A Comparison of Structural CSP Decomposition Methods

Georg Gottlob
Inst, fur Informatioiissystemc
Technische Universitat Wien
A-1040 Vienna, Austria
gottlob@dbai.tuwien.ac.at

Nicola Leone
Inst, fur Informationssysteme
Technische Universitat Wien
A-1040 Vienna, Austria
leone@dbai.tuwien.ac.at

Francesco Scarcello
ISI-CNR
Via P. Bucci 41/C
1-87030 Rende, Italy
scarcello@si.deis.unical.it

Abstract

We compare tractable classes of constraint satisfaction problems (CSPs). We first give a uniform presentation of the major structural CSP decomposition methods. We then introduce a new class of tractable CSPs based on the concept of *hypertree decomposition* recently developed in Database Theory. We introduce a framework for comparing parametric decomposition-based methods according to tractability criteria and compare the most relevant methods. We show that the method of hypertree decomposition dominates the others in the case of general (nonbinary) CSPs.

1 Constraint Satisfaction Problems

An instance of a *constraint satisfaction problem (CSP)* (also *constraint network)* is a triple / = *(Var,U,C),* where *Var* is a finite set of variables, *U* is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints. Each constraint C_i is a pair (S_i, r_i), where S_i is a list of variables of length rn_i called the *constraint scope,* and r_i is an m_i-ary relation over [/, called the *constraint relation.* (The tuples of r_i indicate the allowed combinations of simultaneous values for the variables *Si).* A *solution* to a CSP instance is a substitution V : $Var \longrightarrow U$, such that for each $1 \leq i \leq q$, $S_i\vartheta \in r_i$. The problem of deciding whether a CSP instance has any solution is called *constraint satisfiability (CS).* (This definition is taken almost verbatim from [Jeavons *et* a/., 1997].)

Many problems in Computer Science and Mathematics can be formulated as CSPs. For example, the famous problem of *graph three-colorability (3COL),* is elegantly formulated as a CSP. Constraint Satisfiability is an NP-complete problem.

It is well-known [Bibel, 1988; Gyssens *et al,* 1994; Dechter, 1992] that the CS problem is equivalent to various database problems, e.g., to the problem of evaluating *Boolean conjunctive queries* over a relational database [Maier, 1986], or to the equivalent problem of evaluating *join dependencies* on a given database.

This paper is organized as follows. In Section 2 we discuss tractability of CSPs due to restricted structure. In Section 3 we briefly review well-known CSP decomposition methods. In Section 4 we describe the new method of *hypertree decompositions.* In Section 5 we explain our comparison criteria and in Section 6 we present the comparison results for general CSPs. The case of binary CSPs is briefly discussed in Section 7.

2 Tractable classes of CSPs

Much effort ,has been spent by both the AI and the database communities to indentify *tractable classes* of CSPs. Both communities have obtained deep and useful results in this direction. The various successful approaches to obtain tractable CSP classes can be divided into two main groups [Pearson and Jeavons, 1997]:

1. Tractability due to restricted structure. This includes all tractable classes of CSPs that are identified solely on the base of the structure of the constraint scopes *{S\,... Sq),* independently of the actual constraint relations r_1,..., r_q.

2. Tractability due to restricted constraints. This includes all classes that are tractable due to particular properties of the constraint relations r_1,..., r_q.

The present paper deals with tractability due to restricted structure. The *structure* of a CSP is best represented by its associated *hypergraph* and by the corresponding *primal graph,* defined as follows.

To any CSP instance / = *(Var, U,C),* we associate a hypergraph $\mathcal{H}_I$ — *(V,H),* where *V* = *Var,* and $H = \{var(S) \mid C = (S, r) \in \mathcal{C}\}$, where *var(S)* denotes the set of variables in the scope *S* of the constraint *C.* Since in this paper we always deal with hypergraphs corresponding to CSPs instances, the vertices of any hypergraph *'H* = (V^r, *H)* can be viewed as the variables of some constraint satisfaction problem. Thus, we will often use the term *variable* as a synonym for vertex, when referring to elements of *V.*

Let $\mathcal{H}_I$ = *(V,H)* be the constraint hypergraph of a CSP instance *1.* The *primal graph* of / is a graph *G* = *(V,E),* having the same set of variables (vertices) as $\mathcal{H}_I$ and an edge connecting any pair of variables $X, Y \in V$

such that $\{X, Y\} \subseteq h$ for some $h \in H$.

Note that if all constraints of a CSP are binary, then its associated hypergraph is identical to its primal graph.

The most basic and most fundamental structural property considered in the context of CSPs (and conjunctive queries) is *acyclicity.* It was recognized independently in AI and in database theory that *acyclic* CSPs are polynomial^ solvable. / is an acyclic CSP iff its primal graph *G* is chordal (i.e., any cycle of length greater than 3 has a chord) and the set of its maximal cliques coincide with *edges (Hl)* [Beeri *et* a/., 1983].

A *join tree JT(H)* for a hypergraph *H* is a tree whose nodes are the edges of *H* such that whenever the same vertex $X \in V$ occurs in two edges *A* and A_2 of *H,* then A_1 and A_2 are connected in $JT(\mathcal{H})$, and *X* occurs in each node on the unique path linking A_1 and A_2 in $JT(\mathcal{H})$

Acyclic hypergraphs can be characterized in terms of join trees: A hypergraph *H* is *acyclic* iff it has a join tree [Bernstein and Goodman, 1981; Beeri *et* a/., 1983; Maier, 1986]. Acyclic CSP satisfiability is not only tractable but also highly parallelizable. In fact, this problem is complete for the low complexity class LOGCFL [Gottlob *et* a/., 1998].

