HyperConsistency Width for Constraint Satisfaction: Algorithms and Complexity Results

Georg Gottlob[1], Gianluigi Greco[2], and Bruno Marnette[1]

[1] Oxford University, OX1 3QD Oxford, UK
{georg.gottlob,bruno.marnette}@comlab.ox.ac.uk
[2] University of Calabria, Via P.Bucci 30B, 87036, Rende, Italy
ggreco@mat.unical.it

Abstract. Generalized hypertree width (short: ghw) is a concept that leads to a large class of efficiently solvable CSP instances, whose associated recognition problem (of checking whether the ghw of a CSP is bounded by a constant k) is however known to be NP-hard. An elegant way to circumvent this intractability has recently been proposed in the literature, by means of a "no-promise" approach solving CSPs of bounded ghw without the need of actually computing a generalized hypertree decomposition. In fact, despite the conceptual relevance of this approach, its computational issues have not yet been investigated and, indeed, precise bounds on the running time of the no-promise algorithm are missing.

The first contribution of this paper is precisely to fill this gap. Indeed, the computational complexity of the no-promise approach is analyzed, by exploiting an intuitive characterization relying on the notion of *hyperconsistency width*. It turns out that, in the basic formulation, the approach is hardly suited for practical applications mainly because of its bad scaling in the size of the constraint database. Motivated by these news and based on a variant of hyperconsistency width, a different and more efficient method to decide whether CSPs of bounded ghw admit solutions is then provided. Importantly, the improved method exhibits the same scaling as current evaluation algorithms for instances of bounded hypertree width, nonetheless allowing to isolate a larger class of queries. Finally, to give a complete picture of the complexity issues of the no-promise approach, the problems of computing one solution and of enumerating all the solutions are also studied.

1 Introduction

The Constraint Satisfaction Problem (CSP) is a well-known framework to model and solve search problems in several application domains. Formally, a CSP instance (e.g., [5]) is a triple $(Var, U, \mathcal{C})$, 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, where each C_i is a pair (S_i, r_i), in which $S_i \subseteq Var$ is called the *constraint scope*, and $r_i \subseteq U^{|S_i|}$ is called the *constraint relation*. Then, a solution for a CSP instance is simply a substitution $\theta : Var \mapsto U$ that satisfies all constraints.

Following [13], in this paper we shall exploit the logic-based characterization of a CSP instance as a pair $(Q, \mathcal{D})$, where $\mathcal{D}$ is the *constraint database*, i.e., the set of all the constraint relations r_i, for each constraint $C_i = (S_i, r_i)$, and Q is a *conjunctive query*, i.e., an existentially quantified first-order formula with no negations or disjunctions,

M. Lipshteyn et al. (Eds.): Golumbic Festschrift, LNCS 5420, pp. 87–99, 2009.

over the relational vocabulary consisting of the atoms $r_i(S_i)$. Then, a solution θ is a substitution such that $\theta(Q)$ evaluates true over $\mathcal{D}$. The set of all the solutions for $(Q, \mathcal{D})$ will be denoted by $Q[\mathcal{D}]$.

Since solving CSPs in their general formulation is an NP-hard problem, much research has been spent to identify restricted classes of CSPs over which solutions can efficiently be computed. In this paper, we shall focus on classes of CSPs defined by *structural* restrictions on the queries (see, e.g., [15,5,11,7]). In this context, we recall that a class $\mathcal{C}$ is generally considered an "island of tractability" for CSPs if both the *recognition* problem of deciding whether Q is in $\mathcal{C}$, and the following *promise* problem

$$\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}] : \begin{cases} \text{Input} & : \text{a CSP instance } (Q, \mathcal{D}) \text{ s.t. } Q \in \mathcal{C} \\ \text{Output} & : \text{Yes iff } Q[\mathcal{D}] \neq \emptyset, \\ & \quad \text{No iff } Q[\mathcal{D}] = \emptyset \end{cases}$$

are feasible in polynomial time.

When the recognition problem is hard, the *promise* problem $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$ might still be of interest in some applications, since one may have a guarantee that Q belongs to the class $\mathcal{C}$. But, this is in general not the case and, hence, one is more likely interested in solving the *no-promise* problem $\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$, where an arbitrary CSP instance $(Q, \mathcal{D})$ is given in input, and where an algorithm may possibly decline to answer (e.g., answering $\mathsf{IDon't Know}$) when $Q \notin \mathcal{C}$:

$$\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}] : \begin{cases} \text{Input} & : \text{a CSP instance } (Q, \mathcal{D}) \\ \text{Output} & : \text{Yes only if } Q[\mathcal{D}] \neq \emptyset, \\ & \quad \text{No only if } Q[\mathcal{D}] = \emptyset, \\ & \quad \mathsf{IDon't Know} \text{ only if } Q \notin \mathcal{C} \end{cases}$$

The need of dealing with no-promise problems has recently been argued in [3], where the class of queries whose *generalized hypertree width* [8] is bounded by a constant k (short: class $\mathcal{C}(\mathsf{ghw}, k)$) has been considered, and where it is shown that $\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}(\mathsf{ghw}, k)]$ can be solved in polynomial time, by means of a *projective k-consistency* algorithm. Even though this result is conceptually relevant because checking whether the ghw of a CSP is bounded by a constant k is NP-hard [9], its practical applicability is still unclear because a thorough analysis of the complexity issues of the projective k-consistency algorithm has not been conducted and, in fact, precise bounds on its running time were missing.

In this paper we continue along this line of research, and we face the above research questions by shedding light on the computational aspects of the projective k-consistency algorithm, which have not been discussed in [3]. Indeed:

(1) We introduce the notion of *hyperconsistency width* as a measure for characterizing the intricacy of constraints, and we study the computational properties of the class $\mathcal{C}(\mathsf{hcw}, k)$ of those queries whose hyperconsistency width is bounded by k.
(2) We show that this notion is a different, yet equivalent reformulation of the *projective width*, implicit in [3]. In fact, our reformulation leads to a conceptually simpler method whose correctness is proven without the need of any game-theoretic characterization for it.

