Tractable Optimization Problems through Hypergraph-Based Structural Restrictions[*]

Georg Gottlob[1], Gianluigi Greco[2], and Francesco Scarcello[2]

[1] Oxford University
[2] University of Calabria
`georg.gottlob@comlab.ox.ac.uk`, `ggreco@mat.unical.it`,
`scarcello@deis.unical.it`

Abstract. Several variants of the Constraint Satisfaction Problem have been proposed and investigated in the literature for modelling those scenarios where solutions are associated with some given costs. Within these frameworks computing an optimal solution is an NP-hard problem in general; yet, when restricted over classes of instances whose constraint interactions can be modelled via (nearly-)acyclic graphs, this problem is known to be solvable in polynomial time. In this paper, larger classes of tractable instances are singled out, by discussing solution approaches based on exploiting hypergraph acyclicity and, more generally, structural decomposition methods, such as (hyper)tree decompositions.

1 Introduction

The Constraint Satisfaction Problem (CSP) is a well-known framework [11] for modelling and solving search problems, which received considerably attention in the literature due to its applicability in various areas. Informally, a CSP instance is defined by singling out the variables of interest, and by listing the allowed combinations of values for groups of them, according to the constraints arising in the application at hand. The solutions for this instance are the assignments of domain values to variables that satisfy all such constraints. Many apparently unrelated problems from disparate areas actually turn out to be equivalent to the CSP and can be accommodated within the CSP framework. Examples are puzzles, conjunctive queries over relational databases, graph colorability, and checking whether there is a homomorphism between two finite structures.

Example 1. Figure 1 shows a combinatorial crossword puzzle (taken from [15]). A set of legal words is associated with each horizontal or vertical array of white boxes delimited by black boxes. A solution to the puzzle is an assignment of a letter to each white box such that to each white array is assigned a word from its

---
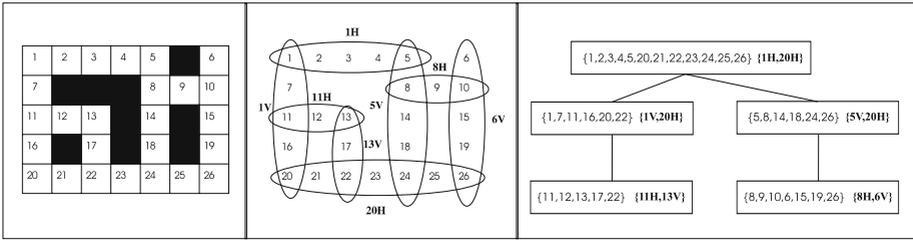
Fig. 1. A crossword puzzle, its associated hypergraph $\mathcal{H}_{cp}$, and a hypertree decomposition of width 2 for $\mathcal{H}_{cp}$

set of legal words. This problem can be recast in a CSP by associating a variable with each white box, and by defining a constraint for each array of white boxes prescribing the legal words that are associated with it.   ◁

When assignments are associated with some given cost, however, computing an arbitrary solution might not be enough. For instance, the crossword puzzle in Figure 1 may admit more than one solution, and expert solvers may be asked to single out the most difficult ones, such as those solutions that minimize the total number of vowels occurring in the used words. In these cases, one is usually interested in the corresponding *optimization problem* of computing the solution of minimum cost, whose modeling is accounted for in several variants of the basic CSP framework, such as fuzzy, probabilistic, weighted, lexicographic, valued, and semiring-based CSPs (see [25,4] and the references therein).

Since solving CSPs—and the above extensions—is an NP-hard problem, much research has been spent to identify restricted classes over which solutions can efficiently be computed. In this paper, *structural decomposition methods* are considered [15], which identify tractable classes by exploiting the structure of constraint scopes as it can be formalized either as a hypergraph (whose nodes correspond to the variables and where each group of variables occurring in some constraint induce a hyperedge) or as a suitable binary encoding of such hypergraph. In particular, we focus on the structural methods based on the notions of (generalized) hypertree width [18,19] and treewidth [28]. In both cases, the underlying idea is that solutions to CSP instances that are associated with acyclic (or nearly-acyclic) structures can efficiently be computed via dynamic programming, by incrementally processing the structure according to some of its topological orderings.

As a matter of fact, however, while in the case of classical CSPs deep and useful results have been achieved for both graph and hypergraph representations, in the case of CSP extensions tailored for optimization problems attention was mainly focused on binary encodings and, in particular, on the *primal graph* representation, where nodes correspond to variables and an edge between two variables indicates that they are related by some constraint. Discussing whether (and how) hypergraph-based structural decomposition techniques in the literature can be lifted to such optimization frameworks is the main goal of this paper. In particular, we consider three CSP extensions:

(1) First, we consider optimization problems where every mapping variable-value is associated with a cost, so that the aim is to find an assignment satisfying all the constraints and having the minimum total cost.

(2) Second, we consider the case where costs are associated with the allowed combinations of simultaneous values for the variables occurring in the constraint, rather than to individual values. Again, within this setting, we consider the problem of computing a solution having minimum total cost.

(3) Finally, we consider a scenario where the CSP instance at hand might not admit a solution at all, and where the problem is hence to find the assignment minimizing the total number of violated constraints (and, more generally, whenever a cost is assigned to each constraint, the assignment minimizing the total cost of violated constraints).

For each of the above settings, the complexity of computing the optimal solution is analyzed in this paper, by overviewing some relevant recent research and by providing novel results. In particular:

▶ We show that optimization problems of kind **(1)** can be solved in polynomial time on instances having bounded (generalized) hypertree-width hypergraphs. This result is based on an algorithm recently designed and analyzed in the context of *combinatorial auctions* [13].

▶ We show that even optimization problems of kind **(2)** are tractable on instances having bounded (generalized) hypertree-width hypergraphs. Indeed, we describe how to transform this kind of instances into equivalent instances of kind **(1)**, by preserving their structural properties.

▶ We observe that optimization problems of kind **(3)** remain NP-hard even over instances having an associated acyclic hypergraph. However, there is also good news: they are shown to be tractable on instances having bounded treewidth *incidence graph* encoding. The latter is a binary encoding of the constraint hypergraph with usually better structural features than the primal graph encoding (see, e.g., [15,22]). Again, proof is via a mapping to case **(1)**.

