Non-Binary Constraints and Optimal Dual-Graph Representations

Gianluigi Greco and Francesco Scarcello
DEIS - University of Calabria
1-87036, Rende, Italy
{ggreco,scarcello}@si.deis.unical.it

Abstract

We study the relationships among structural methods for identifying and solving tractable classes of Constraint Satisfaction Problems (CSPs). In particular, we first answer a long-standing question about the notion of biconnected components applied to an "optimal" reduct of the dual constraint-graph, by showing that this notion is in fact equivalent to the hinge decomposition method. Then, we give a precise characterization of the relationship between the treewidth notion applied to the hidden-variable encoding of a CSP and the same notion applied to some optimal reduct of the dual constraint-graph. Finally, we face the open problem of computing such an optimal reduct. We provide an algorithm that outputs an approximation of an optimal tree decomposition, and give a qualitative explanation of the difference between this graph-based method and more general hypergraph-based methods.

1 Introduction and summary of results

Constraint satisfaction is a central issue of AI research and has an impressive spectrum of applications (see, e.g., [Pearson and Jeavons, 1997]). A constraint (S_i, R_i) consists of a *constraint scope* S_i, i.e., a list of variables and of an associated *constraint relation* r_i containing the legal combinations of values. A CSP consists of a set $\{(S_1, r_1), (S_2, r_2), \ldots, (S_q, r_q)\}$ of constraints whose variables may overlap. A solution to a CSP is an assignment of values to all variables such that all constraints are simultaneously satisfied. By *solving* a CSP we mean determining whether the problem has a solution at all (i.e., checking for *constraint satisfiability),* and, if so, compute one solution.

Constraint satisfiability in its general form is well-known to be NP-hard. Much effort has been spent to identify *tractable classes* of CSPs, and deep and useful results have been achieved. The various successful approaches to obtain tractable CSP classes can be divided into two main groups [Pearson and Jeavons, 1997]: the techniques that identify tractable classes of CSPs only on the base of the structure of the constraint scopes $\{S_1, \ldots S_q\}$ independently of the actual constraint relations $r_1, \ldots, r_q$ and the techniques that identify tractable classes by exploiting particular properties of the constraint relations $r_1, \ldots, r_q$. In this paper, we will deal with this latter group of techniques, usually called *structural decomposition methods.* There are several papers proposing polynomially tractable classes of constraints based on different structural properties of constraint-scopes interactions (see, e.g., [Dechter, 1992; Dechter and Pearl, 1989; Gyssens *et al,* 1994; Gottlob *et al,* 2000J). Such a structure is best represented by the hypergraph $\mathcal{H}(I) = (V, H)$ associated to any CSP instance $I = (Var, U, C)$, where $V = Var$ and $H = \{var(S) \mid C = (S, r) \in C\}$, and $var(S)$ denotes the set of variables in the scope *S* of the constraint *C.* We often denote the set of vertices by N*{H)* and the set of hyperedges by $\mathcal{E}(\mathcal{H})$.

However, many interesting techniques for solving CSPs or for identifying tractable classes of CSPs have been designed for binary CSPs, i.e., CSP instances where each scope contains two variables at most. Therefore, historically, the first attempts to deal with general (i.e., non-binary) constraint problems try to exploit the existent methods, by representing any CSP instance / by some graph, rather than by the hypergraph *H{I).* A first idea is to use the primal graph of H (*I),* whose edges connect each pair of variables occurring together in some constraint of /. Clearly, there is an evident loss of information in using the primal graph instead of the hypergraph. For instance, each constraint-scope of / induces a clique in the primal graph, but if one looks at the graph only, there is no way to understand whether such a clique comes from a hyperedge of the hypergraph, or by some intricate interactions among scopes. In fact, in [Gottlob *et al,* 2000], a deep comparison among various structural decomposition methods showed that some technique designed for hypergraphs is more powerful than all the (known) techniques working on the primal graphs. In this paper, we focus on the other two important graph-based representations of non-binary constraints, described in the literature, only marginally considered in that work:

Dual-graph Representation [Dechter, 1992]. Given a hypergraph H, its *dual graph,* denoted by *dual('H)* = *{N,E),* is the graph whose set of vertices iV is the set of hyperedges $\mathcal{E}(\mathcal{H})$, and whose edges connect each pair of vertices (i.e., hyperedges) having some variable in common, that is $E = \{\{h_1, h_2\} \mid h_1, h_2 \in \mathcal{E}(\mathcal{H}) \text{ and } h_1 \cap h_2 \neq \emptyset\}$

Hidden-variable Representation fSeidel, 1981; Chekuri and Rajaraman, 2000]. Given a hypergraph *H,* we define its *incidence graph* as the bipartite graph *inc(H)* = (AT, *E),*

where $N = \mathcal{E}(\mathcal{H}) \cup \mathcal{N}(\mathcal{H})$, and $E = \{ \{h, a\} \mid h \in \mathcal{E}(\mathcal{H}) \text{ and } a \in h)\}$, i.e. it contains an edge from h to a if and only if a is a node of *h.* (E.g., Figure 2 shows a constraint hypergraph (a) and its incidence graph (b)).