Many CSPs arising in practice are not acyclic but are in some sense or another *close* to acyclic CSPs. In fact, the hypergraphs associated with many naturally arising CSPs contain either few cycles or small cycles, or can be transformed to acyclic CSPs by simple operations (such as, e.g., lumping together small groups of vertices). Consequently, CSP research in AI and in database theory concentrated on identifying, defining, and studying suitable classes of *nearly acyclic* CSPs, or, equivalently, methods for decomposing cyclic CSPs into acyclic CSPs.

3 Decomposition Methods

In order to study and compare various decomposition methods, we find it useful to introduce a general formal framework for this notion.

For a hypergraph *H* = (V, *H),* let *edges(H)* = *H.* Moreover, for any set of edges $H' \subseteq H$, let *var(H')* = $\bigcup_{h \in H'} h$; and for the hypergraph 'H, let *var(H)* — $var(H)$. W.l.o.g., we assume that *var(H)* = V, i.e., every variable in *V* occurs in at least one edge of *7i,* and hence, any hypergraph can be simply represented by the set of its edges. Moreover, we assume w.l.o.g. that all hypergraphs under consideration are both *connected,* i.e., their primal graph consists of a single connected component, and *reduced,* i.e., no hyperedge is contained in any other hyperedge. All our definitions and results easily extend to general hypergraphs.

Let *US* be the set of all (reduced and connected) hypergraphs. A *decomposition method* (short: DM) *D* associates to any hypergraph $\mathcal{H} \in \mathcal{HS}$ a parameter *D-width(H),* called the *D-width* of H.

The decomposition method *D* ensures that, for fixed K, every CSP instance / whose hypergraph $\mathcal{H}_I$ has *D*-width < K: is polynornially solvable, i.e., it is solvable in $O(|I|^{O(1)})$ time, where $|I|$ denotes the size of /. For any *k* > 0, the *k-tractable class C(D,k)* of *D* is defined by *C{D,k)* = {H|D-width.$(\mathcal{H}) \leq k$}. Thus, *C{D,k)* collects the set of CSP instances which, for fixed *k,* are polynornially solvable by using the strategy *D.* Typically, the polynomial above depends on the parameter *k.* In particular, for each *D* there exists a function / such that, for each A:, each instance $I \in C(D,k)$ can be transformed in time $O(|I|^{O(f(k))})$ into an equivalent *acyclic* CSP instance (from where it follows that all problems in *C(D,k)* are polynornially solvable).

Every DM *D* is complete w.r.t. *W,* i.e., *US* = $\bigcup_{k \geq 1} C(D,k)$l. Note that, by our definitions, it holds that JD-widthCW) = min$\{k \mid \mathcal{H} \in C(D,k)\}$

All major tractable classes based on restricted structure fit into this framework. In particular, we shall compare the following decomposition methods:

• Biconnected Components (short: BICOMP) [Freuder, 1985]. Any graph *G* = *(V,E)* can be decomposed into a pair (T, X), where *T* is a tree, and the labeling function X associates to each vertex of *T* a biconnected component of *G* (a component which remains connected after any one-vertex removal). The *biconnected width* of a hypergraph *H,* denoted by BIC0MP-width(H), is the maximum number of vertices over the biconnected components of the primal graph of H..

• Cycle Cutset (short: CUTSET) [Dechter, 1992]. A *cycle cutset* of a hypergraph H is a set $S \subseteq$ *var(H)* such that the subhypergraph of *H* induced by *var(H)* -~ *S* is acyclic. The CUTSET width of *H* is the minimum cardinality over all its possible cycle cutsets.

• Tree Clustering (short: TCLUSTER) [Dechter and Pearl, 1989]. The *tree clustering* method is based on a triangulation algorithm which transforms the primal graph *G* = (V^r, *E)* of any CSP instance I into a chordal graph G'. The maximal cliques of G' are then used to build the constraint scopes of an acyclic CSP /' equivalent to /. The *tree-clustering width* (short: TCLUSTER *width)* of $\mathcal{H}_I$ is 1 if $\mathcal{H}_I$ is an acyclic hypergraph; otherwise it is equal to the maximum cardinality over the cliques of the chordal graph G'

• Treewidth (TREEWIDTH) [Robertson and Seymour, 1986]. We omit a formal definition of graph treewidth here. The TREEWIDTH of a hypergraph *H* is the treewidth of its primal graph plus one. As pointed out below, TREEWIDTH and TCLUSTER are two equivalent methods.

