LSIS Research Report (UMR CNRS 6168) Marseille France

A time-space trade-off for constraint networks decomposition Research Report Number LSIS/2004/005, July 1st 2004

Philippe Jégou and Cyril Terrioux

LSIS - Université d'Aix-Marseille 3
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, cyril.terrioux}@univ.u-3mrs.fr

Abstract
We study here a CSP decomposition method introduced in [1] and called Cyclic-Clustering. Its purpose was to make a trade-off between two decomposition methods by exploiting their respective advantages. The first one was the Cycle-Cutset scheme [2] which does not require a lot of space memory, while the second was Tree-Clustering [3] which is theoretically more efficient for time complexity but is inefficient in practice due to its space requirement.
While [1] 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. Indeed, we prove here 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 present some preliminary experiments which show that Cyclic-Clustering based on an hybrid adaptation of Tree-Clustering called BTD [4] may be efficient in practice.

Key words : Constraint satisfaction problem, decomposition, time-space trade-off

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 complexity is then exponential in the size of the instance. Nevertheless, theoretical results as presented in [5] 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 topological properties of constraint networks which are then represented by graphs (or hypergraphs). For example, in [2, 3], two decomposition strategies have been proposed, the *Cycle-Cutset method* (CCM) and the *Tree-Clustering scheme* (TC). Nevertheless, while their theoretical interest seems significant (see [6] for an analysis), their practical advantages have not been proved yet.
Firstly in [1] and recently in [7], trade-offs between time and space complexity have been proposed to make this class of approaches usable. Moreover, in [4], 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 [1]). Note that [1] only describes the motivations and principles of this method. For example, it does not provide any indication for using the method in practice. Moreover, neither theoretical nor practical analysis of this method was given in [1]. So, with regard to recent developments of approaches which exploit time-space trade-off, it seems interesting to study CC now.
The first step of this method consists in finding all maximal cliques of the initial constraint network (a graph). This approach can have a limited interest because the number of maximal cliques in a constraint

network can be exponential in the number of vertices [8]. So, exploiting in practice CC requires an additional study. While TC runs by approximating a tree-decomposition of the constraint network, our approach of CC proposes to find an induced subgraph of the considered constraint network whose structure is already tree-structured. So, this part of the CSP has desirable properties and then can be easily solved. Moreover, a nice property of hypergraphs allows us to establish a new equivalent CSP whose a part is tree-structured while the other one forms a cycle-cutset. This analysis of the network is based on the properties of triangulated graphs, and more precisely, on properties of triangulated induced subgraphs.

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 [4] 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 give 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.

2. Constraint networks and decomposition methods

2.1. Constraint satisfaction problems

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 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 known to be NP-complete.

Generally, CSPs are solved by different versions of backtrack search whose complexity is then exponential in the size of the instance. Consequently, many works try to improve the search efficiency. They mainly deal with binary CSPs. In [5], 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. This condition concerns on the one hand a structural property of the constraint graph, and on the other hand a local consistency. The most interesting property is the next one:

Theorem 1 ([5]) *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 [9]).*

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 the cycle-cutset method [2] and tree-clustering [3] which are presented in the next subsections.

2.2. Tree-clustering (TC)

Among the decomposition methods based on a tree-decomposition [10] of the constraint network, we have chosen, for sake of simplicity and without loss of generality, to describe TC. However, this work can be applied to other decomposition methods based on tree-decomposition [6, 11]. TC consists in 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. So, the property about acyclic CSPs holds for this CSP. To present TC and after cyclic-clustering, we need to recall some definitions on graph and hypergraph theory:

Definition 1 *The primal graph of a hypergraph (X, C) is the graph (X, E) with $E = \{\{X_i, X_j\} \mid \exists C_k \in C \text{ s.t. } \{X_i, X_j\} \subseteq C_k\}$.*

Definition 2 *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:

Definition 3 *A hypergraph is conformal if every maximal clique of its primal graph corresponds to an edge in the original hypergraph (an original constraint).*

Definition 4 ([12, 13]) *A hypergraph is acyclic if it is conformal and its primal graph is triangulated.*