There is a great interest in comparing CSP solving techniques based on these encodings of the structure [Bacchus *et al,* 2002] and long standing questions about their relationships with hypergraph-based techniques, described below.

One of the major difficulties in doing precise and formal analysis of dual-graph based methods is due to an important feature of this encoding: some edges of the dual graph can be safely removed from this graph, making the evaluation of CSPs easier. Indeed, even if *dual (H)* appears very intricate, sometimes is possible to find suitable simplifications that make it much more useful. Such simplified graphs are called *reducts* of *dual (H).* For instance, if H. is acyclic, there is a polynomial time algorithm for making its dual graph acyclic, and in fact a join tree of H. An example is shown in Figure 2.c, where the acyclic graph obtained by removing the dashed edges is a join tree of the given hypergraph. However, different removal choices may lead to different performances of evaluation algorithms. Thus, the efficiency of any technique based on the dual graph depends crucially on the availability of a good algorithm for simplifying the dual graph. Note that finding the "best reduct" is a difficult task and is currently not known whether it is feasible in polynomial time, in general.

On the other hand, the fact that effective dual graph representations are not unique made comparisons among different methods quite difficult. For instance, Gyssens et al. [1994] compared the notion of Hinge decompositions (short: HINGE) and the notion of Biconnected components of the dual g r (short: $\mathbf{BICOMP}^d$). It turned out that HINGE is a generalization of $\mathbf{BICOMP}^d$, and thus the hinge decomposition technique is not worse than the biconnected components technique. However, the precise relationship between these methods remained an open question, because biconnected components can perform very bad unless "clever simplifications" of the dual graph are chosen.

The first contribution of this paper is solving this question. First we formally define, for each method D, the method D^{optd}, that is, the method *D* applied to the best possible simplification of the dual graph with respect to *D.* Indeed, in general, the notion of best simplification depends on the method *D* used for decomposing the graph. This way, methods applied to dual graph encodings are well-defined and can be compared with other methods.

We formally prove that BIC0MP^{optd} is equivalent to HINGE. In fact, we show that any hinge decomposition corresponds to the biconnected-components tree of some reduct of the dual graph. It is worthwhile noting that, as a corollary of this result, we obtain that, for the $\mathbf{BICOMP}^{optd}$ method, an optimal reduction of the dual graph can be computed in polynomial time, since any hinge decomposition can be computed in polynomial time.

Then, we consider the powerful decomposition method for dealing with graphs: the tree decomposition method [Robertson and Seymour, 1986], which is equivalent to the tree-clustering method [Dechter and Pearl, 1989]. It is known that any class of CSP instances such that the treewidth of their incidence graph (respectively, of some reduct of their dual graphs) is bounded by some constant *k* is tractable. That is, all such instances may be evaluated in time $O(n^c)$, where *n* is the size of a CSP instance and *c* is a constant that depends crucially on the bound A: on the decomposition width.

We perform a detailed comparison of the tree decomposition method applied to the incidence graph of the hypergraph (i.e., on the hidden-variable encoding), denoted by TREEWIDTH^{in}, and the tree decomposition method applied to some optimal reduct of the dual graph, denoted by TREEWIDTH^{0ptd}. It turns out that every CSP class that is tractable according to TREEWIDTH*" is tractable according to TREEWIDTH^{optd}, as well. Moreover, there are CSP classes that are tractable according to $\mathbf{TREEWIDTH}^{optd}$ but are not tractable according to TREEWIDTH, i.e., their largest width is not bounded by any fixed number. However, we show that TREEWIDTH^{optd} does not strongly generalize TREEWIDTH^{*1}. Indeed, there are classes of CSPs whose incidence-graph treewidth is bounded by a constant K, but the largest width of some optimal reduct of the dual graph is much greater. Thus, even if such classes are tractable, their evaluation can be much more efficient by using the $\mathbf{TREEWIDTH}^{*1}$ method. It follows that either of these methods may be useful for some kind of CSP instances, and hence there is no definitely better choice between them.

Finally, we focus on further interesting open questions about $\mathbf{TREEWIDTH}^{optd}$. Define the *optimal treewidth* $tw^{optd}(H)$ of the dual graph of some hypergraph *H* as the minimum treewidth over all reducts of *dual{H).* Kolaitis and Vardi [2000] observed that is not trivial to find a "good" reduct of the dual graph and defined the following problem k-OPT, for any fixed constant $k > 0$: Given a hypergraph H, decide whether the optimal treewidth of *dual(H)* is at most K. The question is whether K-OPT is decidable in polynomial time or not, that is whether there is an efficient way for computing a reduct of the dual graph that have the minimum treewidth. Moreover, even if it is known that TREEWIDTH^{optd} is strongly generalized by the hypertree decomposition method [Gottlob *et al,* 2000], it is not clear why there is such a big difference between these methods. Indeed, at a first glance the kind of tree labelling in these methods seems rather similar.