• Hinge Decompositions (short: HINGE) [Gyssens *et* a/., 1994; Gyssens and Paredaens, 1984], Let *H* be a hypergraph, and let *V* $\subseteq$ *var('H)* be a set of variables and *X,Y* $\in$ *var(H).* *X* is [V^r]-adjacent to *Y* if there exists an edge *h* $\in$ *edges(H)* such that *{X,Y}* $\subseteq$ *{h - V).* A [V]-path π from *X* to *Y* is a sequence *X* = $X_0, \ldots, X_\ell$ = *Y* of variables such that: X_i is [V]-adjacent to X_{i+1}, for each *i* $\in$ [0...l-1]. A set *W* $\subseteq$ *var(H)* of variables is [V']-connected if $\forall X, Y \in W$ there is a [V]-path from *X* to *Y.* A *[V]-component* is a maximal *[V]*-connected non-empty set of variables *W* $\subseteq$ *(var(H)* — *V).* For any

[V1-component *C,* let *edges(C)* = $\{h \in edges(\mathcal{H}) \mid h \cap C \neq \emptyset\}$

Let *H* ∈HS and let *H* be either *edges(H)* or a proper subset of *edges(H)* containing at least two edges. Let *C*\, .., C_m be the connected $[var(H)]$-components of *H.* Then, *H* is a *hinge* if, for i = I,...,m, there exists an edge $h_i \in H$ such that *var(edges* \(C_i)) ∩ $var(H)) \subseteq h_i$. A hinge is *minimal* if it does not contain any other hinge. (Our definition of hinge is equivalent to the original one in [Gyssens *et* a/., 1994; Gyssens and Paredaens, 1984].)

A *hinge-decomposition* of *H* is a tree *T* such that all the following conditions hold: (1) the vertices of *T* are minimal hinges of *H;* (2) each edge in *edges(H)* is contained in at least one vertex of T; (3) two adjacent vertices *A* and *D* of *T* share precisely one edge *L* ∈ *edges(H)*\ moreover, *L* consists exactly of the variables shared by *A* and *B* (i.e., $L = var(A) \cap var(B)$); (4) the variables of *H* shared by two vertices of *T* are entirely contained within each vertex on their connecting path in T.

The size (i.e., the cardinality) of the largest vertex of *T* is called the *degree of cyclicity* of *H.* This is precisely what we call here the HINGE *width* of *H.* It was shown in [Gyssens and Paredaens, 1984] that for any CSP instance /, the HINGE width of H_1 is the cardinality of the largest minimal hinge of $\mathcal{H}_I$.

• Hinge Decomposition + Tree Clustering (short: HINGE^{TCLUSTER}) [Gyssens *et* a/., 1994]. It has been shown [Gyssens *et* a/., 1994] that the minimal hinges of a hypergraph can be further decomposed by means of the triangulation technique of the above-described tree-clustering method. This leads to the HINGE^{TCLUSTER} *method.* Let T = *(N, E)* be a hinge tree of a hypergraph *H.* For any hinge *H* ∈ *N*\ let *w(H)* be the minimum between the cardinality of *H* and the TCLUSTER width of the hypergraph *(var(H), H).* T HINGE^{TCLUSTER} t h of *T* is $max_{H \in N}\{w(H)\}$. Define t HINGE^{TCLUSTER}*d t h* of *H* as the minimum HINGE^{TCLUSTER} width over all its hinge decompositions.

For each of the above decomposition methods *D* it was shown that for any fixed k, given a CSP instance /, deciding whether a hypergraph $\mathcal{H}_I$ has D-width$(\mathcal{H}_I) \leq k$ is feasible in polynomial time and that solving CSPs whose associated hypergraph is of width ≤ A: can be done in polynomial time. In particular, *D* consists of two phases. Given a CSP instance /, the (A:-bounded) D-width *w* of $\mathcal{H}_I$ along with a corresponding decomposition is first computed. Exploiting this decomposition, *I* is then solved in time $O(|I|^{w+1} \log |I|)$ (for most methods this phase consists of the solution of an acyclic CSP instance equivalent to *I).*

The cost of the first phase is independent on the constraint relations of /; in fact, it is $O(|\mathcal{H}_I|^{c_1 k + c_2})$, where $|\mathcal{H}_I|$ is the size of the hypergraph $\mathcal{H}_I$, and c_1, c_2 are two constants relative to the method D ($0 \leq c_1, c_2 \leq 3$ for the methods above). Observe also that computing the D-width *w* of a hypergraph in general (i.e., without the bound *w* ≤k:) is NP-hard for most methods; while it is polynomial for HINGE, and it is even linear for BICOMP.

Further interesting methods that do not explicitly generalize acyclic hypergraphs are based on a notion of *width* as used in [Freuder, 1982; 1985]. If C is a total ordering of the vertices of a graph *G* = *(V, E),* then the C-width of *G* is defined by $w_c(G)$ = max_{vev} |{ *{v,w}* € *E* s.t. *w* C *v}*\. The width of *G* is the minimum of all C-widths over all possible total orderings C of *V.* For each fixed constant A:, it can be determined in polynomial time whether a graph is of width *k.* [Freuder, 1982] observed that many naturally arising CSPs are of very low width. Note that bounded width in this sense is a structural property. The following theorem shows that bounded width alone does not entail tractability.

Theorem 3.1 *Constraint solvability remains NP-complete even if restricted to CSPs whose primal graph has width bounded by 4.*

Proof. 3COL remains NP-complete even for graphs of degree 4 (cf. [Garey and Johnson, 1979]). Such graphs, however, have width < 4. The theorem follows by the well-known natural encoding of 3COL as a CSP. |

Freuder showed that a CSP of width A: whose relations enjoy the property of *k'-consistency,* where K;' > A:, can be solved in a backtrack-free manner, and thus in polynomial time [Freuder, 1982; 1985].