	$\mathsf{hcw} \leq \mathsf{ghw}$	$\mathsf{ahcw} \leq \mathsf{ghw}$
$\mathsf{SAT}^{\mathsf{p}}_{\emptyset}$	$O(\mathbf{q}^{3k}\mathbf{r}^{2k})$	$O(\mathbf{q}^{2k+1}\mathbf{r}^{k})$
$\mathsf{SAT}^{\mathsf{p}}_{\exists}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1})$
$\mathsf{SAT}^{\mathsf{p}}_{\forall}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1}\mathbf{o})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1}\mathbf{o})$
$\mathsf{SAT}^{\mathsf{np}}_{\emptyset}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1})$
$\mathsf{SAT}^{\mathsf{np}}_{\exists}$	$O(\mathbf{q}^{3k+1}\mathbf{r}^{2k+1})$	$O(\mathbf{q}^{2k+2}\mathbf{r}^{k+1})$
$\mathsf{SAT}^{\mathsf{np}}_{\forall}$	$O(\mathbf{q}^{3k+2}\mathbf{r}^{2k+2}\mathbf{o})$	$O(\mathbf{q}^{2k+3}\mathbf{r}^{k+2}\mathbf{o})$

Fig. 1. Results for $\mathsf{SAT}[C(\mathsf{ghw}, k)]$. On the left: bounds for the algorithm in [3]; on the right: improved bounds.

(3) We introduce and investigate a different method to solve problems in $\mathcal{C}(\mathsf{ghw}, k)$, whose complexity improves on the bounds derived in **(1)**. In particular, the basic version of the projective k-consistency algorithm scales as $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$, where $\mathbf{q}$ is the size of Q (i.e. the number of atoms plus the number of variables in the formula), and $\mathbf{r}$ is the size of the largest relation in $\mathcal{D}$ (i.e. the number of its entries). To the contrary, our novel method scales as $O(\mathbf{q}^{2k+1}\mathbf{r}^{k})$, thereby significantly enlarging the class of instances that can practically be managed. Indeed, this scaling is basically the same that one may achieve when solving CSPs on the class of queries having bounded *hypertree width* [8] (short: $\mathcal{C}(\mathsf{hw}, k)$), even though this class is properly contained in $\mathcal{C}(\mathsf{ghw}, k)$ and is, in fact, one of the largest classes of tractable CSPs that is known to be also efficiently recognizable (see, e.g., [2,7]).

(4) Technically, the result in **(3)** is achieved by exploiting the fact that the hyperconsistency width is defined in a way that is parametric w.r.t. the underlying decompositions. This is the main conceptual difference with the projective width. In fact, our improved method is based on restricting the consistency algorithm on the class of *acyclic* decompositions whose width is bounded by k (short: the subclass $\mathcal{C}(\mathsf{ahcw}, k)$ of $\mathcal{C}(\mathsf{hcw}, k)$). The relationships among these two notions and the notion of generalized hypertree width are also clarified in the paper.

(5) Finally, to complement the results on the decision problems related to the class $\mathcal{C}(\mathsf{ghw}, k)$, we investigate the complexity issues arising for the problems of computing *one* solution (short: $\mathsf{SAT}_{\exists}$) and of computing *all* the solutions (short: $\mathsf{SAT}_{\forall}$).

Our results are summarized in Figure 1, where $\mathbf{o}$ denotes the size of the output $Q[\mathcal{D}]$. Notice that the complexity of the problems SAT, $\mathsf{SAT}_{\exists}$, and $\mathsf{SAT}_{\forall}$ has completely been characterized for both the cases of promise and no-promise problems, and for both the approach investigated in [3] and for the improved methods based on the notion of acyclic hyperconsistency width.

The rest of the paper is organized as follows. In Section 2, the basic notion of hyperconsistency width is presented and its links with the notion of projective width are discussed. Then, in Section 3, the subclass $\mathcal{C}(\mathsf{ahcw}, k)$ of $\mathcal{C}(\mathsf{hcw}, k)$ is defined and its computational properties are studied, by also comparing them with the bounds derived

for the approach in [3]. The problems of computing one solution and of computing all the solutions are discussed in Section 4. Eventually, a few final remarks and some directions for further work are reported in Section 5.

2 HyperConsistency Width

In this section, we present the notion of hyperconsistency width, which can be viewed as a natural generalization of the various consistency concepts that have been studied in the context of binary representations of constraints (e.g., [4,14]). Then, we compare it with the notion of *projective width* that is implicit in [3].

The first step is to introduce the concept of *hypergraph decomposition*. In fact, the reader might already be familiar with the notion of *(generalized) hypertree decomposition* [8], which is basically a tree whose vertices are associated with a set of atoms and with a subset of the variables occurring in them, and where for each variable X, the subgraph induced over the vertices containing X is connected (see Section 3.1). Then, hypergraph decompositions can be viewed as an extension of hypertree decompositions where the underlying shape is an arbitrary graph rather than a tree, and where the connectedness condition is omitted. In the following, let us denote by $\mathsf{var}(Q)$ (resp., $\mathsf{atoms}(Q)$) the set of variables (resp., atoms) in a query Q.

Definition 1. A *hypergraph decomposition* J of a query Q, is a tuple $(\mathcal{G}, \chi, \lambda)$ such that:

- $\mathcal{G} = (V(J), E(J))$ is a directed graph;
- $\lambda : V(J) \rightarrow 2^{\mathsf{atoms}(Q)}$ associates to each vertex $v \in V(J)$ a set of atoms $\lambda(v) \subseteq \mathsf{atoms}(Q)$;
- $\chi : V(J) \rightarrow 2^{\mathsf{var}(Q)}$ associates to each vertex $v \in V(J)$ a set of variables $\chi(v) \neq \emptyset$ satisfying $\chi(v) \subseteq \bigcup_{A \in \lambda(v)} \mathsf{var}(A)$.