We face both the above questions. Let $k > 0$ be a fixed constant and H be a constraint hypergraph. We present a polynomial time algorithm k-TREE-APPROX that, if the optimal treewidth of *dual(H)* is at most *k,* outputs a tree decomposition of width at most 2k of some reduct of *dual (H).* Thus, k-TREE-APPROX provides a 2-approximation of $\mathbf{TREEWIDTH}^{optd}$. Note that the question whether K-OPT is decidable in polynomial time remains open, because k-TREE-APPROX can compute a tree decomposition of width at most *2k* even if the optimal treewidth *k'* of *dual (H)* is greater than k, i.e., if $k < k' \leq 2k$.

Moreover, our algorithm is also able to compute a new kind of structural decomposition that allows us to shed some light on the striking difference between TREEWIDTH^{optd} and the more general hypergraph-based notions of query decomposition and hypertree decomposition.

2 Preliminaries

It is well known that CSPs with *acyclic* constraint hypergraphs are polynomially solvable [Dechter, 1992]. The known structural properties that lead to tractable CSP classes are all (explicitly or implicitly) based on some generalization of acyclicity. In particular, each method *D* defines some concept of *width* which can be interpreted as a measure of cyclicity of the underlying constraint (hyper)graph such that, for each fixed width k, all CSPs of width bounded by A: are solvable in polynomial time. This (possibly infinite) set of CSPs is called the *tractability class* of *D* w.r.t. k, and is denoted by *C(D, k).* Any pair of decomposition methods *D*\ and D_2 can be compared according to their ability to identify tractable classes of CSPs. Formally, Gottlob et al. [2000] defined the following criteria:

Generalization. D_2 *generalizes* D_1 ($D_1 \preceq D_2$) if there exists a constant δ such that, for each $k > 0$, $C(D_1, k) \subseteq C(D_2, k + \delta)$. In practical terms, this means that whenever a class C of constraints is tractable according to method D_1, it is also tractable according to D_2.

Beating. D_2 *beats* D_1, denoted by $D_2 \rhd D_1$, if there exists an integer $k > 0$ such that $C(D_2, k)$ is not contained in class $C(D_1, m)$ for any $m > 0$. Intuitively, this means that some classes of problems are tractable according to D_2 but not according to D_1. For such classes, using D_2 is thus better than using D_1.

Strong generalization. D_2 *strongly generalizes* D_1, denoted by $D_1 \prec\prec D_2$, if D_2 generalizes D_1 and D_2 beats D_1. This means that D_2 is really the more powerful method, given that, whenever D_1 guarantees polynomial runtime for constraint solving, then also D_2 guarantees tractable constraint solving, but there are classes of constraints that can be solved in polynomial time by using D_2 but are not tractable according to D_1.

Equivalence. D_1 and D_2 are *equivalent*, denoted by $D_1 \equiv D_2$, if D_1 generalizes D_2 and D_2 generalizes D_1. Intuitively, this means that these two methods do not differ significantly from each other.

We assume the reader is familiar with the main structural decomposition methods proposed in the literature (see, e.g., [Pearson and Jeavons, 1997]). Anyway, for the sake of presentation, we recall the definition of graph *treewidth*.

A *tree decomposition* [Robertson and Seymour, 1986] of a graph $G = (V, E)$ is a pair $\langle T, \chi \rangle$, where $T = (N, F)$ is a tree, and χ is a labelling function associating to each vertex $p \in N$ a set of vertices $\chi(p) \subseteq V$, such that the following conditions are satisfied: (1) for each vertex b of G, there exists $p \in N$ such that $b \in \chi(p)$; (2) for each edge $\{b, d\} \in E$, there exists $p \in N$ such that $\{b, d\} \subseteq \chi(p)$; (3) for each vertex b of G, the set $\{p \in N \mid b \in \chi(p)\}$ induces a (connected) subtree of T. (Connectedness condition.)
The *width* of the tree decomposition $\langle T, \chi \rangle$ is $\max_{p \in N} |\chi(p) - 1|$. The *treewidth* of G, $tw(G)$, is the minimum width over all its tree decompositions.