[Dechter and Pearl, 1988] consequently introduce the notion of *induced width w** which is - roughly the smallest width *k* of any graph *G'* obtained by triangulation methods from the primal graph *G* of a CSP such that *G'* ensures *k* 4- 1-consistency. Graphs having induced width < A: can be also characterized as *partial k-trees* [Freuder, 1990] or, equivalently, as graphs having tree width < *k* [Arnborg *et* a/., 1991]. It follows that, for fixed AT, checking whether *w** < *k* is feasible in linear time [Bodlaender, 1997]. If *w** is bounded by a constant, a CSP is solvable in polynomial time. The approach to CSPs based on *w** is referred to as the *w**-Tractability method [Dechter, 1992]. Note that this method is implicitly based on hypergraph acyclicity, given that the used triangulation methods enforce chordality of the resulting graph *G'* and thus acyclicity of the corresponding hypergraph. It was noted [Dechter and Pearl, 1989; Dechter, 1992] that, for any given CSP instance /, TCLUSTER-width(W/) = *w*(Hj)* + 1.

4 Hypertree Decompositions of CSPs

A new class of tractable conjunctive queries, which generalizes the class of acyclic queries, has been recently identified [Gottlob *et* a/., 1999]. Deciding whether a given query belongs to this class is polynomial time feasible and even highly parallelizable. In this section, we first generalize this notion to the wider framework of hypergraphs, and then show how to employ this notion in order to define a new decomposition method we will refer to as HYPERTREE.

A *hypertree for a hypergraph H* is a triple (T, x,A), where *T* = (N, *E)* is a rooted tree, and X and A are labeling functions which associate to each vertex *p E N*

two sets $\chi(p) \subseteq var(\mathcal{H})$ and $\lambda(p) \subseteq edges(\mathcal{H})$. If $T' = (N', E')$ is a subtree of T, we define $\chi(T') = \bigcup_{v \in N'} \chi(v)$. We denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.

Definition 4.1 A *hypertree decomposition* of a hypergraph $\mathcal{H}$ is a hypertree $\langle T, \chi, \lambda \rangle$ for $\mathcal{H}$ which satisfies all the following conditions:

1. for each edge $h \in edges(\mathcal{H})$, there exists $p \in vertices(T)$ such that $var(h) \subseteq \chi(p)$;
2. for each variable $Y \in var(\mathcal{H})$, the set $\{p \in vertices(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T;
3. for each $p \in vertices(T)$, $\chi(p) \subseteq var(\lambda(p))$;
4. for each $p \in vertices(T)$, $var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

A hypertree decomposition $\langle T, \chi, \lambda \rangle$ of $\mathcal{H}$ is a *complete decomposition* of $\mathcal{H}$ if, for each edge $h \in edges(\mathcal{H})$, there exists $p \in vertices(T)$ such that $var(h) \subseteq \chi(p)$ and $h \in \lambda(p)$.

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

Intuitively, if $\mathcal{H}$ is a cyclic hypergraph, the χ labeling selects the set of variables to be fixed in order to split the cycles and achieve acyclicity; $\lambda(p)$ "covers" the variables of $\chi(p)$ by a set of edges.

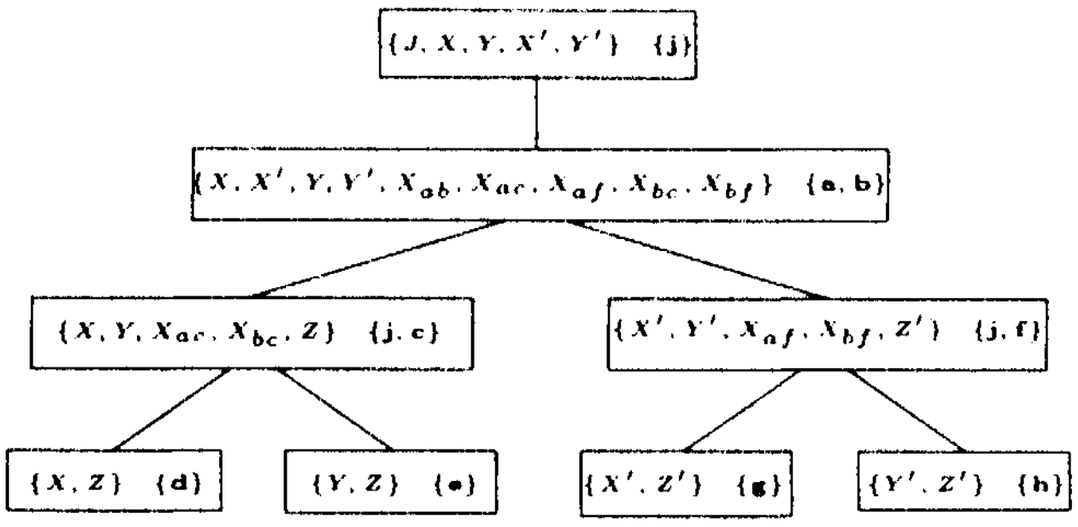

Figure 1: A 2-width hypertree decomposition of *H1*

Example 4.2 Consider the following constraint, scopes:

$$a(X_{ab}, X, X', X_{ac}, X_{af}); b(X_{ab}, Y, Y', X_{bc}, X_{bf});$$
$$c(X_{ac}, X_{bc}, Z); d(X, Z); e(Y, Z); f(X_{af}, X_{bf}, Z');$$
$$g(X', Z'); h(Y', Z'); j(J, X, Y, X', Y')$$

Let H1 be their corresponding hypergraph. $\mathcal{H}_1$ is clearly cyclic, and thus $hw(\mathcal{H}_1) > 1$ (as only acyclic hypergraphs have hypertree width 1). Figure 1 shows a (complete) hypertree decomposition of $\mathcal{H}_1$ having width 2, hence $hw(\mathcal{H}_1) = 2$.