Organization. The rest of the paper is organized as follows. Section 2 discusses preliminaries on CSPs and structural restrictions, and Section 3 provides an overview of the structural decomposition methods based on treewidth and (generalized) hypertree width. Results for optimization problems of kind **(1)** and **(2)** are discussed in Section 4, whereas problems of kind **(3)** are discussed in Section 5. Finally, Section 6 draws our conclusions.

2   CSPs, Acyclic Instances, and Their Desirable Properties

An instance of a constraint satisfaction problem [11] is a triple $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$, 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_v, for $1 \leq v \leq q$, is a pair (S_v, r_v), where $S_v \subseteq Var$ is a set of variables called the *constraint scope*, and r_v is a set of substitutions (also called *tuples*) from variables in S_v
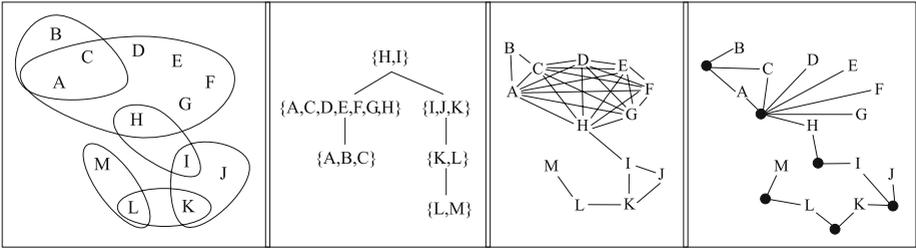
Fig. 2. A hypergraph $\mathcal{H}_1$, a join tree $JT(\mathcal{H}_1)$, the primal graph $G(\mathcal{H}_1)$, and the incidence graph $inc(\mathcal{H}_1)$

to values in U indicating the allowed combinations of simultaneous values for the variables in S_v. Any substitution from a set of variables $V \subseteq Var$ to U is extensively denoted as the set of pairs of the form X/u, where $u \in U$ is the value to which $X \in V$ is mapped. Then, a solution to $\mathcal{I}$ is a substitution $\theta : Var \mapsto U$ for which q-tuples $t_1 \in r_1, ..., t_q \in r_q$ exist such that $\theta = t_1 \cup ... \cup t_q$.

Example 2. In the crossword puzzle of Figure 1, *Var* coincides with the letters of the alphabet, and a variable X_i (denoted by its index i) is associated with each white box. An example of constraint is $C_{1H} = ((1, 2, 3, 4, 5), r_{1H})$, and a possible instance for r_{1H} is $\{\langle h, o, u, s, e\rangle, \langle c, o, i, n, s\rangle, \langle b, l, o, c, k\rangle\}$—in the various constraint names, subscripts H and V stand for "Horizontal" and "Vertical," respectively, resembling the usual naming of definitions in crossword puzzles. $\lhd$

The structure of a CSP instance $\mathcal{I}$ is best represented by its associated hypergraph $\mathcal{H}(\mathcal{I}) = (V, H)$, where $V = Var$ and $H = \{S \mid (S, r) \in \mathcal{C}\}$—in the following, V and H will be denoted by $\mathcal{N}(\mathcal{H})$ and $\mathcal{E}(\mathcal{H})$, respectively. As an example, the hypergraph associated with the crossword puzzle formalized above is illustrated in the central part of Figure 1.

A hypergraph $\mathcal{H}$ is *acyclic* iff it has a join tree [3]. A *join tree* $JT(\mathcal{H})$ for a hypergraph $\mathcal{H}$ is a tree whose vertices are the hyperedges of $\mathcal{H}$ such that, whenever the same node $X \in V$ occurs in two hyperedges h_1 and h_2 of $\mathcal{H}$, then X occurs in each vertex on the unique path linking h_1 and h_2 in $JT(\mathcal{H})$. The notion of acyclicity we use here is the most general one known in the literature, coinciding with α-acyclicity according to Fagin [9]. Note that the hypergraph $\mathcal{H}_{cp}$ of Figure 1 is not acyclic. An acyclic hypergraph is discussed below.

Example 3. Consider the hypergraph $\mathcal{H}_1$ shown on the left of Figure 2, which is associated with a CSP instance over the set of variables $\{A, ..., M\}$. In particular, six constraints are defined over the instance whose scopes precisely correspond to the hyperedges in $\mathcal{E}(\mathcal{H}_1)$; for instance, $\{A, B, C\}$ is an example of constraint scope. Note also that $\mathcal{H}_1$ is acyclic. Indeed, a join tree $JT(\mathcal{H}_1)$ for it is reported in the same figure to the right of $\mathcal{H}_1$. $\lhd$

An important property of acyclic instances is that they can efficiently be processed by dynamic programming. Indeed, according to Yannakakis' algorithm [34] (originally conceived in the equivalent context of evaluating acyclic

Boolean conjunctive queries), they can be evaluated by processing any of their join trees bottom-up, by performing upward semijoins between the constraint relations, thus keeping the size of the intermediate result small. At the end, if the constraint relation associated with the root atom of the join tree is not empty, then the CSP instance does admit a solution. Therefore, the whole procedure is feasible in $O(n \times r_{max} \times \log r_{max})$, where n is the number of constraints and r_{max} denotes the size of the largest constraint relation.

In addition to the polynomial time algorithm for deciding whether a CSP admits a solution, acyclic instances enjoy further desirable properties:

Acyclicity is efficiently recognizable: Deciding whether a hypergraph is acyclic is feasible in linear time [31] and belongs to the class L (deterministic logspace). Indeed, this follows from the fact that hypergraph acyclicity belongs to SL [16], and that SL is equal to L [27].