We next formally define how the dual graph $dual(\mathcal{H})$ of a hypergraph $\mathcal{H}$ can be simplified, by removing edges in a suitable way. For any edge $\{h, h'\}$ of a dual graph, let $\ell(\{h, h'\})$ denote the set of variables $h \cap h'$ that the hyperedges h and

Definition 2.1 *Let $G = (V, E)$ be the dual graph of some hypergraph $\mathcal{H}$. A* reduct *G' of G is a graph (V', E') such that $V' = V$, $E' \subseteq E$, and the following condition holds: for each edge $q = \{h, h'\}$ belonging to $(E - E')$, there exists in G' a path P from h to h', such that the variables in $\ell(q)$ are included in $\ell(q')$ for each edge q' occurring in the path P. That is, if all the variables shared by h and h' occur in some other path between these vertices, the edge connecting them can be deleted from the dual graph.*

Let h be a hyperedge of $\mathcal{H}$, and G be a reduct of $dual(\mathcal{H})$. Then, we denote with $v_G(h)$ the vertex of G' corresponding to the hyperedge h. Moreover, given a set of hyperedges H, we denote by $v_G(H)$ the set of vertices of G' associated with the elements in H. Whenever no confusion arises, $v_G(h)$ (resp. $v_G(H)$) will be denoted simply by $v(h)$ (resp. $v(H)$). Finally, we denote by $dual_G(H)$ the subgraph of G induced by the nodes in $v_G(H)$.

A non-binary CSP can be solved by applying any binary decomposition method, say D, to any reduct of the dual graph. However, the performance of this method depends in general on the particular reduct selected for solving the problem. We are interested in the reduct that guarantees the best possible performances, when the method D is applied to it. Formally, given a hypergraph $\mathcal{H}$, define D^{optd} as the method D applied to the reduct of $dual(\mathcal{H})$ having the minimum D-width over all the possible reducts of $dual(\mathcal{H})$.

It can be seen that the strongly generalization results between graph-based decomposition methods proved by Gottlob et al. [2000] can be extended to their *optd* versions.

Theorem 2.2 *The following relationships hold:*

- $\texttt{BICOMP}^{optd} \prec\prec \texttt{TREEWIDTH}^{optd}$.
- $\texttt{CUTSET}^{optd} \prec\prec \texttt{TREEWIDTH}^{optd}$.

Thus, $\texttt{TREEWIDTH}^{optd}$ seems a good candidate for solving non-binary CSP, as it strongly generalizes all the other graph-based methods. However, we recall that all the known algorithms for computing a k-bounded tree decomposition of a graph are exponential in k (even if they are polynomial for any fixed constant *k),* while computing the biconnected components of a graph is a linear task, independently of their width. This latter technique can be therefore very useful if the size of the structure and the bound *k* are large and the powerfun methods like TREEWIDTH are too expensive. In the next section, we face the problem of computing optimal reducts of the dual graph w.r.t. the BICOMP method.

3 Hinges VS Biconnected Components

In [Gyssens *et al,* 19941, it has been shown that the HINGE method generalizes BICOMP applied to any reduct of the dual graph. Anyhow, in the same paper, Gyssens et al. observed that a fine comparison between the two methods is quite difficult, as there is no obvious way to find a suitable reduct of the dual graph to keep the biconnected width small. Here we complete the picture by showing that, in fact, hinge decompositions correspond to such clever simplifications of the dual graph.

Lemma 3.1 $\texttt{HINGE} \preceq \texttt{BICOMP}^{optd}$.

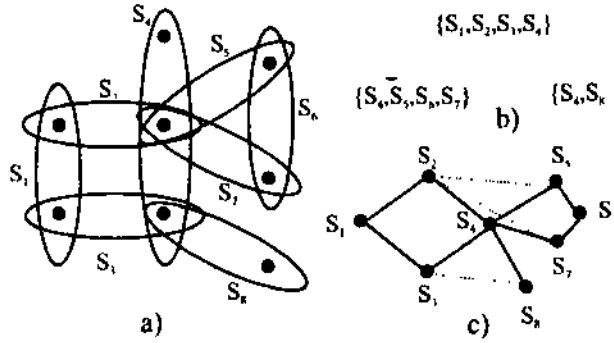


Figure 1: (a) a hypergraph $\mathcal{H}'$, (b) a hinge decomposition of $\mathcal{H}'$, and (c) a reduct of $dual(\mathcal{H}')$

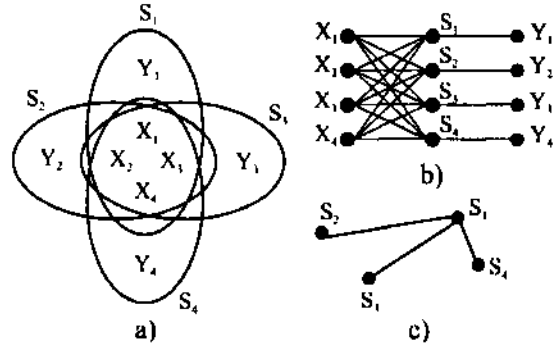


Figure 2: (a) The hypergraph $Rose(4)$, (b) its incidence graph, and (c) a reduct of $dual(Rose(4))$

Proof. (Sketch.) Let $\mathcal{H}$ be a hypergraph and T a hinge decomposition. Let G be a graph constructed by removing from $dual(\mathcal{H})$ each edge $\{v(h), v(h')\}$ such that there exists two distinct nodes of T, say H and H', such that $h \in H$ and $h' \in H'$.