It is easy to see that the acyclic CSPs are precisely the CSPs of hypertree width one.

We say that a CSP instance / has A;-bounded hypertree-width if $hw(\mathcal{H}_I) \leq k$, where $\mathcal{H}_I$ is the hypergraph associated to /. From the results in [Gottlob *et* a/., 1999], it follows that A;-bounded hypertree-width is efficiently decidable, and that a hypertree decomposition of width *k* can be efficiently computed (if any).

We next show that any CSP instance / is efficiently solvable, given a k-bounded complete hypertree-decomposition *HD* of H_1. To this end, we define an acyclic CSP instance which is equivalent to / and whose size is polynomially bounded by the size of /.

For each vertex *p* of the decomposition *HD,* we define a new constraint scope whose associated constraint relation is the projection on X*(p)* of the join of the relations in *X(p).* This way, we obtain a join-tree $\mathcal{JT}$ of an acyclic hypergraph $\mathcal{H}'$. $\mathcal{H}'$ corresponds to a new CSP instance /' over a set of constraint relations of size i$O(n^k)$, where *n* is the input size (i.e., *n* = \1\) and *k* is the width of the hypertree decomposition *HD.* By construction, /' is an acyclic CSP, and we can easily show that it is equivalent to the input CSP instance *L* Thus, all the efficient techniques available for acyclic CSP instances can be employed for the evaluation of /', and hence of 7.

Theorem 4.3 *Given a CSP I and a k-width hypertree decomposition of $\mathcal{H}_I$, I is solvable in $O(n^k \log n^k)$ = 0(n^k* logn) *time, where n is the size of I.*

5 Comparison Criteria

For comparing decomposition methods we introduce the relations $\preceq$, $\rhd$, and $\prec\!\!\prec$ defined as follows:

$\underline{D_1 \preceq D_2}$, in words, D_2 *generalizes* D_1, if $\exists \delta \geq 0$ such that, $\forall k > 0$, $C(D_1, k) \subseteq C(D_2, k+\delta)$. Thus $D_1 \preceq D_2$ if every class of CSP instances which is tractable according to D_1 is also tractable according to D_2.

$\underline{D_1 \rhd D_2}$ (D_1 beats D_2) if there exists an integer k such that $\forall m\ C(D_1, k) \not\subseteq C(D_2, m)$. To prove that $D_1 \rhd D_2$, it is sufficient to exhibit a class of hypergraphs contained in some $C(D_1, k)$ but in r$C(D_2, j)$ for $j' \geq 0$. Intuitively, $D_1 \rhd D_2$ means that at least on some class of CSP instances, D_1 outperforms D_2.

$\underline{D_1 \prec\!\!\prec D_2}$ if $D_1 \preceq D_2$ a $D_2 \rhd D_1$. In this case we say that D_2 *strongly generalizes* D_1

Mathematically, $\preceq$ is a *preorder,* i.e., it is reflexive, transitive but not antisymmetric. We say that D_1 *is* $\preceq$-*equivalent to* D_2, denoted $D_1 \equiv D_2$, if both $D_1 \preceq D_2$ and $D_2 \preceq D_1$ hold.

The decomposition methods *D* and *D2* are *strongly incomparable* if both $D_1 \rhd D_2$ and $D_2 \rhd D_1$. Note that if D_1 and D_2 are strongly incomparable, then they are also incomparable w.r.t. the relations $\preceq$ and $\prec\!\!\prec$.

6 Comparison Results

Figure 2 shows a representation of the hierarchy of DMs determined by the $\prec\!\!\prec$ relation. Each element of the hierarchy represents a DM, apart from that containing *Tree Clustering, w**, and *Treewidth* which are grouped together because they are $\preceq$-equivalent as easily follows from the observations in Section 3.

Theorem 6.1 *For each pair D_1 and D_2 of decompositions methods represented in Figure 2, the following holds:*

- *There is a directed path from D_1 to D_2 iff $D_1 \prec\!\!\prec D_2$, i.e., iff D_2 strongly generalizes D_1.*
- *D_1 and D_2 are not linked by any directed path iff they are strongly incomparable.*

Hence, Fig. 2 gives a complete picture of the relationships holding among the different methods.