The TC procedure involves several steps. The first one is based on the triangulation of the primal graph. This operation, which is run if this graph is not already triangulated, adds new edges to generate a triangulated primal graph [14]. The second step identifies the maximal cliques $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_m\}$ in the triangulated primal graph [15]. The next step establishes an acyclic hypergraph whose edges are the maximal cliques of the new primal graph (so, this hypergraph has a triangulated primal graph and is conformal). The time complexity for these different steps is $O(n + e')$ where n is the number of vertices and e' the number of edges in the triangulated graph. Then, we define the resultant CSP by solving the subproblems defined by the clusters $\mathcal{C}'_1, \ldots, \mathcal{C}'_m$ using the constraints belonging to each cluster, i.e. listing all the consistent tuples for the variables in each cluster. The time complexity for these different steps is $O(m.d^W)$ where d is 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 in the triangulated primal graph). Finally, we obtain a globally consistent CSP using a pairwise-consistent filtering [9] which is equivalent to executing directional arc-consistency on tuples; its time complexity is $O(n.d^W.W.log(d))$. Note that its space complexity is $O(m.d^W)$ but can be limited to $O(m.d^S)$ (like, for instance, Cluster Tree Elimination [11]) where S is the size of the largest intersection between two clusters, that is 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 [11]. In the sequel, we will call TC methods based on tree decomposition as TC [3], Join Tree Elimination [11] or Cluster Tree Elimination [11].
While these methods seem to be the most efficient methods to solve binary CSPs w.r.t. their time complexity (see [6]), their practical interest is really limited. The reason is related to the fact that the time computation is too expensive and the required memory size too large in practical cases. So, in [4], an alternative approach called BTD has been proposed which limits the size of the required memory and also avoids some redundancies in the search. While TC computes all solutions on each cluster and memorizes the corresponding assignments on the intersections between clusters, BTD only computes a part of these solutions and so only records some assignments on the intersections. These assignments are called *goods* if they can be extended to consistent assignments on the part of the CSP which is connected to this intersection, and *nogoods* if they cannot. So, as for other tree-decomposition methods based on TC, the space complexity of BTD is limited since it is bounded by $O(m.d^S)$ but this worst case is not generally achieved in practice thanks to the recorded goods and nogoods. With this trade-off, BTD can solve instances than TC nor its optimizations can solve.

2.3. The cycle-cutset method (CCM)

A cycle-cutset of a graph (respectively hypergraph) is a set of vertices such that the deletion of these vertices induces an acyclic graph (resp. acyclic hypergraph). CCM [2] is based on the fact that assigned variables change the effective connectivity of the constraint graph. So, as soon as all the variables of the cycle-cutset are assigned, all the cycles of the constraint graph are cut. Therefore, the resulting problem is acyclic and the theorem 1 can be applied.
So, if we look for a solution, we can consider that the size of cycle-cutset corresponds to the height of the backtracking. More precisely, for binary CSP, time complexity of CCM is $O(e.d^{K+2})$ where e is the number of constraints and K the size of cycle-cutset. The cost for looking for a consistent assignment in the cycle-cutset is $O(d^K)$. For every solution in the cycle-cutset (i.e. a consistent assignment), it is

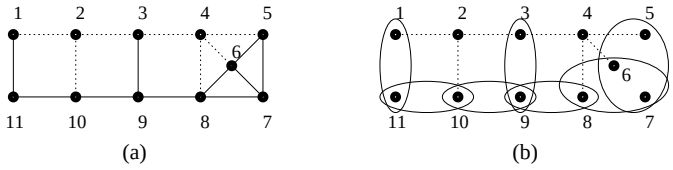

Figure 1: Example of cyclic-clustering decomposition.

necessary to solve the induced acyclic CSP. The time complexity is then $O((n-K).d^2)$, therefore, we obtain globally $O(e.d^{K+2})$. We can note that the most important parameter in this complexity is K, and the optimization problem which consists in minimizing K is NP-hard [16]. This method can be extended to non-binary CSPs and then its time complexity 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 [17]. Note that after the assignment of the cycle-cutset, the induced acyclic CSP can be solved using the same kind of procedure than TC.