Acyclic instances can be efficiently solved: After the bottom-up step described above, one can perform the reverse top-down step by filtering each child vertex from those tuples that do not match with its parent tuples. The relations obtained after the top-down step enjoy the *global consistency* property, i.e., they contain only tuples whose values are part of some solution of the CSP. Then, all solutions can be computed with a backtrack-free procedure, and thus in *total polynomial time*, i.e., in time polynomial in the input plus the output [34] (and actually also with polynomial delay). Alternatively, one may enforce pairwise consistency by taking the semijoins between all pairs of relations until a fixpoint is reached. Indeed, acyclic instances that fulfil this property also fulfil the global consistency property [2].

Acyclic instances are parallelizable: It has been shown that solving acyclic CSP instances is highly parallelizable, as this problem (actually, deciding the existence of a solution) is complete for the low complexity class LOGCFL [16]. Efficient parallel algorithms are discussed in [16] and [17].

We conclude this section by recalling that the above desirable properties of acyclic CSP instances have profitably been exploited in various application scenarios. Indeed, besides their application in the context of Database Theory, they found applications in Game Theory [14,8], Knowledge Representation and Reasoning [21], and Electronic Commerce [13], just to name a few.

3   Generalizing Acyclicity

Many attempts have been made in the literature for extending the good results about acyclic instances to relevant classes of *nearly acyclic* structures. We call these techniques *structural decomposition methods*, because they are based on the "acyclicization" of cyclic (hyper)graphs. We refer the interested reader to [29] for a detailed description of how these techniques may be useful for constraint satisfaction problems and to [22] for further results about graph-based techniques, when relational structures are represented according to various graph representations (primal graph, dual graph, incidence-graph encoding). We also

want to mention recent methods such as Spread-cuts [7] and fractional hypertree decompositions [23].

A survey of most of these techniques is currently available in Wikipedia (look for "decomposition method", at `http://www.wikipedia.org`). In the sequel, we shall briefly overview the tree and hypertree decomposition methods.

3.1 Tree Decompositions

For classes of instances having only binary constraints or, more generally, constraints whose scopes have a fixed maximum arity, the most powerful structural method is based on the notion of treewidth.

Definition 1 ([28]). A *tree decomposition* 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 assigning with each vertex $p \in N$ a set of vertices $\chi(p) \subseteq V$ such that the following conditions are satisfied: (1) for each node 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 $\{b, d\} \subseteq \chi(p)$; and, (3) for each node b of G, the set $\{p \in N \mid b \in \chi(p)\}$ induces a connected subtree of T (*connectedness condition*). The *width* of $\langle T, \chi \rangle$ is the number $\max_{p \in N}(|\chi(p)|-1)$. The *treewidth* of G, denoted by $tw(G)$, is the minimum width over all its tree decompositions. □

It is well-known that a graph G is acyclic if and only if $tw(G) = 1$. Moreover, for any fixed natural number $k > 0$, deciding whether $tw(G) \leq k$ is feasible in linear time [5].

Any CSP with primal graph G such that $tw(G) \leq k$ can be (efficiently) turned into an equivalent CSP whose primal graph is acyclic. Let $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$ be a CSP instance, let G be the primal graph of $\mathcal{H}(\mathcal{I})$, and let $\langle T, \chi \rangle$ be a tree decomposition of G having width k. We may build a new acyclic CSP instance $\mathcal{I}' = \langle Var, U, \mathcal{C}' \rangle$ over the same variables and universe as $\mathcal{I}$, but with a different set of constraints $\mathcal{C}'$, as follows. Firstly, for each vertex v of T, we create a constraint (S_v, r_v), where $S_v = \chi(v)$ and $r_v = U^{|\chi(v)|}$. Then, for every constraint $(S, r) \in \mathcal{C}$ of the original problem such that $S \subseteq \chi(v)$, we eliminate from r_v all those tuples that do not match with r. The resulting constraint is then added to $\mathcal{C}'$. It can be shown that $\mathcal{I}'$ has the same solutions as $\mathcal{I}$, and that it is acyclic. In fact, observe that, by construction, $\langle T, \chi \rangle$ is a join tree of the hypergraph $\mathcal{H}(\mathcal{I}')$ associated with $\mathcal{I}'$, because of the connectedness condition of tree decompositions. Furthermore, building $\mathcal{I}'$ from $\mathcal{I}$ is feasible in $O(n \times |U|^{k+1})$ where n is the number of vertices in T, and where the size of the largest constraint relation in the resulting instance is $|U|^{k+1}$. Since one can always consider only tree decompositions whose number of vertices is bounded by the number of variables of the problem (i.e., the nodes of the graph), it follows that deciding whether $\mathcal{I}'$ (and hence $\mathcal{I}$) is satisfiable is feasible in $O(|Var| \times |U|^{k+1} \times \log |U|^{k+1})$. In fact, as for acyclic instances, even in this case we may compute also solutions for $\mathcal{I}$ with a backtrack-free search, after the preprocessing of the instance performed according to the given tree decomposition (i.e., according to

the join tree of the equivalent acyclic instance). As a consequence, all classes of CSP instances (with primal graphs) having bounded treewidth may be solved in polynomial time, even if with an exponential dependency on the treewidth.

Clearly enough, this technique is not very useful for CSP instances with large constraint scopes. In particular, the class of CSP instances whose associated constraint hypergraphs are acyclic are not tractable according to tree decompositions, because acyclic hypergraphs may have unbounded treewidth. Intuitively, in the primal graph all variables occurring in the same constraint scope are connected to each other, and thus they lead to a clique in the graph. It follows that CSP instances having constraint scopes with large arities have large treewidths, too, because the treewidth of a clique of n nodes is $n-1$. As an example, Figure 2 reports the graph $G(\mathcal{H}_1)$ associated with the acyclic hypergraph $\mathcal{H}_1$, where one may notice how the hyperedge $\{A, C, D, E, F, G, H\}$ is flattened into a clique over all its variables.

3.2   Hypertree Decompositions

Let us now turn our attention to hypergraph based decompositions. Such decompositions are similar to tree decompositions, but they use an additional covering of each set $\chi(p)$ with as few as possible hyperedges. The width is then no longer defined as the maximum cardinality of $\chi(p)$ over all decomposition nodes p, but as the maximum number of hyperedges used to cover $\chi(p)$. Intuitively, this notion of width is better, because it will allow us to expresses more accurately the computational effort needed to transform an instance into an acyclic one.