The *width* of J is defined as $\max_{v \in V(J)} |\lambda(v)|$. □

Next, we shall make use of some standard *relational operators* to manipulate constraint relations (see, e.g., [1]). Thus, if r_i is a relation over the scope S_i, and X is a subset of S_i, we denote by $\Pi_X(r_i)$ the relation obtained by projecting r_i over X. Also, given two relations r_i and r_j, the *join* of r_i and r_j is denoted by $r_i \bowtie r_j$; and, the *semi-join* of r_i and r_j, denoted by $r_i \ltimes r_j$, is just a shortcut for $r_i \bowtie (\Pi_{S_i \cap S_j} r_i)$.

Let $\mathcal{G} = (V, E)$ be a directed graph. A $\mathcal{G}$-set of relations $\mathcal{R}$ is a set of relations in one-to-one correspondence with the nodes V; thus, for each $v \in V$, $r_v \in \mathcal{R}$ is the associated relation in $\mathcal{R}$. The $\ltimes_{\mathcal{G}}$*-fixed-point* of $\mathcal{R}$, denoted by $\gamma_{\mathcal{G}}(\mathcal{R})$, is the $\mathcal{G}$-set of relations obtained as the unique fixed-point of $\mathcal{R}$ for the set of rules $\{r_v := r_v \ltimes r_{v'} \mid (v, v') \in E\}$.

Based on these notions, we can define an extremely simple and clear algorithm $\mathcal{A}_J$ (for a decomposition J) deciding whether $Q[\mathcal{D}] = \emptyset$. This algorithm, reported in Figure 2, firstly computes for each node v, the joins and the projections corresponding to $\lambda(v)$ and $\chi(v)$, respectively. Then, it computes the $\ltimes_{\mathcal{G}}$-fixed-point of the resulting $\mathcal{G}$-set of relations. Finally, it reports No (resp., IDon'tKnow) if some relation in this $\ltimes_{\mathcal{G}}$-fixed-point is (resp., is not) empty.

Input: a CSP instance $(Q, \mathcal{D})$, and a decomposition J;
Output: No or IDon'tKnow;

begin
  for each vertex $v \in V(\mathcal{G})$ **do**
    let $r_v = \Pi_{\chi(v)}(\ltimes_{A \in \lambda(v)} (A[D]))$;
  compute $\{r'_v \mid v \in V(\mathcal{G})\} = \gamma_{\mathcal{G}}(\{r_v \mid v \in V(\mathcal{G})\})$;
  if there is $v \in V(\mathcal{G})$ such that $r'_v = \emptyset$, **then return** No;
  else return IDon'tKnow;
end.

Fig. 2. Algorithm $\mathcal{A}_J$

Actually, $\mathcal{A}_J$ may be incomplete, but one may easily see that it is sound. In addition, we next show that it terminates in polynomial time and that, for some special classes of decomposition, it leads to a very efficient procedure.

Proposition 1. *For a k-width hypergraph decomposition J, algorithm $\mathcal{A}_J$ returns* No *on input $(Q, \mathcal{D})$ only if $Q[\mathcal{D}] = \emptyset$. Moreover, the $\ltimes_{\mathcal{G}}$-fixed-point can be computed so that $\mathcal{A}_J$ terminates in time: $O(|V| \times |E| \times \mathbf{r}^{2k})$, in the general case, and $O((|V|+|E|) \times \mathbf{r}^k)$ when $\mathcal{G}$ is acyclic.*

Proof. If some relation r'_v in the $\ltimes_{\mathcal{G}}$-fixed-point is empty, then any substitution θ for the variables in $\chi(v)$ can not be extended to a solution, i.e., to a substitution for all the variables in the query satisfying the constraints.

As for the running time, on a RAM machine, the join of k relations can be computed in time $O(\mathbf{r}^k)$, the projection of one relation in linear time, and the semijoin of two relations in linear time [6]. Therefore, the set $\{r_v \mid v \in V(\mathcal{G})\}$ can be computed in time $O(|V| \times \mathbf{r}^k)$, and each semijoin required for computing $\gamma_{\mathcal{G}}(\{r_v \mid v \in V(\mathcal{G})\})$ takes $O(\mathbf{r}^k)$. When $\mathcal{G}$ is acyclic, its edges can be sorted according to a topological order, so that only $|E|$ steps are needed to reach the fixed point. In the general case, instead, the result follows since $|V| \times \mathbf{r}^k$ steps are required at most to reach the fixed point, because this is an upper bound on the number of tuples in $\{r_v \mid v \in V(\mathcal{G})\}$, and since each step involves $|E|$ semijoin computations. □

Depending on the structure of Q and J, $\mathcal{A}_J$ might be a complete decision procedure. Formally, we say that a hypergraph decomposition J of a query Q is *consistent* if, for each database $\mathcal{D}$,

$$\mathcal{A}_J \text{ returns } \begin{cases} \mathsf{No} & \text{iff } Q[\mathcal{D}] = \emptyset \\ \mathsf{IDon't Know} & \text{iff } Q[\mathcal{D}] \neq \emptyset \end{cases}$$

Then, the *hyperconsistency width* $\mathsf{hcw}(Q)$ of Q is the minimum width over all its consistent decompositions. The class of those queries whose hyperconsistency width is bounded by k is denoted by $\mathcal{C}(\mathsf{hcw}, k)$. Focusing on this class provides a guarantee for the tractability of CSPs. This is shown below, by providing a polynomial bound for solving the $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ problem.

Theorem 1. *Let $(Q, \mathcal{D})$ be a CSP instance, let* $\mathbf{q}$ *be the size of* Q *and* $\mathbf{r}$ *the size of the largest relation in* $\mathcal{D}$*. Then,* $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ *is feasible in* $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$.