2.4. Comparison of the methods

The most important advantage of TC concerns the form of the result: it is possible to obtain every solution without backtracking after the preprocessing phase. On the contrary, the same possibility does not exist for CCM: each solution must be processed with a backtrack search bounded by the size of the cycle-cutset.

Firstly in [18] and more recently in [6], 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. Moreover, neither TC nor CCM has shown their practical interest. As indicated in [4], 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)

3.1. Principles of the method

To avoid drawbacks of TC and CCM, an alternative approach called *cyclic-clustering* (CC) has been introduced in [1]. Note that [1] 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 [8] 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 summarizes 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. In this example, these vertices are vertices 2 and 4. Theoretical justifications of this approach are based on properties of TIS.

Definition 5 *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, that is if $\nexists T'$ such that $T \subsetneq T' \subseteq X$ and $G(T')$ is triangulated.*

In [19], 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. This is not surprising because the task of finding such a maximum subgraph is a NP-hard problem [16, 20]. 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. Based on properties of acyclic hypergraph, 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.

Definition 6 *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.*

Definition 7 *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.*

Proof: By definition $H_{max}(G(T))$ is conformal because it is defined by the maximal cliques of its primal graph $G(T)$. Since $G(T)$ is triangulated and $H_{max}(G(T))$ is conformal, by definition of acyclicity, $H_{max}(G(T))$ 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 $H_T = (X, \mathcal{C}' \cup E_G(X \setminus T))$.*

Proof: Since no edge in $E_G(X \setminus T)$ is included in T, and since $H_{max}(G(T)) = (T, \mathcal{C}') = H_T(T)$ is acyclic, it is clear that $X \setminus T$ is a cycle-cutset of H_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 $G(T)$.*

Proof: We prove this proposition by contradiction: If $Y = X \setminus T$ is not a minimal cycle-cutset of H_T, necessarily, there exists a subset Y' of Y such that $Y \neq Y'$ and Y' is a cycle-cutset of H_T. Consequently, if $T' = X \setminus Y'$, then $H_T(T')$ is an acyclic hypergraph, and its primal graph is triangulated. Therefore $G(T')$ is triangulated and then $G(T)$ is not a maximal TIS of G. □

Definition 8 *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* $Y = X \setminus T$
- $R_T = \{R_i \in R : C_i \in E_G(Y)\} \cup \{R'_i : \mathcal{C}'_i \in \mathcal{C}'$ *and* $R'_i = \bowtie_{C_j \subseteq \mathcal{C}'_i} R_j\}$ *where the symbol* $\bowtie$ *denotes the join operator of relational databases theory.*
- $H_T = (X, C_T)$.

If $G(T)$ *is a maximal TIS of* G*, then* $\mathcal{P}_{H_T}$ *is called the maximal CSP induced by* T *on* $\mathcal{P}_G$.

We apply this definition to the example given in figure 1. Here, $T = X \setminus \{2, 4\}$ 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 $R_{\{6,7,8\}} = R_{\{6,7\}} \bowtie R_{\{6,8\}} \bowtie R_{\{7,8\}}$.
Actually, the maximal induced CSP $\mathcal{P}_{H_T}$ has exactly the same set of solutions as $\mathcal{P}_G$.

Theorem 2 *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* $\mathcal{P}_{H_T}$*, denoted* $Sol_{\mathcal{P}_{H_T}}$.