Definition 2 ([19]). A *generalized hypertree decomposition* of a hypergraph $\mathcal{H}$ is a triple $HD = \langle T, \chi, \lambda \rangle$, where $\langle T, \chi \rangle$ is a tree decomposition of the primal graph of $\mathcal{H}$, and λ is a labelling of the tree T by sets of hyperedges of $\mathcal{H}$ such that, for each vertex $p \in vertices(T)$, $\chi(p) \subseteq \bigcup_{h \in \lambda(v)} h$. That is, all variables in the χ labeling are covered by hyperedges (scopes) in the λ labeling. The *width* of HD is the number $\max_{p \in vertices(T)}(|\lambda(p)|)$. The *generalized hypertree width* of $\mathcal{H}$, denoted by $ghw(\mathcal{H})$, is the minimum width over all its generalized hypertree decompositions. If I is a CSP instance then $ghw(I) := ghw(\mathcal{H}(I))$.   □

Clearly, for each CSP instance I, $ghw(I) \leq tw(I)$. Moreover, there are classes of CSPs having unbounded treewidth whose generalized hypertree width is bounded[19].

Finding a suitable tree decomposition whose sets $\chi(p)$ may each be covered with a few hyperedges seems to be quite a hard task even in case we have some fixed upper bound k. Indeed, it has been shown that deciding whether $ghw(\mathcal{H}) \leq k$ is NP-complete (for any fixed $k \geq 3$) [20]. Fortunately, since its first proposal in [18], this notion comes with a tractable variant, called hypertree decomposition, whose associated width is at most 3 times (+1) larger than the generalized hypertree width [1]. As a consequence, it can be shown that every class of CSPs that is tractable according to generalized hypertree width is tractable according to hypertree width, as well.

Definition 3 ([18]). A *hypertree decomposition* of a hypergraph $\mathcal{H}$ is a generalized hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ that satisfies the following additional condition, called *Descendant Condition* or also *special condition*: $\forall p \in vertices(T)$, $\forall h \in \lambda(p)$, $h \cap \chi(T_p) \subseteq \chi(p)$, where T_p denotes the subtree of T rooted at p, and $\chi(T_p)$ the set of all variables occurring in the χ labeling of this subtree.

The *hypertree width* $hw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its hypertree decompositions.   □

As an example, on the right part of Figure 1 a hypertree decomposition of the hypergraph $\mathcal{H}_{cp}$ in Example 1 is reported. Note that this decomposition has width 2.

We refer the interested reader to [18,29] for more details about this notion and in particular about the descendant condition. Here, we just observe that the notions of hypertree width and generalized hypertree width are true generalizations of acyclicity, as the acyclic hypergraphs are precisely those hypergraphs having hypertree width and generalized hypertree width one. In particular, the classes of CSP instances having bounded (generalized) hypertree width have the same desirable computational properties as acyclic CSPs [16]. Indeed, from a CSP instance $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$ and a (generalized) hypertree decomposition HD of $\mathcal{H}(\mathcal{I})$ of width k, we may build an acyclic CSP instance $\mathcal{I}' = \langle Var, U, \mathcal{C}' \rangle$ with the same solutions as $\mathcal{I}$. The overall cost of deciding whether $\mathcal{I}$ is satisfiable is in this case $O((m-1) \times r_{max}^k \times \log r_{max}^k)$, where r_{max} denotes the size of the largest constraint relation and m is the number of vertices of the decomposition tree, with $m \leq |Var|$ (in that we may always find decompositions in a suitable normal form without redundancies, so that the number of vertices in the tree cannot exceed the number of variables of the given instance). To be complete, if the input consists of $\mathcal{I}$ only, we have to compute the decomposition, too. This can be done with a guaranteed polynomial-time upper bound in the case of hypertree decompositions [18].

In the following two sections, we provide some tractability results for optimization problems. For the sake of presentation, we give algorithms for the acyclic case, provided that these results may be clearly extended to any class of instances having bounded (generalized) hypertree width, after the above mentioned polynomial-time transformation.

4   Optimization Problems over CSP Solutions

In this section, we consider optimization problems where an assignment has to be singled out that satisfies all the constraints of the underlying CSP instance and that has minimum total cost; in other words, we look for a "best" solution among all the possible solutions. In particular, below, we shall firstly address the case where each possible variable-value mapping is associated with a cost (also called constraint satisfaction optimization problem); then we shall consider the case where costs are defined over the constraints tuples (weighted CSP).

Input: An acyclic CSOP instance $\langle \mathcal{I}, w \rangle$ with $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$, $\mathcal{C} = \{(S_1, r_1), ..., (S_q, r_q)\}$,
            and a join tree $T = (N, E)$ of the hypergraph $\mathcal{H}(\mathcal{I})$;
Output: A solution to $\langle \mathcal{I}, w \rangle$;
var $t^* : Var \mapsto U$;
  $\ell^v_{t_v}$: rational number, for each tuple $t_v \in r_v$;
  $t_{t_v, c}$: tuple in r_c, for each tuple $t_v \in r_v$, and for each $(v, c) \in E$;

Procedure $BottomUp$;
begin
  $Done :=$ the set of all the leaves of T;
  while $\exists v \in T$ such that (i) $v \notin Done$, and (ii) $\{c \mid c$ is child of $v\} \subseteq Done$ **do**
    $r_v := r_v - \{t_v \mid \exists (v, c) \in E$ such that $\forall t_c \in \theta_c,\ t_v \not\approx t_c\}$;
    if $r_v = \emptyset$ **then** EXIT;   (* $\mathcal{I}$ is not satisfiable *)
    for each $t_v \in r_v$ **do**
      $\ell^v_{t_v} := w(t_v)$;
      for each c such that $(v, c) \in E$ **do**
        $\bar{t}_c := \arg\min_{t_c \in r_c | t_v \approx t_c} \left(\ell^c_{t_c} - w(t_c \cap t_v) \right)$;
        $t_{t_v, c} := \bar{t}_c$;   (* set best solution *)
        $\ell^v_{t_v} := \ell^v_{t_v} + \ell^c_{\bar{t}_c} - w(\bar{t}_c \cap t_v)$;
      end for
    end for
    $Done := Done \cup \{v\}$;
  end while
end;

---

begin (* MAIN *)
  $BottomUp$;
  let r be the root of T;
  $\bar{t}_r := \arg\min_{t_r \in r_r} \ell^r_{t_r}$;
  $t^* := \bar{t}_r$;   (* include solution *)
  $TopDown(r, \bar{t}_r)$;
  return t^*;
end.