Proof. Let $J = (\mathcal{G}, \chi, \lambda)$ and $J' = (\mathcal{G}', \chi', \lambda')$ be two hypergraph decompositions for the same query Q. We say that J is contained in J', denoted by $J \sqsubseteq J'$, if there is a morphism f from the nodes of $\mathcal{G}$ to the nodes of $\mathcal{G}'$ such that $f(\mathcal{G}')$ is a subgraph of $\mathcal{G}$ and, for each node v of $\mathcal{G}$: $\chi'(f(v)) \supseteq \chi(v)$, and $\lambda'(f(v))) \supseteq \lambda(f(v)$.

Armed with this notion, one may note that $J \sqsubseteq J' \wedge J$ is consistent $\Rightarrow$ J' is consistent. In addition, we observe that for any fixed natural number k, the hypergraph decomposition J_k where:

- $V(J_k) = \{v \subseteq \mathsf{atoms}(Q) \mid |\mathsf{atoms}(v)| \leq k)\}$;
- $E(J_k) = V(J_k) \times V(J_k)$;
- $\forall v \in V(J_k)$, $\lambda(v) = v$ and $\chi(v) = (\cup_{A \in v} \mathsf{var}(A))$.

is $\sqsubseteq$-maximal among the set of all the k-width hypergraph decompositions.

When $\mathsf{hcw}(Q) \leq k$, we know that J_k is consistent and we can decide $\mathsf{SAT}^{p}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ by using the algorithm $\mathcal{A}_{J_k}$. Then, we can apply Proposition 1, and the running time follows by observing that $|V(J_k)| \times |E(J_K)| = |V(J_k)|^3$ and that $|V(J_k)| = O(\mathbf{q}^k)$, by construction. □

2.1 Links with Projective Width

We are now in the position of showing that the notion of hyperconsistency width is a different formulation of the notion of projective width of [3]. The result is next proven by observing that this notion provides a simple and natural characterization of the k-cover game, introduced in [3] to define classes of queries with bounded projective width.

For a query Q and a database $\mathcal{D}$, the game $\mathsf{k\text{-}cover}(Q, \mathcal{D})$ can be described as follows. Two players, *spoiler* and *duplicator*, play one after the other. At each step i, the spoiler chooses a pair (X_i, L_i) such that $L_i \subseteq \mathsf{atoms}(Q)$, $|L_i| \leq k$ and $X_i \subseteq \bigcup_{A \in L_i} \mathsf{var}(A)$. Then, the duplicator chooses a homomorphism h_i from variables in X_i to constants in $\mathcal{D}$ such that $h_i(X_i) \in \Pi_{X_i}(\bowtie_{A \in L_i} A[\mathcal{D}])$. Moreover, from the step $i > 1$, the duplicator is also asked to choose h_i such that, for each variable $V \in X_{i-1} \cap X_i$, $h_i(V) = h_{i-1}(V)$. The spoiler wins if the duplicator cannot find the required homomorphism, and the duplicator wins if the play is infinite.

Theorem 2. *Let Q be a query. Then, the following statements are equivalent:*

- *for each database $\mathcal{D}$, $Q[\mathcal{D}] = \emptyset$ implies that the spoiler has a winning strategy in the* $\mathsf{k\text{-}cover}(Q, \mathcal{D})$ *game;*
- $\mathsf{hcw}(Q) \leq k$.

Proof. (Sketch) For each k-width hypergraph decomposition $J = (\mathcal{G}, \chi, \lambda)$ of Q, and for each database $\mathcal{D}$, we can naturally define the game $\mathsf{k\text{-}cover}_J(Q, D)$ as a restriction of $\mathsf{k\text{-}cover}(Q, D)$, where the spoiler is asked to choose tuples of the form $(X_i, L_i) = (\chi(v_i), \lambda(v_i))$ for some $v_i \in V(J)$, and to follow the edges of E, i.e., for each $i > 1$, $(X_{i-1}, L_{i-1}) = (\chi(v_{i-1}), \lambda(v_{i-1}))$ is such that $(v_{i-1}, v_i) \in E(J)$. Then, we can show that for each node $v \in V(J)$, the relation r'_v computed by the

algorithm $\mathcal{A}_J$ in Figure 2 exactly corresponds to a mapping witnessing a winning strategy for the duplicator in the game $\mathsf{k\text{-}cover}_J(Q, \mathcal{D})$. Therefore, a hypergraph decomposition J is consistent if and only if for each database $\mathcal{D}$ such that $Q[\mathcal{D}] = \emptyset$, the spoiler has a winning strategy in $\mathsf{k\text{-}cover}_J(Q, D)$. To conclude the proof, we claim that $\mathsf{k\text{-}cover}(Q, \mathcal{D})$ coincides with the game $\mathsf{k\text{-}cover}_{J_k}(Q, D)$, where J_k is the $\sqsubseteq$-maximal hypergraph decomposition constructed as in the proof of Theorem 1. □

Note that a similar link was established in [14] for binary representations of CSPs.

3 Acyclic HyperConsistency Width

According to Theorem 1, the promise problem for the class $\mathcal{C}(\mathsf{hcw}, k)$ can be solved in polynomial time in the size of the query and the database. However, despite this theoretical guarantee on the tractability, the running time $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$ seems not suited for practical applications, mainly because of the bad scaling in the size of the database. In particular, we point out that solving CSPs in the class $\mathcal{C}(\mathsf{hw}, k)$ (or, in $\mathcal{C}(\mathsf{ghw}, k)$, when a decomposition is given to hand) can be done by means of algorithms scaling as $O(\mathbf{r}^k)$ in the size of the database (see, e.g., [8]).

Hence, improving on the scaling we have derived for the notion of hyperconsistency width is a major requirement for making this approach practical and competitive w.r.t. structural methods already used in the literature. To achieve this goal, the basic idea we shall exploit is to introduce a refinement of the notion of hyperconsistency width, for which a good scaling is obtained by suitably restricting the shape of the underlying hypergraph decompositions to acyclic graphs (as suggested by the better scaling which can be achieved in this case according to Proposition 1).