Proof: By definition, $Sol_{\mathcal{P}_G} = \bowtie_{C_i \in C} R_i$ and since $C = E_G(Y) \cup (C \setminus E_G(Y))$, we have $Sol_{\mathcal{P}_G} = (\bowtie_{C_i \in E_G(Y)} R_i) \bowtie (\bowtie_{C_i \in (C \setminus E_G(Y))} R_i)$.
Then $Sol_{\mathcal{P}_G} = (\bowtie_{C_i \in E_G(Y)} R_i) \bowtie (\bowtie_{\mathcal{C}'_i \in \mathcal{C}'} (\bowtie_{C_j \subseteq \mathcal{C}'_i} R_j))$ where $\mathcal{C}'$ is defined as in definition 8, because $\cup_{C_i \in (C \setminus E_G(Y))} \{C_i\} = \cup_{\mathcal{C}'_i \in \mathcal{C}'} (\cup_{C_j \subseteq \mathcal{C}'_i} \{C_j\})$
Finally, $Sol_{\mathcal{P}_G} = (\bowtie_{C_i \in E_G(Y)} R_i) \bowtie (\bowtie_{\mathcal{C}'_i \in \mathcal{C}'} (\bowtie_{C_j \subseteq \mathcal{C}'_i} R_j)) = \bowtie_{C_{T_i} \in C_T} R_{T_i} = Sol_{\mathcal{P}_{H_T}}$. □