Procedure $TopDown(v$: vertex of N, $t_v \in r_v)$;
begin
  for each $c \in N$ s.t. $(v, c) \in E$ **do**
    $\bar{t}_c := t_{t_v, c}$;
    $t^* := t^* \cup \bar{t}_c$;   (* include solution *)
    $TopDown(c, \bar{t}_c)$;
  end for
end;

Fig. 3. Algorithm COMPUTEOPTIMALSOLUTION

4.1   Constraint Satisfaction Optimization Problems

An instance of a *constraint satisfaction optimization problem* (CSOP) consists of a pair $\langle \mathcal{I}, w \rangle$, where $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$ is a CSP instance and where $w : Var \times U \mapsto \mathbb{Q}$ is a function mapping substitutions for individual variables to rational numbers. For a substitution $\{X_1/u_1, ..., X_n/u_n\}$, we denote by $w(\{X_1/u_1, ..., X_n/u_n\})$ the value $\sum_{i=1}^{n} w(X_i, u_i)$. Then, a solution to a CSOP instance $\langle \mathcal{I}, w \rangle$ is a solution θ to $\mathcal{I}$ such that $w(\theta) \leq w(\theta')$, for each solution θ' to $\mathcal{I}$. Details on this framework can be found, e.g., in [32].

Constraint satisfaction optimization problems naturally arise in various application contexts. As an example they have recently been used in the context of combinatorial auctions [13], in order to model and solve the *winner determination problem* of determining the allocation of the items among the bidders that maximizes the sum of the accepted bid prices. In particular, in [13], it has been observed that CSOPs and, in particular, the winner determination problem, can be solved in polynomial time on some classes of acyclic instances via a dynamic programming algorithm founded on the ideas of [34]. This algorithm, named COMPUTEOPTIMALSOLUTION, is reported in Figure 3 and will be briefly illustrated in the following.

The algorithm receives in input the instance $\langle \mathcal{I}, w \rangle$ and a join tree $T = (N, E)$ for $\mathcal{H}(\mathcal{I})$. Recall that each vertex $v \in N$ corresponds to a hyperedge of $\mathcal{H}(\mathcal{I})$ and, in its turn, to a constraint in $\mathcal{C}$; hence, we shall simply denote by (S_v, r_v) the constraint in $\mathcal{C}$ univocally associated with vertex v.

Based on $\langle \mathcal{I}, w \rangle$ and T, COMPUTEOPTIMALSOLUTION computes an optimal solution (or checks that there is no solution) by looking for the "conformance" of the tuples in each relation r_v with the tuples in r_c, for each child c of v in T, where $t_v \in r_v$ is said to *conform* with $t_c \in r_c$, denoted by $t_v \approx t_c$, if for each $X \in S_v \cap S_c$, $X/u \in t_v \Leftrightarrow X/u \in t_c$. In more detail, COMPUTEOPTIMALSOLUTION solves $\langle \mathcal{I}, w \rangle$ by traversing T in two phases. First, vertices of T are processed from the leaves to the root r, by means of the procedure $BottomUp$ that updates the weight $\ell^v_{t_v}$ of the current vertex v. Intuitively, $\ell^v_{t_v}$ stores the cost of the best partial solution for $\mathcal{I}$ computed by using only the variables occurring in the subtree rooted at v. Indeed, if v is a leaf, then $\ell^v_{t_v} = w(t_v)$. Otherwise, for each child c of v in T, $\ell^v_{t_v}$ is updated by adding the minimum value $\ell^c_{t_c} - w(t_c \cap t_v)$ over all tuples t_c conforming with t_v. The tuple $\bar{t}_c$ for which this minimum is achieved is stored in the variable $t_{t_v, c}$ (resolving ties arbitrarily). Note that if this process cannot be completed, because there is no tuple in r_v conforming with some tuple in each relation associated with the children of v, then we may conclude that $\mathcal{I}$ is not does not admit any solution. Otherwise, after the root $r \in N$ is reached, this part ends, and the top-down phase may start.

In this second phase, the tree T is processed starting from the root. Firstly, the assignment t^* is defined as the tuple in r_r with the minimum cost over all the tuples in r_r (again, resolving ties arbitrarily). Then, procedure $TopDown$ extends t^* with a tuple for each vertex of T: at each vertex v and for each child c of v, t^* is extended with the tuple $t_{t_v, c}$ resulting from the bottom-up phase.

Being based on a standard dynamic programming scheme, correctness of COMPUTEOPTIMALSOLUTION can be shown by structural induction on the subtrees of T [13]. Moreover, by analyzing its running time, one may note that dealing with cost functions does not (asymptotically) provide any overhead w.r.t. Yannakakis's algorithm [34] for plain CSPs. Following [13], the following can be shown for the more general case of CSOP instances having bounded generalized hypertree-width hypergraphs.[1]

Theorem 1. *Let $\langle \mathcal{I}, w \rangle$ be a CSOP instance and HD a (generalized) hypertree decomposition of $\mathcal{H}(\mathcal{I})$. Moreover, let k be the width of HD and m be the number of vertices in its decomposition tree. Then, a solution to $\langle \mathcal{I}, w \rangle$ can be computed (or it is discovered that no solution exists) in time $O((m-1) \times r^k_{max} \times \log r^k_{max})$, where r_{max} is the size of the largest constraint relation in $\mathcal{I}$.*

---

[1] In all complexity results, we assume the weighting function w be explicitly listed in the input (otherwise, just add the cost of computing through w all cost values for the variable assignments of the given input instance).