We claim that i) G is a reduct of $dual(\mathcal{H})$, and ii) for each hinge H in T, there exists a biconnected component C of G, such that C is the subgraph of G induced by the nodes $v_G(H)$, i.e. $C = dual_G(H)$. Moreover, every biconnected component of G corresponds to some hinge, because each node of G is an edge of the original hypergraph, and thus must be contained in some hinge of T, by definition of hinge decomposition. Finally, note that the size (number of edges) of any hinge H, is the same as the size (number of nodes) of the corresponding biconnected component $C = dual_G(H)$. □

Combining the above lemma and the results in [Gyssens *et al.*, 19941, it follows that these two methods identifies the same classes of tractable CSPs.

Theorem 3.2 $\texttt{BICOMP}^{optd} \equiv \texttt{HINGE}$.

It is worthwhile noting that, given any hinge decomposition of a hypergraph *H,* the proof of Lemma 3.1 provides in fact an algorithm for computing an optimal reduct of *dual(H)* with respect to the BICOMP method. For instance, Figure 1 shows a hypergraph *H'* a hinge decomposition H' and the optimal reduct of the dual graph obtained by applying the above construction, where dotted edges represent the edges of the dual graph removed in this simplification. Note that the biconnected components of this graph correspond to the hinges of the given decomposition. Since biconnected components can be computed in linear time, it follows that a $\text{BICOMP}^{\text{Optd}}$ $O(|\mathcal{N}(\mathcal{H})||\mathcal{E}(\mathcal{H})|^2)$, which is the best known upper bound for computing a hinge decomposition.

Observe that the above result, together with Theorem 2.2, gives us a new insight of the power of $\texttt{TREEWIDTH}^{optd}$. Indeed, while the tree decomposition method applied to the primal graph is incomparable with HINGE [Gottlob *et al.*, 2000], its application to an optimal reduct of the dual graph strongly generalizes HINGE.

Corollary 3.3 $\texttt{HINGE} \prec\!\!\prec \texttt{TREEWIDTH}^{optd}$.

4 Hidden-Variables VS Dual Graph

It is well known that both the dual graph and the hidden-variable (incidence-graph) representations may be used for identifying tractable classes of non-binary CSPs according to the tree decomposition method (see, e.g., [Kolaitis and Vardi, 2000]). However, it was not clear whether either of these methods generalizes the other one or beats the other one on some classes of CSPs. In this section, we precisely characterize the relationship between $\text{TREEWIDTH}^{\text{Optd}}$ and $\text{TREEWIDTH}^{\text{m}}$. First, we observe that there is a CSP class where $\text{TREEWIDTH}^{\text{Optd}}$ is definitely better than $\text{TREEWIDTH}^{\text{in}}$.

Theorem 4.1 $\texttt{TREEWIDTH}^{optd} \rhd \texttt{TREEWIDTH}^{in}$.

Proof. (Sketch.) We show that there is a class of hypergraphs that is tractable w.r.t. $\texttt{TREEWIDTH}^{optd}$, but not w.r.t. $\texttt{TREEWIDTH}^{in}$. For any $n > 0$, let $Rose(n)$ be the hypergraph having n edges $\{h_1, ..., h_n\}$ over the nodes $\{X_1, ..., X_n, Y_1, ..., Y_n\}$, such that $h_i = \{X_1, ..., X_n, Y_i\}$, for each $0 < i \leq n$. Figure 2 shows the graph $Rose(4)$, its incidence graph and a reduct of its dual graph. It can be seen that any optimal reduct of $dual(Rose(n))$ is a tree and hence its width according to $\texttt{TREEWIDTH}^{optd}$ is 1. On the other hand, the bipartite graph $inc(Rose(n))$ contains a clique of size n and hence its treewidth is $n - 1$. It follows that the class $\{Rose(n) \mid n > 0\}$ has unbounded treewidth. □

Note that any hypergraph $Rose(n)$ in the above proof is an acyclic hypergraph. It follows that the class of all acyclic CSP instances is not tractable according to $\texttt{TREEWIDTH}^{in}$.

Even though $\texttt{TREEWIDTH}^{optd}$ beats $\texttt{TREEWIDTH}^{in}$, we next shows that $\texttt{TREEWIDTH}^{optd}$ does not generalize $\texttt{TREEWIDTH}^{in}$.

Theorem 4.2 $\texttt{TREEWIDTH}^{in} \not\prec\!\!\prec \texttt{TREEWIDTH}^{optd}$.