3.2. The method

Theorem 2 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. CyclicClustering(P_G, Sol_{P_G})
2. Begin
3.     TRIANG(G, T);
4.     CliquesMaxTriangulated(G(T), C');
5.     Generate(P_G, T, C', P_{H_T});
6.     CycleCutsetMethod(P_{H_T}, X \ T, Sol_{P_G})
7. End
```

Figure 2: Cyclic Clustering.

Theorem 3 *Given a binary CSP* $\mathcal{P}_G$*, the procedure* `CyclicClustering` *computes* $Sol_{\mathcal{P}_G}$.

Proof: By theorem 2, we know that $\mathcal{P}_G$ and $\mathcal{P}_{H_T}$ have the same set of solutions. Thus, by property 3, we know that the subset $X \setminus T$ is a minimal cycle-cutset of the hypergraph H_T. So, applying the cycle-cutset method to solve $\mathcal{P}_{H_T}$ with the cycle-cutset $X \setminus T$ produces the set of solutions $Sol_{\mathcal{P}_G}$. □

Theorem 4 *The time complexity of* `CyclicClustering` *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.*

Proof: We analyze each step, giving their time and space complexity:

- `TRIANG`(G, T). Its time complexity is $O(n + e)$. [19]
- `CliquesMaxTriangulated`$(G(T), \mathcal{C}')$. The time complexity of this algorithm is $O(n + e)$ [15] and we can give a bound for $|\mathcal{C}'|$ with $|T|$ [21].
- `Generate`$(\mathcal{P}_G, T, \mathcal{C}', \mathcal{P}_{H_T})$. The difficulty is in the computation of R_T. Indeed, it is necessary to solve all the subproblems associated to the maximal cliques of $G(T)$, i.e. edges of C_T that do not belong to $E_G(X \setminus T)$. For each R_{T_i}, the cost is $O(d^{a_i})$ if $a_i = |C_{T_i}|$. Consequently, the time complexity for this step is $O(|C_T|.d^a)$ if $a = max_{C_{T_i} \in C_T}|C_{T_i}|$. Space complexity is then $O(|C_T|.a.d^a)$.
- `CycleCutsetMethod`$(\mathcal{P}_{H_T}, X \setminus T, Sol_{\mathcal{P}_G})$. It is exactly the extended method of cycle-cutset for non-binary CSPs. If $k = |X \setminus T|$, the cost of searching the consistent assignment on $X \setminus T$ is $O(d^k)$. So, we obtain the time complexity $O(|\mathcal{C}'|.d^a.log(d^a).d^k)$, that is $O(|\mathcal{C}'|.a.d^a.d^k) = O(|\mathcal{C}'|.a.d^{a+k})$.

Finally, we obtain $O(n + e + |\mathcal{C}'|.d^a + |\mathcal{C}'|.a.d^{a+k})$; since $|\mathcal{C}'| \leq |T| \leq n$, we can give a bound to this expression: $O(e + n.a.d^{a+k})$ for time complexity while it is $O(n.s.d^s)$ for space complexity. □

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 complexity is bounded by $exp(a + k)$.
To apply efficiently this method, we must solve an optimization problem since the first step serves to compute the most important parameters of the complexity: a and k. So, it can be interesting to minimize k, that is equivalent to maximize $|T|$, searching to minimize a. It seems to us that the algorithm `TRIANG` may not be the best approach to realize that; however, we do not know any result on this optimization problem.

4. Relations between a, s, k and other structural parameters

In this section, we present a comparison between CC and other decomposition methods, namely TC and CCM. We prove that the time complexity of CC is less than one of the Cycle-Cutset scheme while its space complexity is less than Tree-Clustering's one.
So, 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 optimal values for 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 cyclic-clustering theoretically outperforms CCM for time complexity.

Theorem 5 *Given a binary CSP* $\mathcal{P}_G = (X, D, C, R)$*, if there exists a cycle-cutset of size* K*, then there exists a cyclic-clustering decomposition with parameters* a *and* k *satisfying* $k + a \leq K + 2$.

Proof: Consider a cycle-cutset Y of the graph $G = (X, C)$. The induced subgraph $G(X \setminus Y)$ is an acyclic graph, precisely a forest since Y is a cycle-cutset (the size of the set Y is K). Since the subgraph $G(X \setminus Y)$ is a triangulated graph (a forest is a triangulated graph), this subgraph allows us to define a basic cyclic-clustering decomposition. Nevertheless, $X \setminus Y$ is not necessarily a maximal TIS. Consider T such that $T \supset (X \setminus Y)$ and $G(T)$ is a maximal TIS. Since the forest $G(X \setminus Y)$ is a subgraph of $G(T)$, necessarily, the maximum size a for maximal cliques in $G(T)$ is bounded by $2 + |T \setminus (X \setminus Y)|$ because

2 is the maximal size for maximal cliques in $G(X \setminus Y)$, and because added vertices - that are vertices appearing in $T \setminus (X \setminus Y)$ - do not necessarily belong to the same clique of $G(T)$. Moreover, the value k is exactly the number of vertices that do not appear in T, so $k = |X \setminus T|$. We obtain the inequality $a + k \leq 2 + |T \setminus (X \setminus Y)| + |X \setminus T|$. Since $X \setminus Y$ is a subset of T, T is a subset of X, and Y a subset of X, this inequality can be rewritten as $a + k \leq 2 + |T| - |X \setminus Y| + |X| - |T|$, and since $|X \setminus Y| = n - K$ and $|X| = n$, we obtain finally $a + k \leq 2 + K$. □

Note that for space complexity, CCM is better than CC since the required space is limited to n.
The next theorem allows us to affirm that cyclic-clustering decomposition always outperforms tree-clustering decomposition considering worst-case space complexity. Its proof is based on the fact that the value W is always greater than the size of the clique of maximum size in a graph.

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 $a \leq W$.*