4.2   Weighted CSPs: Costs over Tuples

Let us now turn to study a slight variation of the above scenario, where costs are associated with each *tuple* of the constraint relations, rather than with substitutions for individual variables. In fact, this is the setting of weighted CSPs, a well-known specialization of the more general *valued* CSP framework [30].

Formally, a *weighted* CSP (WCSP) instance consists of a tuple $\langle \mathcal{I}, w_1, ..., w_q \rangle$, where $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$ with $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a CSP instance, and where, for each tuple $t_v \in r_v$, $w_v(t_v) \in \mathbb{Q}$ denotes the cost associated with t_v. For a solution $\theta = t_1 \cup ... \cup t_q$ to $\mathcal{I}$, we define $w(\theta) = \sum_{v=1}^{q} w_v(t_v)$ as its associated cost. Then, a solution to $\langle \mathcal{I}, w_1, ..., w_q \rangle$ is a solution θ to $\mathcal{I}$ such that $w(\theta) \leq w(\theta')$, for each solution θ' to $\mathcal{I}$.

A few tractability results for WCSPs (actually, for valued CSPs) are known in the literature when structural restrictions are considered over binary encodings of the constraint hypergraphs. Indeed, it has been observed that WCSPs are tractable when restricted on classes of instances whose associated primal graphs are acyclic or nearly-acyclic (see, e.g., [33,10,26]). However, the primal graph obscures much of the structure of the underlying hypergraph since, for instance, each hyperedge is turned into a clique there—see the discussion in Section 3.

Therefore, whenever constraints have large arities, tractability results for primal graphs are useless, and it becomes then natural to ask whether polynomial-time solvability still holds when moving from (nearly-)acyclic primal graphs to acyclic hypergraphs, possibly associated with very intricate primal graphs. Next, we shall positively answer this question, by simply recasting weighted CSPs as constraint optimization problems, and by subsequently solving them via the algorithm COMPUTEOPTIMALSOLUTION. To this end, given a WCSP instance $\langle \mathcal{I}, w_1, ..., w_q \rangle$, we define its associated CSOP instance, denoted by $\text{CSOP}(\langle \mathcal{I}, w_1, ..., w_q \rangle)$, as the pair $\langle \mathcal{I}', w' \rangle$ with $\mathcal{I}' = \langle Var', U', \mathcal{C}' \rangle$ such that:

- $Var' = Var \cup \{D_1, ..., D_q\}$, where each D_v is a fresh auxiliary variable in $\mathcal{I}'$;
- $U' = U \cup \bigcup_{v=1}^{q} \bigcup_{t_v \in r_v} \{u_{t_v}\}$, i.e., for each constraint $(S_v, r_v) \in \mathcal{C}$, U' contains a fresh value for each tuple in r_v—intuitively, mapping the variable D_v to u_{t_v} encodes that the tuple t_v is going to contribute to a solution for $\mathcal{I}$;
- $\mathcal{C}' = \{(S_v \cup \{D_v\}, r_v') \mid (S_v, r_v) \in \mathcal{C}\}$, where $r_v' = \{t_v \cup \{D_v/u_{t_v}\} \mid t_v \in r_v\}$;
- $w'(X/u) = w_v(t_v)$ if $X = D_v$ and $u = u_{t_v}$, for some tuple $t_v \in r_v$; otherwise, $w'(X/u) = 0$. That is, the whole cost of each tuple is determined by the mapping of its associated fresh variable D_v.

It is immediate to check that the above transformation is feasible in linear time. In addition, the transformation enjoys two relevant preservation properties: Firstly, it preserves the structural properties of the WCSP instance in that $\mathcal{H}(\mathcal{I}')$ is acyclic if and only if $\mathcal{H}(P)$ is acyclic; and secondly, it preserves its solutions, in that $\theta' = t_1' \cup ... \cup t_q'$ is a solution to $\langle \mathcal{I}, w_1, ..., w_q \rangle$ if and only if $\theta = t_1 \cup ... \cup t_q$ is a solution to $\langle \mathcal{I}', w' \rangle$, where $t_v' = t_v \cup \{D_v/u_{t_v}\}$ for each $1 \leq v \leq q$. By exploiting these observations and Theorem 1, the following can be established.

Theorem 2. *Let $\langle \mathcal{I}, w_1, ..., w_q \rangle$ be a WCSP instance and HD a (generalized) hypertree decomposition of $\mathcal{H}(\mathcal{I})$. Moreover, let k be the width of HD and m be the number of vertices in its decomposition tree. Then, a solution to $\langle \mathcal{I}, w_1, ..., w_q \rangle$ can be computed (or we may state that there is no solution) in time $O((m-1) \times r_{max}^k \times \log r_{max}^k)$, where r_{max} is the size of the largest constraint relation in $\mathcal{I}$.*

5   Minimizing the Number of Violated Constraints

In this section, we shall complete our picture by considering those scenarios where problems might possibly be overconstrained and where, hence, the focus is on finding assignment minimizing the total number of violated constraints. These kinds of problems are usually referred to in the literature as Max-CSPs [12], which similarly as WCSPs are specializations of valued CSPs.

Formally, let $\theta : Var \mapsto U$ be an assignment for a CSP instance $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$. We say that the violation degree of θ, denoted by $\delta(\theta)$, is the number of relations r_v such that there is no tuple $t_v \in r_v$ with $t_v \subseteq \theta$. An assignment $\theta : Var \mapsto U$ is a solution to the Max-CSP instance (associated with $\mathcal{I}$) if $\delta(\theta) \leq \delta(\theta')$, for each assignment $\theta' : Var \mapsto U$. Note that Max-CSPs instances, by definition, do always have a solution.

5.1   Acyclic Instances Remain Intractable

After the tractability results established in Section 4.2 for WCSPs, one may expect good news for Max-CSPs, too. Surprisingly, this is not the case.