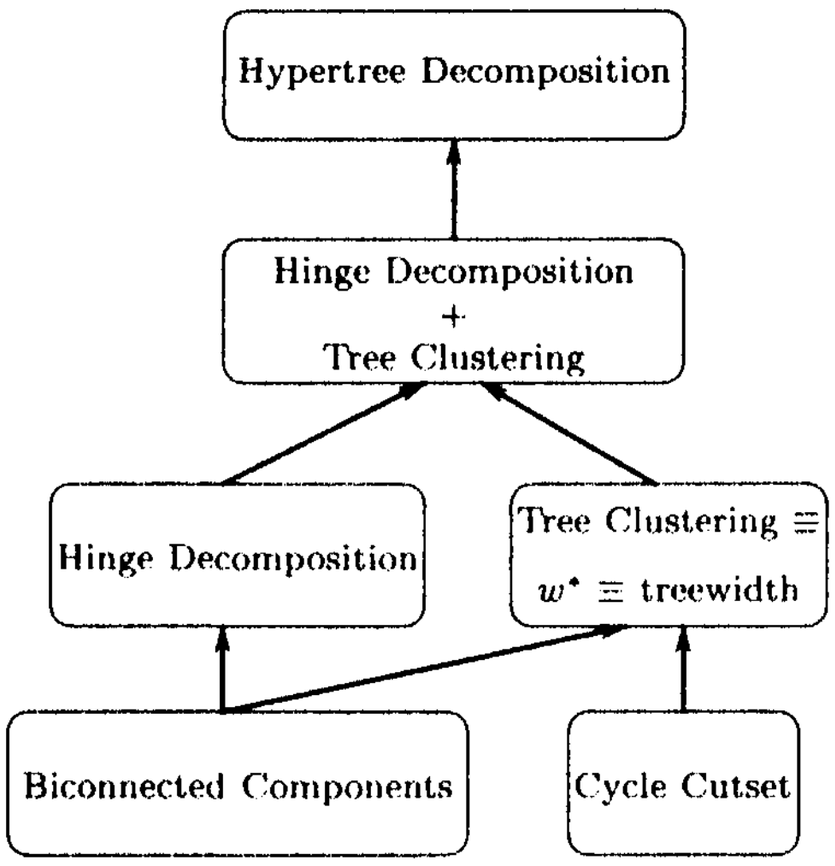

Figure 2: Constraint Tractability Hierarchy

Below we sketch a proof of Theorem 6.1. For space reasons, we report only succinct versions of selected proofs. Detailed proofs of all results are available in the full version of this paper [Gottlob *et* a/., 1999b].

For any $n > 1$ and $m > 0$, let *Circle(n,m)* be the hypergraph having n edges $\{h_1, \ldots, h_n\}$ defined as follows:

- $h_i = \{X_i^1, \ldots, X_i^m, X_{i+1}^1, \ldots, X_{i+1}^m\}$ $\forall 1 \leq i \leq n-1$;
- $h_n = \{X_n^1, \ldots, X_n^m, X_1^1, \ldots, X_1^m\}$.

For m = 1, *Circlc{n,* 1) is a graph consisting of a simple cycle with n edges (like a circle). Note that, for any $n > 1$ and $m > 0$, *Circle(n,m)* has hypertree width 2; a 2-width hypertree decomposition of *Circle(n,* ra) is shown in Figure 3. Thus, $(\bigcup_{n>1,m>0}\{Circle(n,m)\}) \subseteq$ C(HYPERTREE, 2)

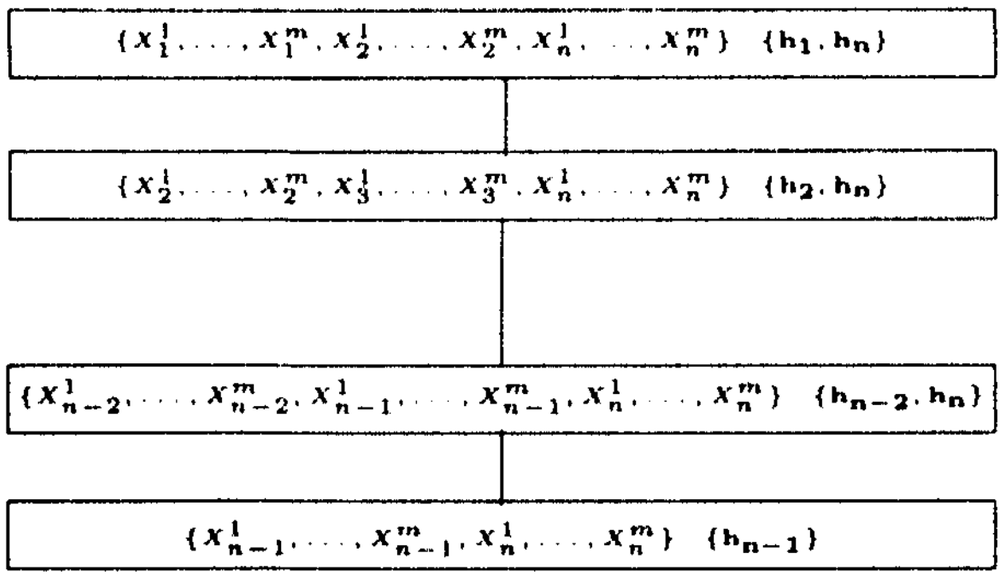

Figure 3: 2-width hypertree decomposition of $Circle(n,m)$

Lemma 6.2 HINGE$\prec\!\!\prec$ HYPERTREE.

Proof. (HINGE$\preceq$ HYPERTREE.) Given a hinge decomposition $T = (V, E)$ of $\mathcal{H}$, we build a hypertree decomposition HD of $\mathcal{H}$ having the same width as T. HD has the same tree shape as T; the labels of a vertex p of (the tree of) HD are $\lambda(p) = H$ and $\chi(p) = var(H)$, where H is the hinge corresponding to p in the hinge decomposition.