The next theorem is clearly more interesting since it establishes the link between the theoretical time complexities of TC and CC.

Theorem 7 *Given a binary CSP $\mathcal{P}_G = (X, D, C, R)$, if there is a cyclic-clustering decomposition with parameters a and k, then there is a tree-clustering decomposition with parameter W satisfying $W = k + a$.*

Proof: Based on the cyclic-clustering decomposition, we exhibit a tree-clustering decomposition with the parameter W satisfying $W = k + a$. Consider a cyclic-clustering decomposition with set of vertices T, and parameters a and k. That is T is a subset of vertices such that the graph $G(T)$ is a TIS of the graph $G = (X, C)$ such that a is the maximum size for the maximal cliques of $G(T)$ and $|X \setminus T| = k$. Associated to $G(T)$, the set of maximal cliques is $\mathcal{C}' = \{\mathcal{C}'_1, \ldots \mathcal{C}'_m\}$. Consider the set $Y = X \setminus T$. Based on Y and $\mathcal{C}'$, we can define a triangulation of G, by adding a set of edges E to C, defined as:

$$E = (\cup_{1 \leq i \leq m} \{\{x, y\} : x \in \mathcal{C}'_i \text{ and } y \in Y\}) \bigcup \{\{x, y\} : x, y \in Y\}$$

That is, for every maximal clique $\mathcal{C}'_i$ in $\mathcal{C}'$, we add edges allowing to define new maximal cliques defined as $\mathcal{C}'_i \cup Y$. Then, we can prove that the graph $(X, C \cup E)$ is triangulated and that $\{\mathcal{C}'_1 \cup Y, \ldots, \mathcal{C}'_m \cup Y\}$ is the set of its maximal cliques. Consequently, we can consider a tree-clustering decomposition based on the sets $\mathcal{C}'_1 \cup Y, \ldots, \mathcal{C}'_m \cup Y$ where every set $\mathcal{C}'_i \cup Y$ is a cluster of variables. For this decomposition, parameter W is exactly equal to $a + k$ since the maximum size for the maximal cliques $\mathcal{C}'_i$ in $G(T)$ is a, and since k is the size of the set $Y = X \setminus T$. □

The proof of this theorem indicates a way to optimize TC using cyclic-clustering decomposition. Suppose that for a given CSP and for an application of the two decompositions, we have W, a and k such that $k + a < W$. By applying the principle described in the proof, we can define a new tree-clustering decomposition such that $k + a = W$. It is sufficient to add, in every maximal clique induced by the TIS, all the vertices appearing in the cycle-cutset.
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). Moreover, if we consider other trade-offs methods as [7] which use TC and CCM in another way (w.r.t. CC), no practical validation has been given. Indeed, only the values of the structural parameters are computed and presented by the authors.

5. Implementing cyclic-clustering with BTD

This section reports preliminary practical results on CC. Our first objective is to assess the interest of CC. So we analyze it on two kinds of instances. On the one hand, we check if applying CC on problems which do not have a particular structure degrades or not its performances. On the other hand, we show than for well structured problems, CC may significantly outperform other approaches.

5.1. Two natural approaches: CC-BTD$_1$ and CC-BTD$_2$

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 8 *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 $G(T)$.*

5.2. Experimental results

First, before presenting some preliminary empirical results, 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 [19]. 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 [22], MAC [23] and FC-BTD [4]. For ordering variables in FC, MAC or inside a cluster for FC-BTD and CC-BTD$_i$, we use the *dom/deg* heuristic[24]. This heuristic 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, C. Bessière, R. Dechter and J.-C. Régin. This generator takes 4 parameters n, d, e and t. It builds a CSP of class (n, d, e, t) with n variables, each having a domain of size d, and e binary constraints ($0 \leq e \leq \frac{n(n-1)}{2}$) in which t tuples are forbidden ($0 \leq t \leq d^2$). Considered classes are close to the satisfiability threshold. For each class, we solve 50 instances whose constraint graph is connected.

Table 1 presents the mean run-time of each method. We can remark that CC-BTD$_1$ and CC-BTD$_2$ obtain similar results. Such a result is not surprising because, like observed in [4], few structural nogoods are produced on classical random instances. So, few nogoods are exploited too, what explains that CC-BTD$_i$ does not perform better than FC or FC-BTD. However, it outperforms MAC when FC appears better

Class	FC	MAC	FC-BTD	CC-BTD$_1$	CC-BTD$_2$
(50,15,245,93)	7.87	12.36	7.61	8.22	8.21
(50,15,306,78)	28.06	55.76	27.19	30.10	29.97
(50,15,368,68)	97.20	232.95	92.72	105.08	104.28
(50,25,150,397)	2.48	2.66	2.49	2.61	2.61
(75,10,277,43)	1.45	0.65	1.44	1.59	1.59

Table 1: Run-time (in seconds) of FC, FC-BTD, CC-BTD$_1$ and CC-BTD$_2$ for classical random instances.

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 2: Run-time (in seconds) of FC, FC-BTD, CC-BTD$_1$ and CC-BTD$_2$ for structured random instances.

than MAC. Moreover, in spite of the absence of suitable properties, we do not observe a degradation of performance with respect to FC or FC-BTD.

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 [4]) with a the size of the largest clique and s the size of the largest intersection between two cliques. 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 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 2 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. Discussion and Conclusion