Proof. (Sketch.) We show that there is a class of hypergraphs $Subset(k)$ such that the treewidth of the incidence graph is k, but the treewidth of any optimal reduct of $dual(Subset(k))$ is $(k/3)^2$. For any k, the hypergraph $Subset(k)$ is such that $\mathcal{N}(Subset(k)) = X \cup Y \cup Z$, where $X = \{X_1, ..., X_m\}$, $Y = \{Y_1, ..., Y_m\}$, and $Z = \{Z_1, ..., Z_m\}$, with $m = k/3$.

There are m^2 distinct edges, each containing $m - 1$ nodes from X, $m - 1$ nodes from Y, and $m - 1$ nodes from Z. In particular, each edge e contains one of the m subsets of $m - 1$

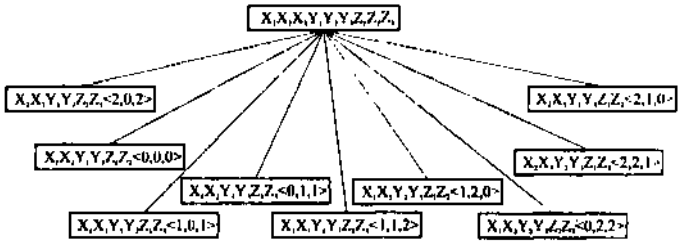


Figure 3: A tree decomposition of $inc(Subset(9))$.

elements from A' and one the m subsets of *m* - 1 elements from Y These subsets may be identified by means of two integers, say *a* and *b,* ranging from 0 to *m* — 1. Similarly, the edge contains a subset of variables from *Z* identified by an integer *c* and functionally determined by *a* and b as follows: *c — (a + b)mod m.* Thus, each edge *e* is simply denoted as a triple (a, b,c).

Note that the incidence treewidth of *Subset(k)* is *k.* For instance, Figure 3 shows a tree decomposition of the incidence graph of *Subset(9),* where *in* — 3. Moreover, it can be proved that the dual graph *dual(Subset(k))* cannot be reduced and contains a clique of size m^2 — $(k/3)^2$. It follows that its treewidth is $(k/3)^2$. D

To give a complete picture of the relationship between TREEWIDTH^{ptd} and TREEWIDTH^{in}, we next show that any decomposition of the incidence graph with width k can be modified to be a decomposition of a reduct of the dual graph having width at most 2^k Thus, TREEWIDTH^{n1} does not beat TREEWIDTH^{0ptd}.

The algorithm is shown in Figure 4. It takes a tree decomposition $TD = \langle T, \chi \rangle$ of $inc(\mathcal{H})$, and outputs a tree decomposition $TD' = \langle T, \chi' \rangle$ of a suitable reduct of $dual(\mathcal{H})$.

It performs a breadth-first visit of T. For each visited node p, we compute the labelling $\chi'(p)$ of this node in the new tree decomposition TD' from its label $\chi(p)$ in the given tree decomposition TD: Let $\chi(p) = V_p \cup C_p$, where $V_p \subseteq \mathcal{N}(\mathcal{H})$ and $C_p \subseteq \mathcal{E}(\mathcal{H})$. Then, $\chi'(p)$ contains all edges in C_p plus one arbitrarily chosen edge $h_{V'}$ that covers V' (i.e., such that $V' \subseteq h_{V'}$), for each subset V' of V_p. This way, p is now labelled only by edges of $\mathcal{H}$ (i.e., nodes of the dual graph). Moreover, the algorithm forces all such edges $h_{V'}$ to belong to the χ' labellings of any vertex q in the subtree rooted at p such that $V' \subseteq \chi(q)$.

Note that, if $\langle T, \chi \rangle$ has width k, then $\langle T, \chi' \rangle$ has width at most 2^k. Then, from this algorithm (whose proof of correctedness is omitted), we get the following relationship.

Theorem 4.3 $\text{TREEWIDTH}^{in} \not\triangleright \text{TREEWIDTH}^{optd}$.

5 A 2-approximation of TREEWIDTH^{optd}

In this section, we face the problem of computing an optimal reduct of a dual graph in order to get the minimum possible treewidth. We recall that it is not known whether this problem is feasible in polynomial time or not [Kolaitis and Vardi, 2000]. Moreover, we provide a qualitative explanation of the remarkable difference between TREEWIDTH^{0ptd}