(HYPERTREE $\rhd$ HINGE.) For any $k > 0$, C(HYPERTREE, 2) $\supseteq$ $(\bigcup_{n>1}\{Circle(n,1)\})$ $\not\subseteq$ C(HINGE, k). Indeed, for any $n > 1$, the degree of cyclicity of $Circle(n,1)$ is n [Gyssens *et al.*, 1994]. ∎

Lemma 6.3 TCLUSTER$\prec\!\!\prec$ HYPERTREE.

Proof. (TCLUSTER$\preceq$ HYPERTREE.) Let H = *(V, H)* be a hypergraph, and $\langle T, \psi \rangle$ be the result of the application of the tree-clustering method on H, where *T = (N,E)* is a tree, and $\psi : N \rightarrow 2^V$ is a labeling function which assigns to each vertex of *T* a set of variables of H. For any $p \in N$, the set of variables $\psi(p)$ corresponds to a maximal clique identified by the tree-clustering method.

From $\langle T, \psi \rangle$, we define a complete hypertree decomposition *HD* = (T,x, A) having the same tree *T* as the output of the tree clustering method. The labelings X and λ are defined according to the following procedure.

1. for each $p \in N$, set $\chi(p) = \psi(p)$ and $\lambda(p) = \emptyset$;
2. for each edge $h \in H$, choose a vertex $p \in N$ s.t. $h \subseteq \chi(p)$, and add h to $\lambda(p)$ (i.e., $\lambda(p) := \lambda(p) \cup \{h\}$);
3. While there is a vertex $p \in N$ s.t. $\chi(p)$ contains a variable X not covered by $\lambda(p)$ (i.e., $X \in (\chi(p) - var(\lambda(p)))$, proceed as follows. (A) Find a path π in T linking p to a vertex $q \in N$ s.t. (i) $X \in var(\lambda(q))$ and, (ii) $X \notin var(\lambda(s))$ for every vertex s in $\pi - \{q\}$. (B) Choose an edge $h \in \lambda(q)$ s.t. $X \in h$. (C) Add h to both $\lambda(s)$ and $\chi(s)$, for every vertex $s \in (\pi - \{q\})$ (i.e., $\chi(s) := \chi(s) \cup h$, and $\lambda(s) := \lambda(s) \cup \{h\}$).

We have that *HD* is a complete hypertree decomposition of *H,* and its HYPERTREE width is smaller than or equal to the TCLUSTER-width of H.

(HYPERTREE $\rhd$ TCLUSTER.) Let $S = \{Circle(3,m) \mid m > 1\}$. For any ra, > 1, the primal graph *G* of *Circle(3,m)* is a clique of 3ra variables. Thus, *G* does not need any triangulation, because it is a chordal graph. The TCLUSTER-width of *Circle(3,m)* is clearly 3ra; while its hypertree width is 2. Hence, for any $k > 0$, $S \not\subseteq C$(TCLUSTER,k), whereas $S \subseteq C$(HYPERTREE, 2) |

Lemma 6.4 HINGE *and* TCLUSTER *are strongly incomparable.*

Proof. (HINGE $\rhd$ TCLUSTER). From the proof above, $S = \{Circle(3,m) \mid m > 1\} \not\subseteq C$(TCLUSTER, k), for any $k > 0$. However, $S \subseteq C$(HINGE, 3), because every hypergraph in S has only three (hyper)edges.

(TCLUSTER $\rhd$ HINGE). From the proof of Lemma 6.2, $(\bigcup_{n>1}\{Circle(n,1)\}) \not\subseteq C$(HINGE, k), for any $k > 0$. However, $(\bigcup_{n>1}\{Circle(n,1)\}) \subseteq C$(TCLUSTER, 3), because all these graphs can be triangulated in a way that their maximal cliques have cardinality 3 at most. |

Interestingly, even the combination of TCLUSTER with HINGE is strongly generalized by the hypertree-decomposition method.

Lemma 6.5 $\mathbf{HINGE}^{\mathbf{TCLUSTER}} \prec\prec \mathbf{HYPERTREE}$

Proof. The proof of $\mathbf{HINGE}^{\mathbf{TCLUSTER}} \preceq$ HYPERTREE is very similar to that of Lemma 6.3, except that the λ labelings must be initialized in a suitable way (instead of $\emptyset$). To see that $\mathbf{HYPERTREE} \rhd \mathbf{HINGE}^{\mathbf{TCLUSTER}}$, note that $(\bigcup_{n>1,m>0} \{Circle(n,m)\}) \not\subseteq C(\mathbf{HINGE}^{\mathbf{TCLUSTER}}, k)$ holds for any $k > 0$; while, $(\bigcup_{n>1,m>0} \{Circle(n,m)\}) \subseteq C(\mathbf{HYPERTREE}, 2)$. ∎

7 Binary CSPs

On binary constraint networks, where the constraints relations have arity two, the differences among the decomposition strategies highlighted in Section 6 become less evident. Indeed, bounding the arities of the constraint relations, the k-tractable classes of some decomposition strategies collapse. In particular, as shown in the full version of this paper [Gottlob *et ai,* 1999b], on binary constraints networks, $\mathbf{TCLUSTER} \equiv \mathbf{HINGE}^{\mathbf{TCLUSTER}}$ and $\mathbf{HINGE}^{\mathbf{TCLUSTER}} \equiv \mathbf{HYPERTREE}$ hold. The relationships among the other decomposition methods remain the same as for the general case (Fig. 2).