Definition 2. The *acyclic hyperconsistency width* $\mathsf{hcw}(Q)$ of a query Q is the minimum width over all its consistent decompositions $J = (\mathcal{G}, \chi, \lambda)$ such that:

(1) J is acyclic; and,
(2) the *depth* of J (the length of the longest path in $\mathcal{G}$) is bounded by the size of Q. □

Clearly enough, this novel notion comes as a refinement of the hyperconsistency width and, therefore, it holds that $\mathcal{C}(\mathsf{ahcw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$. On the other hand, it allows us to improve the complexity result of Theorem 1, as stated below.

Theorem 3. $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{ahcw}, k)]$ *is feasible in* $O(\mathbf{q}^{2k+1}\mathbf{r}^k)$.

Proof. We observe that for any fixed natural number k, the following hypergraph decomposition $J_{k,n}$ where:

- $V(J_{k,n})$=$V \times \{0, 1, ..., n\}$, where $V = \{v \subseteq \mathsf{atoms}(Q) \mid \ |\mathsf{atoms}(v)| \leq k)\}$;
- $E(J_{k,n})$=$\{((v, i), (v', i')) \mid (v, v')\in V\times V \wedge\ i'=i + 1\}$
- $\forall v \in V(J_k)$, $\lambda(v) = v$ and $\chi(v) = (\cup_{A\in v}\mathsf{var}(A))$.

is $\sqsubseteq$-maximal among all the k-width decompositions of depth bounded by n. Therefore, when $\mathsf{ahcw}(Q) \leq k$, and $n = |Q|$ we know that $J_{k,n}$ is consistent and we can decide $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{hcw}, k)]$ by using the algorithm $\mathcal{A}_{J_{k,n}}$. The running time, then, follows from Proposition 1. □

3.1 Links with Generalized Hypertree Width

We next investigate on how the classes $\mathcal{C}(\mathsf{ahcw}, k)$ and $\mathcal{C}(\mathsf{hcw}, k)$ compare with $\mathcal{C}(\mathsf{ghw}, k)$. Let us start by preliminary recalling the definition of generalized hypertree width [8], by stating it as a specialization of the notion of hypergraph decomposition.

In fact, the focus is on acyclic decompositions (similarly to the case of ahcw), satisfying some additional requirements.

Definition 3. A *generalized hypertree decomposition* J of a query Q is a hypergraph decomposition $(\mathcal{G}, \chi, \lambda)$ such that:

- $\mathcal{G}$ is a rooted tree;
- $\forall A \in \mathsf{atoms}(Q)$, $\exists v \in V(G)$ such that $A \in \lambda(v)$;
- $\forall X \in \mathsf{var}(Q)$, the subgraph of $\mathcal{G}$ induced over the nodes in $\{v \in V(G) \mid X \in \chi(v)\}$ is a tree.

The *generalized hypertree width* of Q, denoted by $\mathsf{ghw}(Q)$, is the minimum width over all its generalized hypertree decompositions. □

Proposition 2. *Let J be a generalized hypertree decomposition. Then, J is consistent.*

Proof. (Sketch) It is sufficient to observe that when J is a generalized hypertree decomposition, the algorithms $\mathcal{A}_J$ in Figure 2 basically coincides with the evaluation algorithm in [16], proposed for the class of acyclic queries. □

Therefore, $\mathcal{C}(\mathsf{ghw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$ (cf. [3]) and, hence, when $\mathsf{ghw}(Q) \leq k$ and when a generalized hypertree decomposition J is given to hand, we can decide whether $Q[\mathcal{D}] = \emptyset$, for each database $\mathcal{D}$, by using Theorem 1.

Unfortunately, this approach to solve instances of bounded generalized hypertree width has two important drawbacks. First, its scaling is $O(\mathbf{q}^{3k}\mathbf{r}^{2k})$ as a direct consequence of the result on hyperconsistency width. And, second, no polynomial-time algorithm may exist to recognize instances in the class $\mathcal{C}(\mathsf{ghw}, k)$, unless P = NP (cf. [9]). Dealing with the intractability of ghw will be addressed in Section 4. Here, we focus instead on the former drawback, by relating the three notions studied in the paper and by observing that the nice scaling results of the ahcw can be extended to the generalized hypertree width.

Theorem 4. *For every natural number k, it holds that: $\mathcal{C}(\mathsf{ghw}, k) \subseteq \mathcal{C}(\mathsf{ahcw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$.*

Proof. (Sketch) We know that $\mathcal{C}(\mathsf{ahcw}, k) \subseteq \mathcal{C}(\mathsf{hcw}, k)$; hence, only $\mathcal{C}(\mathsf{ghw}, k) \subseteq \mathcal{C}(\mathsf{ahcw}, k)$ remains to be proven. Actually, since generalized hypertree decompositions are always acyclic, it suffices to show that when $\mathsf{ghw}(Q) \leq k$ for a query Q, a generalized hypertree decomposition of Q exists whose depth is bounded by size of Q. In fact, this follows from the game-theoretic characterization of generalized hypertree width in [10], where monotone winning strategies can be mapped into generalized hypertree decompositions; indeed, because of the monotonicity of the game, the depth of the generalized hypertree decomposition is bounded by $|\mathsf{var}(Q)|$. □

A consequence of the theorem above is that the algorithm for $\mathsf{SAT}^p_\emptyset[\mathcal{C}(\mathsf{ahcw}, k)]$ discussed in the proof of Theorem 3 is also an algorithm for $\mathsf{SAT}^{\mathsf{p}}_\emptyset[\mathcal{C}(\mathsf{ghw}, k)]$.

Corollary 1. $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}(\mathsf{ghw}, k)]$ *is feasible in* $O(\mathbf{q}^{2k+1}\mathbf{r}^{k})$.

To conclude the analysis on the relationships among the various notions, we observe that from [3] and Theorem 2 it follows that there are classes of queries where the generalized hypertree width is unbounded, while the hyperconsistency width remains bounded.

