_G$.

We apply this definition to the example given in figure 1. Here, $T = X\backslash\{2,4\}$ and then $C_T = \{\{1,2\},\{1,11\},\{2,3\},\{2,10\},\{3,4\},\{3,9\},\{4,5\},\{4,6\},\{4,8\},\{5,6,7\},\{6,7,8\},\{8,9\},\{9,10\},\{10,11\}\}$. The relations associated to binary constraints correspond to initial relations while new constraints, defined by ternary relations, are obtained by solving associated subproblems. So we have $R_{\{5,6,7\}} = R_{\{5,6\}} \bowtie R_{\{5,7\}} \bowtie R_{\{6,7\}}$ and $R_{\{6,7,8\}} = R_{\{6,7\}} \bowtie R_{\{6,8\}} \bowtie R_{\{7,8\}}$.

Theorem 1 *Let $\mathcal{P}_G = (X,D,C,R)$ be a binary CSP with a graph $G = (X,C)$, T a subset of X such that $G(T)$ is a TIS, and $\mathcal{P}_{H_T} = (X,D,C_T,R_T)$ the CSP induced by T on $\mathcal{P}_G$. The set of solutions of $\mathcal{P}_G$, denoted $Sol_{\mathcal{P}_G}$, is equal to the set of solutions of $\mathcal{P}_{H_T}$.*

Theorem 1 summarizes the method: given a binary CSP $\mathcal{P}_G = (X,D,C,R)$ with graph G, it is sufficient to determine a set T of vertices such that $G(T)$ is a TIS of G. To find the set T, we use the procedure `TRIANG(`G,T`)`. After, we consider $G(T)$ and we compute its maximal cliques $\mathcal{C}'$ thanks to an appropriate procedure `CliquesMaxTriangulated(`$G(T),\mathcal{C}'$`)`. At the next step, we generate the CSP induced by T on $\mathcal{P}_G$, $\mathcal{P}_{H_T} = (X,D,C_T,R_T)$ by solving each subproblem corresponding to each clique in $\mathcal{C}'$. We denote this procedure `Generate(`$\mathcal{P}_G,T,\mathcal{C}',\mathcal{P}_{H_T}$`)`. Finally, since the subset $X \setminus T$ is a minimal cycle-cutset of the hypergraph H_T, we can apply the general cycle-cutset method to solve $\mathcal{P}_{H_T}$, using the procedure `CycleCutsetMethod(`$\mathcal{P}_{H_T}, X\setminus T, Sol_{\mathcal{P}_G}$`)`.

```
1. CC($\mathcal{P}_G, Sol_{\mathcal{P}_G}$)
2. Begin
3.      TRIANG($G,T$);
4.      CliquesMaxTriangulated($G(T),\mathcal{C}'$);
5.      Generate($\mathcal{P}_G,T,\mathcal{C}',\mathcal{P}_{H_T}$);
6.      CycleCutsetMethod($\mathcal{P}_{H_T}, X\setminus T, Sol_{\mathcal{P}_G}$)
7. End
```

Figure 2. **Cyclic Clustering.**

Theorem 2 *The procedure* `CC` *is sound.*

Theorem 3 *The time complexity of* `CC` *is $O(e + n.a.d^{a+k})$ while its space complexity is $O(n.s.d^s)$ where s is the size of the largest intersection between two clusters, that is the maximal size between minimal separators in the TIS and k the cycle-cutset size.*

Note that the first step of this approach allows to compute the complexity of parameters a and k in O(n+e) and then gives a new measure of the complexity of a CSP whose computation can be made in linear time. Another advantage of this approach of CC is that the total height of backtracking is decomposed in two different parts: one for solving subproblems (parameter a), and the other one for solving the cycle-cutset (parameter k). Finally, we see that time complexity is bounded by $exp(a + k)$.

4. Theoretical comparisons

We consider here W and S for TC, K for CCM and a, s and k for CC. Though optimal values for W, S, K, a, s and k are difficult to obtain in practice because all the associated optimization problems are NP-hard, the analysis proposed here takes into account the optimal values of these parameters.

Firstly, it is clear that $s \leq S$. Note that it is easy to find some examples of CSPs such that $k + a \ll K$. More generally and formally, the first result allows us to claim that CC theoretically outperforms CCM for time complexity while the second theorem indicates that given a CC decomposition, we can obtain an equivalent TC decomposition w.r.t. time complexity.