To evidentiate the differences of the above decomposition strategies on the domain of binary CSPs, we can compare their respective widths. For each discussed decomposition method *D,* any CSP instance / is solvable in time $O(|I|^{w+1} \log |I|)$, once the *D*-width *w* of $\mathcal{H}_I$ along with a corresponding decomposition have been computed, as noted in Section 3. Thus, the *D*-width is a measure of the efficiency of a decomposition method: the smaller the D-width of $\mathcal{H}_I$, the more efficient the application of strategy *D* to /.

Theorem 7.1

1. *For every binary CSP instance* I, $\mathbf{TCLUSTER}\text{-}width(\mathcal{H}_I) = \mathbf{HINGE}^{\mathbf{TCLUSTER}}\text{-}width(\mathcal{H}_I)$.
2. *For every binary CSP instance* I, $\mathbf{HYPERTREE}\text{-}width(\mathcal{H}_I) \leq \mathbf{TCLUSTER}\text{-}width(\mathcal{H}_I)$.
3. *For some CSP instance* I, $\mathbf{TCLUSTER}\text{-}width(\mathcal{H}_I) = 2 \cdot \mathbf{HYPERTREE}\text{-}width(\mathcal{H}_I)$.

Acknowledgments

Research supported by *FWF (Austrian Science Funds)* under the project Z29-INF and by the *CNR (Italian National Research Council),* under grant n.203.15.07.

References

[Arnborg *et ai,* 1991] S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs. *J. of Algorithms,* 12:308 340, 1991.

[Beeri *et ai,* 1983] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM,* 30(3):479 513, July, 1983.

[Bernstein and Goodman, 1981] RA. Bernstein, and N. Goodman. The power of natural semijoins. *SI AM Journal on Computing,* 10(4):751 771, 1981.

[Bibel, 1988] W. Bibel. Constraint Satisfaction from a Deductive Viewpoint. *AIJ,* 35, 401-413, 1988.

[Bodlaender, 1997] H.L. Bodlaender. Treewidth: Algorithmic Techniques and Results. In *Proc. of MFCS'97,* Bratislava. LNCS 1295, Springer, pp. 19-36, 1997.

[Dechter, 1992] R. Dechter. Constraint Networks. In *Encyclopedia of AI,* 2nd ed., Wiley and Sons, pp. 276-285,1992.

[Dechter and Pearl, 1988] R. Dechter and J. Pearl. Network based heuristics for CSPs. *AIJ,* 34(1):1 38, 1988.

[Dechter and Pearl, 1989] R. Dechter and J. Pearl. Tree clustering for constraint networks. *AIJ,* 38:353-366, 1989.

[Freuder, 1982] E.G. Freuder. A sufficient condition for backtrack-free search. *J ACM,* 29(1):24~32, 1982.

[Freuder, 1985] E.G. Freuder. A sufficient condition for backtrack-bounded search. *J ACM,* 32(4):755 761, 1985.

[Freuder, 1990] E.C. Freuder. Complexity of K-Tree Structured CSPs. *Proc. of AAAI'90,* 1990.

[Garey and Johnson, 1979] MR. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of* NP-*completeness.* Freeman and Comp., NY, USA, 1979.

[Gottlob *et ai,* 1998] G. Gottlob, N. Leone, and F. Scarcello. The Complexity of Acyclic Conjunctive Queries, in *Proc. of FOCS'98,* pp.706 715, Palo Alto, CA, 1998.

[Gottlob *et al,* 1999] G. Gottlob, N. Leone, and F. Scarcello. "Hypertree Decompositions and Tractable Queries," in *Proc of PODS'99,* Philadelphia, May, 1999.

[Gottlob *et* a/., 1999b] G. Gottlob, N. Leone, and F. Scarcello. "A Comparison of Structural CSP Decomposition Methods," Tech. Rep. DBAI-TR-99/25, currently available on the web as: www.dbai.tuwien.ac.at/staff/gottlob/CSP-decomp.ps

[Gyssens *et ai,* 1994] M. Gyssens, P. Jeavons, and D. Cohen. Decomposing constraint satisfaction problems using database techniques. *AIJ,* 66:57-89, 1994.

[Gyssens and Paredaens, 1984] M. Gyssens and J. Paredaens. A Decomposition Methodology for Cyclic Databases. In *Advances in Database Theory,* volume 2, pp. 85-122. Plenum Press New York, NY, 1984.

[Jeavons *et al,* 1997] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *J ACM,* 44(4), 1997.

[Maier, 1986] D. Maier, *The theory of relational databases.* Computer Science Press, Rockville, MD, 1986.

[Pearson and Jeavons, 1997] J. Pearson and P.G. Jeavons. A Survey of Tractable Constraint Satisfaction Problems, CSD-TR-97-15, Royal Holloway Univ. of London, 1997.

[Robertson and Seymour, 1986] N. Robertson and P.D. Seymour. Graph Minors II. Algorithmic aspects of tree width. *Journal of Algorithms,* 7:309-322, 1986.

[Seidel, 1981] R. Seidel. A new method for solving constraint satisfaction problems. In *Proc. of IJCAI'81,* 1981.