Indeed, this is because the hyperconsistency width is preserved under taking *cores* [12], while the generalized is not. Recall, here, that the *core* of a query Q, denoted by $\mathsf{CORE}(Q)$, is a query Q' such that: *(1)* $\mathsf{atoms}(Q') \subseteq \mathsf{atoms}(Q)$; *(2)* there is a mapping $f : \mathsf{var}(Q) \mapsto \mathsf{var}(Q')$ such that for each atom $r_i(X_1, ..., X_n) \in \mathsf{atoms}(Q)$, $r_i(f(X_1), ..., f(X_n))$ is in $\mathsf{atoms}(Q')$; and, *(3)* there is no query Q'' satisfying *(1)* and *(2)* such that $\mathsf{atoms}(Q'') \subset \mathsf{atoms}(Q')$.

In fact, we can show that a similar result holds when comparing the notion of acyclic hyperconsistency width and the generalized hypertree width.

Proposition 3. *There is a class* $\{Q_n \mid n > 0\}$ *of queries with* $\frac{\mathsf{ghw}(Q_n)}{\mathsf{ahcw}(Q_n)} \to +\infty$.

Proof. Let $(Q_n)_{n\in\mathbb{N}}$ be a class of queries such that $\mathsf{ghw}(Q_n) \to +\infty$. For each query Q_n, we select a variable $X \in \mathsf{var}(Q_n)$, and we define the novel query $Q'_n = Q_n \wedge \bigwedge\{r_n(X, \ldots, X) \mid r_n(S_n) \in \mathsf{atoms}(Q_n)\}$. Then, we can check that $\mathsf{ghw}(Q'_n) = \mathsf{ghw}(Q_n)$, $\mathsf{ghw}(\mathsf{CORE}(Q'_n)) = 1$, and $\mathsf{ahcw}(Q_n) = 1$. □

4 Complexity Results on No-Promise Problems

In this section, we complete our investigation by focusing on the description of incomplete algorithms for solving CSPs. The main aim is to extend the tractability result in Corollary 1 to the case of no-promise problems, i.e., when a generalized hypetree decomposition is not given to hand. Actually, besides the decision problem of checking whether $Q[\mathcal{D}]$ is empty for an instance $(Q, \mathcal{D})$, we will also consider the related computation problem $\mathsf{SAT}_{\exists}$ of computing an element in $Q[\mathcal{D}]$, and the problem $\mathsf{SAT}_{\forall}$ of computing the whole set $Q[\mathcal{D}]$.

Moreover, rather than focusing on some specific structural decomposition method, the algorithms we shall present correctly behave on any class of queries which is *stable*. Formally, a class $\mathcal{C}$ of queries is stable if, for each $Q \in \mathcal{C}$, for each unary constraint r not in Q, and for each variable X, we have that $Q \wedge r(X)$ is in $\mathcal{C}$.

Theorem 5. *Let* $\mathcal{C}$ *be a class of queries which is stable. Let* $\mathcal{A}$ *be an algorithm solving in* $O(\mathbf{q}^{a}\mathbf{r}^{b})$ *the problem:*

> **Input**: a CSP instance $(Q, \mathcal{D})$;
> **Output**: No only if $Q[\mathcal{D}] = \emptyset$, and
> IDon′tKnow only if $Q[\mathcal{D}] \neq \emptyset$ or $Q \notin \mathcal{C}$

Then, we can decide:

$\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a}\mathbf{r}^{b})$;	$\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$;
$\mathsf{SAT}^{\mathsf{p}}_{\exists}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$;	$\mathsf{SAT}^{\mathsf{np}}_{\exists}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$;
$\mathsf{SAT}^{\mathsf{p}}_{\forall}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1}\mathbf{o})$;	$\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$ *in* $O(\mathbf{q}^{a+2}\mathbf{r}^{b+2}\mathbf{o})$.

Input: a CSP instance $(Q, \mathcal{D})$;
Output: a solution $\theta \in Q[\mathcal{D}]$ only if $Q[\mathcal{D}] \neq \emptyset$, and
No only if $Q[\mathcal{D}] = \emptyset$, and
IDon'tKnow only if $Q \notin \mathcal{C}$

Given: An incomplete algorithm $\mathcal{A}$ solving the following :
Input: a CSP instance $(Q', \mathcal{D}')$;
Output: No only if $Q'[\mathcal{D}'] = \emptyset$, and
IDon'tKnow only if $Q'[\mathcal{D}'] \neq \emptyset$ or $Q' \notin \mathcal{C}$

---

begin
  if $\mathcal{A}(Q, \mathcal{D})$ returns No **then return** No;
  let $\theta = \emptyset$;
  for each variable $X \in \mathsf{var}(Q)$ **do**
    let r_X be a fresh predicate symbol of arity 1;
    let $A \in \mathsf{atoms}(Q)$ be such that $x \in \mathsf{var}(A)$;
    let $Q_X = Q \wedge r_X(X)$;
    let $l_X = \Pi_{\{X\}}(A[\mathcal{D}])$;
    for each $e \in l_X$ **do**
      let $\mathcal{D}_{X,e} = \mathcal{D} \cup \{r_X(e)\}$;
    if $\forall e \in l_X \;\; \mathcal{A}(Q_X, \mathcal{D}_{X,e})$ returns No **then**
      stop and **return** IDon'tKnow:
    else
      let e be s.t. $\mathcal{A}(Q_X, \mathcal{D}_{X,e})$ returns IDon'tKnow;
      $(Q, \mathcal{D}) := (Q_X, \mathcal{D}_{X,e})$;
      let $\theta(X) = e$;
  if $\theta \notin Q[\mathcal{D}]$ **then return** IDon'tKnow;
  else return θ;
end.