Theorem 4 *Given a binary CSP, if there exists a cycle-cutset of size K, then there exists a CC decomposition with a and k satisfying $k + a \leq K + 2$.*

Theorem 5 *Given a binary CSP and a CC decomposition with parameters a and k, then there is a TC decomposition with parameter W satisfying $W = k + a$.*

Note that for space complexity, CCM is better than CC since the required space is limited to $O(n)$ while the next theorem allows us to affirm that CC decomposition always outperforms TC decomposition considering worst-case space complexity.

Theorem 6 *Given a binary CSP $\mathcal{P}_G = (X,D,C,R)$, for all cyclic-clustering decomposition with parameters a and k, and for all tree-clustering decomposition with parameter W, we have $a \leq W$.*

Note that the comparison with other structural methods is easy from a theoretical viewpoint. Unfortunately, from a practical viewpoint, CCM and TC have seldom shown their interest for solving hard problems (even if the theoretical parameters seem interesting).

5. Implementing cyclic-clustering

Because of the limited practical interest of TC, it is natural to implement CC by replacing TC, after the assignment of the cycle-cutset, by BTD whose efficiency is better than TC's one. So, we will show here how it is possible to exploit BTD and we present an optimization of its use in CC.

For implementing CC, a natural approach should consist in running BTD when the cycle-cutset has been assigned. This approach, denoted **CC-BTD**$_1$, is clearly possible and guarantees the complexity bounds given for CC. Nevertheless, this approach is not necessarily the most efficient. Indeed, each time CC has consistently assigned the cycle-cutset, it must run BTD for computing and recording goods and nogoods again. But it is clear that a part of this processing can be avoided. Let us consider a consistent assignment of the cycle-cutset. BTD will compute and record goods and nogoods which are related to the considered assignment. Indeed, some nogoods are obtained because the assignment of a particular variable of the cycle-cutset causes the inconsistency of a part of assignments of future variables. For another assignment of the cycle-cutset, the same assignment of the same variable can now give a good. So, given a new consistent assignment of the cycle-cutset, it seems impossible to exploit the goods and nogoods produced by a previous call of BTD. Nevertheless, it is possible to exploit the nogoods produced by a preliminary call of BTD. We will denote this second approach as **CC-BTD**$_2$. Now, before running CC, we first consider a call to BTD. It is clear that all the recorded nogoods correspond to assignments which cannot be extended to a complete consistent assignment, in particular including variables belonging to the cycle-cutset. So, these nogoods can be exploited for all consistent assignments of the cycle-cutset to cut search. On the other hand, computed goods cannot be exploited directly because they have been computed on subproblems which are not necessarily compatible with any assignment of the cycle-cutset.

Before providing experiments about these two approaches, note that the integration of BTD in CC offers already a theoretical interest:

Theorem 7 *Given a CSP $\mathcal{P} = (X, D, C, R)$ and a cyclic-clustering decomposition with a TIS $G(T)$ and parameters a and k, the time complexity of **CC-BTD**$_i$ $(i = 1, 2)$ is $O(e+n.a.d^{a+k})$ while its space complexity is $O(n.s.d^s)$ where s is the size of the largest intersection between two maximal cliques of $G(T)$.*

First, we must note that the efficiency of CC mostly depends on the number of solutions of the cycle-cutset. Indeed, in the worst case, CC computes all the solutions of the cycle-cutset. So, for efficiency reasons, CC needs a cycle-cutset with few solutions, what raises the question about the computation of a suitable cycle-cutset. A solution may consist in computing a MTIS thanks to the Balas and Yu's algorithm [1]. By so doing, we obtain cycle-cutsets with a reasonable size, but they often have a lot of solutions. So, we have preferred build a larger but more constrained cycle-cutset which has fewer solutions by using a heuristic method. This method consists in choosing a variable, removing it from the constraint graph and repeating this process until the graph is triangulated. The selected variables form the cycle-cutset. To limit the number of solutions of the cycle-cutset, at each step, we choose the variable which shares most constraints with previously selected variables.

We first experiment CC-BTD$_i$ on binary classical random instances and we compare them with Forward-Checking [8], MAC [12] and FC-BTD [10]. For ordering variables in FC, MAC or inside a cluster for FC-BTD and CC-BTD$_i$, we use the well-known dom/deg heuristic which first chooses the variable x_i which minimizes the ratio $\frac{|d_{x_i}|}{|\Gamma_{x_i}|}$ with d_{x_i} the current domain of x_i and Γ_{x_i} the set of the variables sharing a constraint with x_i. The classical random instances are produced thanks to the generator written by D. Frost et al. Considered classes are close to the satisfiability threshold. For each class, we solve 50 instances whose constraint graph is connected. In spite of the absence of suitable properties, we do not observe a degradation of performance with respect to FC or FC-BTD. Details of these experiments are reported in [11].