Input: A tree decomposition $\langle T, \chi \rangle$ of $inc(\mathcal{H})$;
Output: A tree decomposition $\langle T, \chi' \rangle$ of a reduct of $dual(\mathcal{H})$,

for all p in T **and all** $V' \subseteq \chi(p)$ **do**
  $cover(V') := \emptyset$,
$queue := root(T)$,
while $queue \neq \emptyset$ **do**
  extract p from $queue$;
  $V_p := \chi(p) \cap \mathcal{N}(\mathcal{H})$;
  $C_p := \chi(p) \cap \mathcal{E}(\mathcal{H})$;
  $\chi'(p) := C_p$;
  for all $V' \subseteq V_p$ s.t. there is $h \in \mathcal{E}(\mathcal{H})$ s.t. $V' \subseteq h$ **do**
    if $cover(V') = \emptyset$ **then**
      choose any $h \in \mathcal{E}(\mathcal{H})$ s.t. $V' \subseteq h$;
      $\chi'(p) := \chi'(p) \cup \{h\}$,
      $cover(V') := \{h\}$,
    else
      $\chi'(p) := \chi'(p) \cup cover(V')$;
    end if
  insert($queue$,$children(p)$);
end while
return $\langle T, \chi' \rangle$

Figure 4: **Algorithm** incidenceToDualTreeDecomposition

and hypergraph-based notions such as query width and hypertree width.

Let $k > 0$ be a fixed constant and $\mathcal{H}$ be a constraint hypergraph. Figure 5 shows the Algorithm k-TREE-APPROX that, if the optimal treewidth of $dual(\mathcal{H})$ is at most k, outputs in polynomial time a tree decomposition of width at most $2k$ of some reduct of $dual(\mathcal{H})$.

Let S be a subset of nodes of $dual(\mathcal{H})$. A non-empty set C of nodes of $dual(\mathcal{H})$ is [S]-*connected* if, for each pair $h, h' \in C$, there exists a path $h = h_1, \ldots, h_n = h'$ in $dual(\mathcal{H})$ such that $h_i \cap h_{i+1} \not\subseteq z$, for every $z \in S$. The [S]-*components* of $dual(\mathcal{H})$ are the maximal [S]-*connected* sets of nodes of $dual(\mathcal{H})$.

Roughly, k-TREE-APPROX is based on a recursive procedure *kDecomp* that, given a vertex p and a $[\chi(p)]$ - *component* C to be "decomposed," computes the labelling $\chi(q)$ of a child q of p chosen to deal with C. Then, it recursively call itself for computing the subtree rooted at q that should cover all the $\chi(q)$-components included in C.

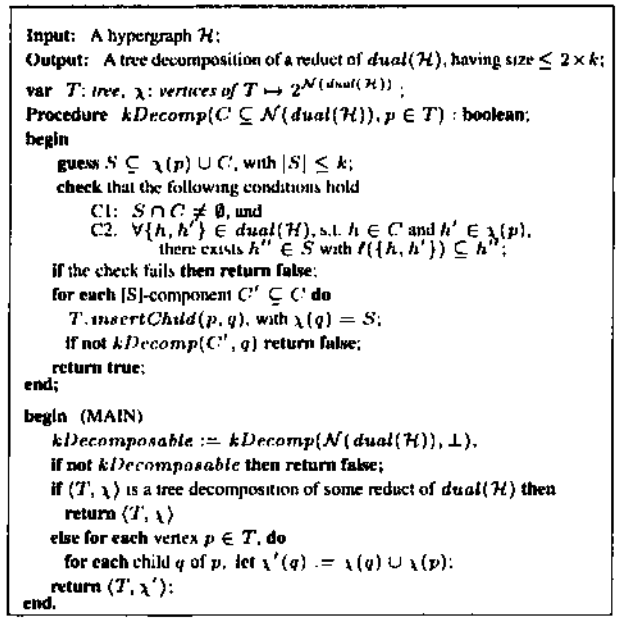
Finally, if the output $\langle T, \chi \rangle$ of *kDecomp* is not a tree decomposition of some reduct of $dual(\mathcal{H})$, then the MAIN module of the algorithm computes a new labelling χ' that is guaranteed to be a legal tree decomposition.

The proof of the following theorem is quite involved and will be omitted for space limitation.

Theorem 5.1 *Let k be a fixed integer. Given any hypergraph $\mathcal{H}$, if there exists a reduct of $dual(\mathcal{H})$ having treewidth bounded by k, then k-TREE-APPROX computes in* L^{LOGCFL} *a tree decomposition of some reduct of $dual(\mathcal{H})$ having width at most $2k$.*

It is worthwhile noting that the output $\langle T, \chi \rangle$ of *kDecomp*, even if it is not a tree decomposition, is in fact a new kind of structural decomposition, defined below, that we call *weak query decomposition.*