Fig. 3. Algorithm $\mathcal{A}^{\mathsf{np}}_{\exists}$ for $\mathsf{SAT}^{\mathsf{np}}_{\exists}[\mathcal{C}]$ (cf. [3])

Proof. ($\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$). We can solve $\mathsf{SAT}^{\mathsf{p}}_{\emptyset}[\mathcal{C}]$ by means of an algorithm $\mathcal{A}^{\mathsf{p}}_{\emptyset}$ that returns No iff $\mathcal{A}(Q, \mathcal{D})$ returns No, and returns Yes iff $\mathcal{A}(Q, \mathcal{D})$ returns IDon'tKnow.

($\mathsf{SAT}^{\mathsf{np}}_{\exists}[\mathcal{C}]$). Consider the algorithm $\mathcal{A}^{\mathsf{np}}_{\exists}$ reported in Figure 3, that is basically the algorithm discussed in [3] for the class of queries having bounded generalized hypertree width. This algorithm only returns No when $\mathcal{A}(Q, \mathcal{D})$ returns No (in which case, $Q[\mathcal{D}] = \emptyset$). Otherwise, i.e., when $Q[D] \neq \emptyset$, the algorithm starts computing a solution θ by iteratively fixing a value $e \in l_X$, for each variable X such that $Q_X[D_{X,e}] \neq \emptyset$. Note that during the computation, the query Q is updated by adding a unary constraint precisely fixing the value for X. Hence, provided that the class $\mathcal{C}$ is stable, any such modification preserves the membership of Q in $\mathcal{C}$. Finally, the algorithm returns IDon'tKnow iff θ is not a solution. Note that when $Q[D] \neq \emptyset$, there always exists at least one value $e \in l_X$, for each variable X such that $Q_X[D_{X,e}] \neq \emptyset$. For the complexity of $\mathcal{A}^{\mathsf{np}}_{\exists}$, note that the size of Q (resp. $\mathcal{D}$) always remains smaller than $2\mathbf{q}$ (resp. $2\mathbf{r}$) and therefore

Input: a CSP instance $(Q, \mathcal{D})$, a set $\mathcal{O} \subseteq \mathsf{var}(Q)$;
Output: $\Pi_{\mathcal{O}}(Q[\mathcal{D}])$, and
IDon'tKnow only if $Q \notin \mathcal{C}$;

Given: An algorithm $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ for $\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$

begin
  if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q, \mathcal{D})$ returns IDon'tKnow **then**
    stop and **return** IDon'tKnow;
  if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q, \mathcal{D})$ returns No **then**
    stop and **return** $\emptyset$;
  if $\mathcal{O} = \emptyset$ **then**
    stop and **return** $\emptyset$;
  else
    choose a variable X in $\mathcal{O}$ and **let** $\mathcal{O}' = \mathcal{O} - \{X\}$;
    let r_X be a fresh predicate symbol of arity 1;
    let $A \in \mathsf{atoms}(Q)$ be such that $X \in \mathsf{var}(A)$;
    let $Q_X = Q \wedge r_X(X)$, $l_X = \Pi_{\{X\}}A[\mathcal{D}]$, and $\mathcal{R} = \emptyset$;
    for each $e \in l_X$
      let $\mathcal{D}_{X,e} := \mathcal{D} \cup \{r_X(e)\}$
      if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_X, \mathcal{D}_{X,e})$ returns IDon'tKnow **then**
        stop and **return** IDon'tKnow
      if $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_X, \mathcal{D}_{X,e})$ returns Yes **then**
        for each $h' \in \mathcal{A}^{\mathsf{np}}_{\forall}(Q_X, \mathcal{D}_{X,e}, \mathcal{O}')$ **do**
          let $h'(X) := e$;
          $\mathcal{R} := \mathcal{R} \cup \{h'\}$;
    return $\mathcal{R}$;
end.

Fig. 4. Algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$ for $\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$

each call to the algorithm $\mathcal{A}$ costs $O(\mathbf{q}^a\mathbf{r}^b)$. The number of such calls is bounded by $\Sigma\{|l_X|, X \in \mathsf{var}(Q)\} = O(\mathbf{qr})$. Therefore, $\mathcal{A}^{\mathsf{np}}_{\exists}$ terminates in time $O(\mathbf{q}^{a+1}\mathbf{r}^{a+1})$.

($\mathsf{SAT}^{\mathsf{np}}_{\emptyset}[\mathcal{C}]$ and $\mathsf{SAT}^{\mathsf{p}}_{\exists}[\mathcal{C}]$). We can use the algorithm $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ which simply returns Yes iff $\mathcal{A}^{\mathsf{np}}_{\exists}$ returns one solution $\theta \in Q[D]$, and the same output of $\mathcal{A}^{\mathsf{np}}_{\exists}$, if no solution is computed. Clearly enough, $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ is also an algorithm for $\mathsf{SAT}^{\mathsf{p}}_{\exists}[\mathcal{C}]$.

($\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$). Armed with $\mathcal{A}^{\mathsf{np}}_{\emptyset}$, we present in Figure 4 the recursive algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$ for $\mathsf{SAT}^{\mathsf{np}}_{\forall}[\mathcal{C}]$, taking as extra-argument a set of variables $\mathcal{O}$ (for $\mathcal{O} = \mathsf{var}(Q)$, the whole relation $Q[\mathcal{D}]$ is computed). The algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$ only returns a relation $\mathcal{R} = \Pi_{\mathcal{O}}(Q[\mathcal{D}])$ when every call to $\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_X, \mathcal{D}_{X,e})$ have returned either Yes or No, in which case we know by correctness of $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ that we didn't forget any (or add any extra) tuple in $\Pi_{\mathcal{O}}(Q[\mathcal{D}])$. Therefore, $\mathcal{A}^{\mathsf{np}}_{\forall}$ only returns IDon'tKnow when $Q \notin \mathcal{C}$. Each call to $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ costs $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1})$. Indeed, $\mathcal{A}^{\mathsf{np}}_{\forall}$ first executes $\mathbf{r}$ calls to the algorithm $\mathcal{A}^{\mathsf{np}}_{\emptyset}$ for the first variable $X \in \mathcal{O}$, and then applies $\mathbf{r}$ more calls each time that the case "$\mathcal{A}^{\mathsf{np}}_{\emptyset}(Q_x, \mathcal{D}_{x,e})$ returns Yes" appears. This may happen $O(|\mathcal{O}| \times \mathbf{o}) = O(\mathbf{qo})$ times at most. Therefore, $\mathcal{A}^{\mathsf{np}}_{\forall}$ terminates in time $O(\mathbf{qro}) \times O(\mathbf{q}^{a+1}\mathbf{r}^{b+1}) = O(\mathbf{q}^{a+2}\mathbf{r}^{b+2}\mathbf{o})$.