Theorem 3. *Solving Max-CSPs is NP-hard, even when restricted over classes of instances with acyclic constraint hypergraphs.*

Proof. Consider any class $\mathcal{T}$ of CSPs instances having an NP-hard satisfiability problem. Then, let $\mathcal{T}'$ be a new class of Max-CSP instances such that, for each $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle \in \mathcal{T}$, $\mathcal{T}'$ contains an instance $\mathcal{I}' = \langle Var, U, \mathcal{C}' \rangle$ with $\mathcal{C}' = \mathcal{C} \cup \{(Var, \emptyset)\}$. That is, any instance $\mathcal{I}' \in \mathcal{T}'$ has a constraint over all variables with an empty constraint relation, and thus it is not satisfiable. Moreover, because of the big hyperedge associated with such a constraint, its hypergraph $\mathcal{H}(\mathcal{I}')$ is trivially acyclic. Also, by construction, there is an assignment for $\mathcal{I}'$ violating only one constraint if and only if $\mathcal{I}$ is satisfiable. It follows that finding an assignment minimizing the total number of violated constraints is NP-hard on the class of acyclic instances $\mathcal{T}'$. $\square$

5.2   Incidence Graphs and Tractable Cases

Given that hypergraph acyclicity and hence its generalizations are not sufficient for guaranteeing the tractability of Max-CSPs, it makes sense to explore acyclicity properties related to suitable graph representations. In fact, as observed in Section 4.2, it is well-known that valued CSPs (and, hence, Max-CSPs) are tractable over acyclic primal graphs (e.g., [33,10,26]). More precisely, tractability

has been observed in the literature to hold over primal graphs having bounded *treewidth* (see Section 3). Our main result in this section is precisely to show that tractability still holds in case the *incidence graph* of $\mathcal{H}(\mathcal{I})$ has bounded treewidth, which is a more general condition than the bounded treewidth of primal graphs and which can be used to establish better complexity bounds and to enlarge the class of tractable instances [22]. The fact that the standard CSP is tractable for instances whose incidence graphs have bounded treewidth was already shown in [6]. We here extend this tractability result to Max-CSPs.

Recall that the incidence encoding of a hypergraph $\mathcal{H}$, denoted by $inc(\mathcal{H}) = (N, E)$, is the bipartite graph where $N = \mathcal{E}(\mathcal{H}) \cup \mathcal{N}(\mathcal{H})$ and $E = \{ \{h, a\} \mid h \in \mathcal{E}(\mathcal{H})$ and $a \in h)\}$, i.e. it contains an edge between h and a if and only if the variable a occurs in the hyperedge h. As an example, Figure 2 reports on the rightmost part the incidence graph $inc(\mathcal{H}_1)$, where nodes associated with hyperedges in $\mathcal{E}(\mathcal{H}_1)$ are depicted as black circles. Note that the treewidth of $inc(\mathcal{H}_1)$ is 2, which is much smaller than the treewidth of $G(\mathcal{H}_1)$. This does not happen by chance since, for each hypergraph $\mathcal{H}$, it holds that $tw(inc(\mathcal{H})) \leq tw(G(\mathcal{H}))$; in addition, there are also classes of hypergraphs with incidence encodings of bounded treewidth and primal encodings of unbounded treewdith (see, e.g., [22]).

While enlarging the class of instances having bounded treewidth, the incidence encoding still conveys all the information needed to solve Max-CSP instances. Again, the solution algorithm consists of a transformation into a suitable CSOP instance. Formally, let $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$ be a Max-CSP instance with $\mathcal{C} = \{(S_1, r_1), ..., (S_q, r_q)\}$, and let $\langle T, \chi \rangle$ be a k-width tree decomposition of $inc(\mathcal{H}(\mathcal{I}))$—recall that for each vertex $v \in T$, $\chi(v)$ is a set of variables (i.e., nodes of $\mathcal{N}(\mathcal{H}(\mathcal{I}))$) and constraint scopes (i..e, edges in $\mathcal{E}(\mathcal{H}(\mathcal{I}))$). Then, the constraint satisfaction optimization problem instance $\text{CSOP}(\mathcal{I}, \langle T, \chi \rangle)$ is the pair $\langle \mathcal{I}', w' \rangle$, where $\mathcal{I}' = \langle Var', U', \mathcal{C}' \rangle$ and such that:

- $Var' = Var \cup \{S_1, ..., S_q\}$, that is, also the constraint scopes of $\mathcal{C}$ belong to the variables of the new problem;
- $U' = U \cup \{unsat\} \cup \{u_t \mid t \in r_i, \text{ for } 1 \leq i \leq q\}$;
- $\mathcal{C}' = \{(\chi(v), r'_v) \mid v \in T\}$ where the constraint relation r'_v is defined as follows. Let $\mu = |\chi(v) \cap Var|$, and let U^μ denote the set of all possible tuples over the μ variables in $\chi(v) \cap Var$. Let also $S_{i_1}, ... S_{i_h}$ be the scope-variables in $\chi(v)$. Then, for each tuple $\theta \in U^\mu$, the relation r'_v contains all tuples $\theta \cup \{S_{i_1}/v_{i_1}\} \cup \cdots \cup \{S_{i_h}/v_{i_h}\}$, where $v_{i_j} \in U$ $(1 \leq j \leq h)$ is a value for the scope-variable S_{i_j} such that: $v_{i_j} = u_t$ if there is a tuple $t \in r_{i_j}$ conforming with θ; and $v_{i_j} = unsat$, if no such a tuple exists in r_{i_j}.
- $w'(X/u) = 0$ if $u \neq unsat$; otherwise $w'(X/u) = 1$, that is, each constraint of $\mathcal{C}$ that is not satisfied increases the cost of a solution by a unitary factor.

Note that this transformation is feasible in time exponential in the width of $\langle T, \chi \rangle$ only. Moreover, solutions of $\mathcal{I}'$ with minimum total cost precisely correspond to assignments over $\mathcal{I}$ minimizing the total number of violated constraints. In fact, the following can be established.

Theorem 4. *Let* $\mathcal{I} = \langle Var, U, \mathcal{C} \rangle$ *be a* Max-CSP *instance with* $tw(inc(\mathcal{H}(\mathcal{I}))) = k$. *Then, a solution to* $\mathcal{I}$ *can be computed in time* $O(|Var| \times |U|^{k+1} \times \log |U|^{k+1})$.