Definition 5.2 *A* weak query decomposition *of a hypergraph $\mathcal{H}$ is a pair $\langle T, \lambda \rangle$, where $T = (N, E)$ is a tree, and λ is a*



```
Input: A hypergraph H;
Output: A tree decomposition of a reduct of dual(H), having size ≤ 2×k;
var T: tree, λ: vertices of T ↦ 2^{N(dual(H))} ;
Procedure kDecomp(C ⊆ N(dual(H)), p ∈ T) : boolean;
begin
    guess S ⊆ λ(p) ∪ C, with |S| ≤ k;
    check that the following conditions hold
        C1: S ∩ C ≠ ∅, and
        C2. ∀{h, h'} ∈ dual(H), s.t. h ∈ C and h' ∈ λ(p),
                there exists h'' ∈ S with ℓ({h, h'}) ⊆ h'';
    if the check fails then return false;
    for each [S]-component C' ⊆ C do
      T.insertChild(p, q), with λ(q) = S;
      if not kDecomp(C', q) return false;
    return true;
end;

begin (MAIN)
    kDecomposable := kDecomp(N(dual(H)), ⊥).
    if not kDecomposable then return false;
    if ⟨T, λ⟩ is a tree decomposition of some reduct of dual(H) then
      return ⟨T, λ⟩
    else for each vertex p ∈ T, do
      for each child q of p, let λ'(q) := λ(q) ∪ λ(p);
    return ⟨T, λ'⟩;
end.
```

Figure 5: Algorithm k-TRLE-APPROX

labelling function which associates to each vertex $p \in N$ a set $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$, such that the following conditions hold:

1. *for each edges h of $\mathcal{H}$, there exists $p \in N$ such that $h \in \lambda(p)$;*
2. *for each edge h of $\mathcal{H}$, the set $\{p \in N \mid h \in \lambda(p)\}$ induces a (connected) subtree of T;*
3. *for each pair of vertices v, v' of T and each pair of edges $h \in \lambda(v)$, $h' \in \lambda(v')$ there is a path P connecting v and v' in T such that, for each v in P, there exists $h'' \in \lambda(v)$ such that $\ell(\{h, h'\}) \subseteq h''$.*

The width *of the weak query decomposition $\langle T, \lambda \rangle$ is $max_{p \in N}|\lambda(p)|$. The* weak query-width *$wqw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its weak query decompositions.*

Note that Condition 3 above entails the usual *connectedness condition* 3': for each variable $Y \in \mathcal{N}(\mathcal{H})$, the set $\{p \in N \mid Y$ occurs in some edge $h \in \lambda(p)\}$ induces a (connected) subtree of T. Thus, any weak query decomposition of a hypergraph is also a query decomposition of the hypergraph, and thus any class of CSPs having bounded weak query-width is tractable, i.e., can be solved in polynomial time. However, unlike query decompositions [Gottlob *et al.*, 2002], weak query decompositions are efficiently computable. Indeed, LOGCFL is not only included in polynomial time, but also in NC^2, and thus it contains highly parallelizable problems. Moreover, this new notion has essentially the same "decomposition power" as $\texttt{TREEWIDTH}^{optd}$.

Theorem 5.3 *For any hypergraph $\mathcal{H}$,*

$$wqw(\mathcal{H}) \leq tw^{optd}(\mathcal{H}) \leq 2wqw(\mathcal{H}).$$

Thus, $\texttt{TREEWIDTH}^{optd}$ and the notion of weak query decomposition are almost equivalent, up to a bounded difference in favor of the latter method.

Remark. The results in this section shed some light on the difference between the "decomposition power" of TREEWIDTH^{optd} and the strictly more general methods of query and hypertree decompositions. Note that the crucial condition to be maintained in all these tree-structured decompositions is the connectedness condition for the constraint variables, and looking at Condition 3 and Condition 3' above we can make the following observation: while in the weak query decompositions (and similarly in TREEWIDTH^{optd} each hypcredge in the labelling of a tree node plays an independent role, in query decompositions (and hypertree decompositions) all such edges contribute together to maintain the connectedness condition. That is, these hypergraph based notions exploit the union of the hyperedges labelling any tree-node, while the TREEWIDTH^{optd} and the weak query width methods exploit the power of each hyperedge separately.

References

[Bacchus *et al.,* 2002] F. Bacchus, X. Chen, P. van Beck, and T. Walsh. Binary vs Non-Binary Constraints. *Artificial Intelligence,* 140: (1-2), 1-37, 2002.

[Chekuri and Rajaraman, 2000] Ch. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *Theoretical Computer Science,* 239(2):211-229, 2000.

[Dechter, 1992] R. Dechter. Constraint Networks. In *Encyclopedia of Artificial Intelligence,* second edition, Wiley and Sons, pp. 276-285, 1992.

[Dechter and Pearl, 1988] R. Dechter and J. Pearl. Network based heuristics for constraint satisfaction problems. *Artificial Intelligence,* 34(1): 1-38, 1988.

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

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

[Gottlob *et al,* 2000] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence,* 124(2): 243-282,2000.

[Gottlob *et al,* 2002] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences,* 64(3): 579-627, 2002.

[Gyssens et al., 1994] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence,* 66:57-89, 1994.

[Kolaitis and Vardi, 2000] Ph. G. Kolaitis and M. Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences,* 61(2): 302-332,2000.

[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.