($\mathsf{SAT}^{\mathsf{p}}_{\forall}[\mathcal{C}]$). The problem can be solved in time $O(\mathbf{q}^{a+2}\mathbf{r}^{b+2}\mathbf{o})$, by a slight variation of the algorithm $\mathcal{A}^{\mathsf{np}}_{\forall}$, where $\mathcal{A}^{\mathsf{p}}_{\emptyset}$ is used instead of $\mathcal{A}^{\mathsf{np}}_{\emptyset}$. The complexity is then $O(\mathbf{q}^{a+1}\mathbf{r}^{b+1}\mathbf{o})$. □

As an example application of the above theorem, we next show that the class of queries having bounded generalized hypertree width is stable, so that bounds in Figure 1 are correct.

Proposition 4. *For every k, the class $\mathcal{C}(\mathsf{ghw}, k)$ is stable.*

Proof. Let $J = (\mathcal{G}, \chi, \lambda)$ be a k-width generalized hypergraph decomposition of Q, let r be a unary constraint not in Q, let X be a variable, and let $Q' = Q \wedge r(X)$. We build a hypergraph decomposition J' by adding a node v' to $V(J)$ such that $\chi(v') = \{X\}$ and $\lambda(v') = \{r(X)\}$, and an edge (v, v') to $E(J)$, where v is an arbitrarily chosen node such that $X \in \chi(v)$. Then, J' is a k-width generalized hypertree decomposition. □

5 Discussion and Conclusion

We have characterized the computational complexity of solving the promise and no-promise problems on the class of queries having bounded generalized hypertree width, by means of consistency-based algorithms. On the one hand, we have provided precise bounds for the technique of [3], which firstly suggested the idea of no-promise algorithms for $\mathcal{C}(\mathsf{ghw}, k)$. On the other hand, the notion of hyperconsistency width being parameterized by the underlying decomposition allowed us to obtain improved complexity bounds. These bounds have been established for both the case of promise and no-promise problems, and for the problems SAT, $\mathsf{SAT}_{\exists}$, and $\mathsf{SAT}_{\forall}$.

It turned out that instances of bounded generalized hypertree width can be solved with the enhanced no-promise approach by means of an algorithm whose scaling is the same as current evaluation algorithms for instances of bounded hypertree width (see [8], for the description of these algorithms), but with the advantage of isolating a larger class of instances (recall that $\mathcal{C}(\mathsf{ghw}, k) \supset \mathcal{C}(\mathsf{hw}, k)$). This is relevant in the light that $\mathcal{C}(\mathsf{hw}, k)$ is one of the largest classes of tractable CSPs that is known to be also efficiently recognizable (see, e.g., [2,7]).

Several research questions naturally arise from this contribution. First, assessing the *exact* relationship among the classes $\mathcal{C}(\mathsf{ghw}, k)$, $\mathcal{C}(\mathsf{hcw}, k)$, and $\mathcal{C}(\mathsf{ahcw}, k)$ is still an open problem. In particular, given that one may easily observe that for each query Q, it is the case that $\mathsf{hcw}(Q)$=$\mathsf{hcw}(\mathsf{CORE}(Q))$, it is natural to ask whether $\mathsf{ghw}(\mathsf{CORE}(Q))$ coincides with $\mathsf{hcw}(Q)$ (and $\mathsf{ahcw}(Q)$).

Second, the (improved) no-promise algorithm scales as $O(\mathbf{q}^{2k})$ in the size of the query. Interestingly, this is also the best upper bound known for computing a hypertree decomposition. Whether it is possible to improve on this exponent appears therefore a crucial question for a deeper and unifying understating of these notions.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley, Reading (1995)
2. Adler, I., Gottlob, G., Grohe, M.: Hypertree-width and related hypergraph invariants. In: Proc. of EUROCOMB 2005, pp. 5–10 (2005)
3. Chen, H., Dalmau, V.: Beyond hypertree width: Decomposition methods without decompositions. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 167–181. Springer, Heidelberg (2005)
4. Dechter, R.: From local to global consistency. Artificial Intelligence 55(1), 87–108 (1992)
5. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
6. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. Journal of the ACM 49(6), 716–752 (2002)
7. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural csp decomposition methods. Artificial Intelligence 124(2), 243–282 (2000)
8. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. Journal of Computer and System Sciences 64(3), 579–627 (2002)
9. Gottlob, G., Miklos, Z., Schwentick, T.: Generalized hypertree decompositions: Np-hardness and tractable variants. In: Proc. of PODS 2007 (2007)
10. Greco, G., Scarcello, F.: Tree projections: Hypergraph games and minimality. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 736–747. Springer, Heidelberg (2008)
11. Gyssens, M., Jeavons, P., Cohen, D.A.: Decomposing constraint satisfaction problems using database techniques. Artificial Intelligence 66(1), 57–89 (1994)
12. Hell, P., Nesetril, J.: The core of a graph. Discrete Math. 109(1-3), 117–126 (1992)
13. Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. Journal of Computer and System Sciences 61(2), 302–332 (1998)
14. Kolaitis, P.G., Vardi, M.Y.: A game-theoretic approach to constraint satisfaction. In: Proc. of AAAI 2000, pp. 175–181 (2000)
15. Pearson, J., Jeavons, P.: A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London (July 1997)
16. Yannakakis, M.: Algorithms for acyclic database schemes. In: Proc. of VLDB 1981, pp. 82–94 (1981)