In this paper, we have studied the Cyclic-Clustering decomposition method [1]. While [1] only presented the principles of the method, we have explained here how this method can be made operational by ex-

ploiting 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 [4] may be efficient in practice as soon as the instance to solve has the suitable structural properties.

The comparison with other decomposition methods is not easy. From a theoretical viewpoint, the methods based on TC clearly outperform CC w.r.t. the time complexity. Nevertheless, if we analyze published papers on this topic (see [11] for a complete survey), no practical validation of these approaches has been made, but for BTD. For example, in [7] where another approach of a trade-off between TC and CCM is presented, no practical validation is given. Indeed the authors limit their analysis of the practical interest to an empirical assessment of structural parameters as S and W without solving any instance. Note that in [7], CCM is only used in TC, instead of any enumerative method, for solving the subproblems associated to clusters. Moreover, for other papers on TC, no study of the practical interest is provided. This is not surprising since, when these methods are implemented, their space requirements prevent them from running on small instances. So, it seems to us that the trade-off offered by CC constitutes a realistic alternative to exploit structural features of well structured instances.

Nevertheless, from a practical viewpoint, many works must be developed, principally in two directions. Firstly, we must study algorithmic tools for finding TIS. Indeed, experiments have shown that the quality of the computed TIS (w.r.t. CC) is not necessary related to its size. The number of solutions of the associated cycle-cutset is involved too. For instance, the algorithm TRIANG finds a maximal TIS with a reasonable size, but, unfortunately, the associated minimal cycle-cutset has too many consistent assignments, what implies a deterioration of the practical efficiency of CC.

The second direction concerns a better use of informations produced by BTD during the preliminary phase of CC-BTD2. Indeed, the nogoods recorded before running CC allow to speed up CC. So, instead of stopping BTD as soon as it finds a solution of the TIS, it should be more interesting to compute all the possible nogoods. Another use of BTD could be to find a way for exploiting the goods produced during the preliminary phase. While finding all nogoods is theoretically easy, using goods requires more theoretical works.

Bibliography

1. P. Jégou. Cyclic-Clustering: a compromise between Tree-Clustering and the Cycle-Cutset method for improving search efficiency. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-90)*, pages 369–371, 1990.
2. R. Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41:273–312, 1990.
3. R. Dechter and J. Pearl. Tree-Clustering for Constraint Networks. *Artificial Intelligence*, 38:353–366, 1989.
4. P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.
5. E. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29:24–32, 1982.
6. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:343–282, 2000.
7. R. Dechter and Y. El Fattah. Topological Parameters for Time-Space Tradeoff. *Artificial Intelligence*, 125:93–118, 2001.
8. I. Tomescu. Sur le nombre de cliques maximales d'un graphe et quelques problèmes sur les graphes parfaits. *Revue roumaine de Mathématiques Pures et Appliquées*, 16:1115–1126, 1970. In french.
9. P. Janssen, P. Jégou, B. Nouguier, and M.C. Vilarem. A filtering process for general constraint satisfaction problems: achieving pairwise-consistency using an associated binary representation. In *Proceedings of IEEE Workshop on Tools for Artificial Intelligence*, pages 420–427, 1989.
10. N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *Algorithms*, 7:309–322, 1986.
11. R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

12. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30:479–513, 1983.
13. C. Berge. *Hypergraphes – Combinatoire des ensembles finis*. Gauthier-Villars, 1987.
14. R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3):566–579, 1984.
15. F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1 (2):180–187, 1972.
16. M. Krishnamoorthy and N. Deo. Node deletion NP-complete problems. *SIAM J. on Computing*, 8(4):619–625, 1979.
17. P. Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution – Propagation de contraintes dans les réseau dynamiques*. PhD thesis, Université des Sciences et Techniques du Languedoc, January 1991. In french.
18. U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.
19. E. Balas and C. Yu. Finding a maximum clique in an arbitrary graph. *Siam J. on Computing*, 15(4):1054–1068, 1986.
20. M. Yannakakis. Node and Edge-deletion NP-complete problems. In *Proceedings ACM Symp. on Theory of Comput.*, pages 253–264, 1978.
21. D.R. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.
22. R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
23. D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-94)*, pages 125–129, 1994.
24. C. Bessière and J.-C. Régin. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems. In *Proceedings of CP'96*, pages 61–75, 1996.