6 Conclusion and Discussion

In this paper, classes of tractable CSOP, WCSP, and Max-CSP instances are singled out by overviewing and proposing solution approaches applicable to instances whose hypergraphs have bounded (generalized) hypertree width, or whose incidence graphs have bounded treewidth. The techniques described in this paper are mainly based on Algorithm COMPUTEOPTIMALSOLUTION, which has been designed to optimize costs expressed as rational numbers and combined via the summation operation. However, it is easily seen that it remains correct if costs are specified over an arbitrary totally ordered monoid structure, where some binary operation $\oplus$ (in place of standard summation) is used in order to combine costs, provided it is commutative, associative, closed, and that it verifies identity and monotonicity. It follows that all tractable classes of CSOP, WCSP, and Max-CSP instances identified in this paper remain tractable in such extended scenarios, which indeed emerge with valued CSPs (see, e.g., [4]).

References

1. Adler, I., Gottlob, G., Grohe, M.: Hypertree-Width and Related Hypergraph Invariants. European Journal of Combinatorics 28, 2167–2181 (2007)
2. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. Journal of the ACM 30(3), 479–513 (1983)
3. Bernstein, P.A., Goodman, N.: The power of natural semijoins. SIAM Journal on Computing 10(4), 751–771 (1981)
4. Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., Fargier, H.: Semiring-Based CSPs and Valued CSPs: Frameworks, Properties,and Comparison. Constraints 4(3), 199–240 (1999)
5. Bodlaender, H.L., Fomin, F.V.: A Linear-Time Algorithm for Finding Tree Decompositions of Small Treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)
6. Chekuri, C., Rajaraman, A.: Conjunctive Query Containment Revisited. MFPS 1985 239(2), 211–229 (2000); Preliminary version in: Schwentick, T., Suciu, D. (eds.): ICDT 2007. LNCS, vol. 4353, pp. 211–229. Springer, Heidelberg (2007); Afrati, F.N., Kolaitis, P.G. (eds.): ICDT 1997. LNCS, vol. 1186, pp. 56–70. Springer, Heidelberg (1996) (Full version)
7. Cohen, D.A., Jeavons, P.G., Gyssens, M.: A unified theory of structural tractability for constraint satisfaction problems. Journal of Computer and System Sciences 74(5), 721–743 (2008)
8. Daskalakis, C., Papadimitriou, C.H.: Computing pure nash equilibria in graphical games via markov random fields. In: Proc. of ACM EC 2006, pp. 91–99 (2006)
9. Fagin, R.: Degrees of acyclicity for hypergraphs and relational database schemes. J. ACM 30(3), 514–550 (1983)
10. de Givry, S., Schiex, T., Verfaillie, G.: Exploiting Tree Decomposition and Soft Local Consistency In Weighted CSP. In: Proc. of AAAI 2006 (2006)
11. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
12. Freuder, E.C., Wallace, R.J.: Partial Constraint Satisfaction. Artificial Intelligence 58(1-3), 21–70 (1992)

13. Gottlob, G., Greco, G.: On the complexity of combinatorial auctions: structured item graphs and hypertree decomposition. In: Proc. EC 2007, pp. 152–161 (2007) (full version currently available as Technical Report, University of Calabria)

14. Gottlob, G., Greco, G., Scarcello, F.: Pure Nash Equilibria: Hard and Easy Games. Journal of Artificial Intelligence Research 24, 357–406 (2005)

15. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence 124(2), 243–282 (2000)

16. Gottlob, G., Leone, N., Scarcello, F.: The complexity of acyclic conjunctive queries. Journal of the ACM 48(3), 431–498 (2001)

17. Gottlob, G., Leone, N., Scarcello, F.: Advanced parallel algorithms for processing acyclic conjunctive queries, rules, and constraints. In: Proc. of SEKE 2000, pp. 167–176 (2000)

18. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. J. of Computer and System Sciences 64(3), 579–627 (2002)

19. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. J. of Computer and System Sciences 66(4), 775–808 (2003)

20. Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: NP-hardness and tractable variants. In: Proc. of PODS 2007, pp. 13–22 (2007)

21. Gottlob, G., Pichler, R., Wei, F.: Bounded Treewidth as a Key to Tractability of Knowledge Representation and Reasoning. In: Proc. of AAAI 2006 (2006)

22. Greco, G., Scarcello, F.: Non-Binary Constraints and Optimal Dual-Graph Representations. In: Proc. of IJCAI 2003, pp. 227–232 (2003)

23. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: Proc. of SODA 2006, Miami, Florida, USA, pp. 289–298 (2006)

24. Kask, K., Dechter, R., Larrosa, J., Dechter, A.: Unifying tree decompositions for reasoning in graphical models. Artificial Intelligence 166(1-2), 165–193 (2005)

25. Meseguer, P., Rossi, F., Schiex, T.: Soft Constraints. In: Handbook of Constraint Programming. Elsevier, Amsterdam (2006)

26. Ndiaye, S., Jégou, P., Terrioux, C.: Extending to Soft and Preference Constraints a Framework for Solving Efficiently Structured Problems. In: Proc. of ICTAI 2008, pp. 299–306 (2008)

27. Reingold, O.: Undirected ST-connectivity in log-space. Journal of the ACM 55(4) (2008)

28. Robertson, N., Seymour, P.D.: Graph minors III: Planar tree-width. Journal of Combinatorial Theory, Series B 36, 49–64 (1984)

29. Scarcello, F., Gottlob, G., Greco, G.: Uniform Constraint Satisfaction Problems and Database Theory. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 156–195. Springer, Heidelberg (2008)

30. Schiex, T., Fargier, H., Verfaillie, G.: Valued Constraint Satisfaction Problems: Hard and Easy Problems. In: Proc. of IJCAI 1995, pp. 631–639 (1995)

31. Tarjan, R.E., Yannakakis, M.: 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)

32. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press, London (1993)

33. Terrioux, C., J'egou, P.: Bounded Backtracking for the Valued Constraint Satisfaction Problems. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 709–723. Springer, Heidelberg (2003)

34. Yannakakis, M.: Algorithms for acyclic database schemes. In: Proc. of VLDB 1981, pp. 82–94 (1981)