As it is well known that classical random instances do not have a particular structure, we assess the interest of our method on binary structured random instances. By structured instances, we mean that the constraint graph of produced instances presents some suitable properties for CC-BTD$_i$. Formally, our random generator takes 8 parameters. It builds a CSP of class $(n, d, a, t, s, k, e_1, e_2)$ with $n + k$ variables, each having a domain of size d. For each constraint, t tuples are forbidden. We first build a clique-tree (like in [10]) with a the size of the largest clique and s the size of the largest intersection between two cliques. This clique-tree contains n variables. Note that a clique-tree is a TIS. Then, we build the cycle-cutset which contains k variables and e_1 constraints. Finally, we add e_2 constraints between the cycle-cutset and the clique-tree. By so doing, produced problems have a suitable structure. Unfortunately, we do not have any method to recognize such a structure. So, as we aim to assess the interest of CC-BTD$_i$ for structured problems and not the quality of the computed cycle-cutset, we provide the

Class	FC	MAC	FC-BTD	CC-BTD$_1$	CC-BTD$_2$
(50,15,15,75,5,15,80,50)	42.00	89.98	10.14	4.76	3.35
(50,15,15,76,5,15,80,50)	24.68	85.14	23.62	3.35	2.44
(50,15,15,71,5,20,130,50)	46.42	159.96	40.70	0.35	0.35
(50,15,15,72,5,20,130,50)	211.36	699.47	128.53	0.29	0.30
(50,15,15,77,5,15,80,30)	1.89	8.64	1.58	4.89	1.49
(50,15,15,78,5,15,80,30)	39.98	89.56	1.51	3.45	2.15

Table 1. Structured random instances.

cycle-cutset to CC-BTD$_i$. Selected classes are close to the satisfiability threshold. For each class, we solve 50 instances whose constraint graph is connected.

Table 1 shows that CC-BTD$_i$ may outperform MAC, FC and FC-BTD when the problem structure has the suitable properties. Note that this structure is not really suitable for FC-BTD, what may explain the results obtained by FC-BTD. In particular, the cluster size for FC-BTD is significantly more important than one for CC-BTD$_i$. For instance, the largest cluster of FC-BTD often involves about 50 variables while CC-BTD$_i$'s one contains at most 15 variables. We can also observe that CC-BTD$_2$ is better than CC-BTD$_1$. So the nogoods recorded during the preliminary call of BTD allow CC-BTD$_2$ to prune some parts of the search space.

To sum up, CC-BTD$_i$ may be an interesting approach for solving structured problems. The main difficulty lies in recognizing the problem structure, that is computing a good cycle-cutset with only few solutions. So, methods which compute such cycle-cutsets must be developed before we can fully assess the practical interest of CC-BTD$_i$ (for instance, by solving real-world instances).

6. Conclusion

In this paper, we have studied the Cyclic-Clustering decomposition method [9]. While [9] only presented the principles of the method, we have explained here how this method can be made operational by exploiting good properties of triangulated induced subgraphs. Then, we have shown that Cyclic-Clustering proposes a time-space trade-off w.r.t. theoretical complexities. Indeed, we have proved that its time complexity is less than one of the Cycle-Cutset scheme while its space complexity is less than Tree-Clustering's one. Finally, we have presented some preliminary experiments which show that Cyclic-Clustering based on an hybrid adaptation of Tree-Clustering called BTD [10] may be efficient in practice as soon as the instance to solve has the suitable structural properties. From a practical viewpoint, many works must be developed, principally in two directions. Firstly, we must study algorithmic tools for finding better TIS. The second direction concerns a better use of informations produced by BTD during the preliminary phase of CC-BTD2.

References

[1] E. Balas and C. Yu. Finding a maximum clique in an arbitrary graph. *Siam J. on Computing*, 15(4):1054–1068, 1986.

[2] R. Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41:273–312, 1990.

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

[4] R. Dechter and Y. E. Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.

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

[6] E. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29:24–32, 1982.

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

[8] R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[9] P. Jégou. Cyclic-Clustering: a compromise between Tree-Clustering and the Cycle-Cutset method for improving search efficiency. In *Proceedings of ECAI-90*, pages 369–371, 1990.

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

[11] P. Jégou and C. Terrioux. A time-space trade-off for constraint networks decomposition. Technical report, LSIS, 2004. Available at www.lsis.org.

[12] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of ECAI-94*, pages 125–129, 1994.