

Tree Projections and Constraint Optimization Problems: Fixed-Parameter Tractability and Parallel Algorithms

Georg Gottlob^a, Gianluigi Greco^{b,*}, Francesco Scarcello^c

^a*Department of Computer Science, University of Oxford, UK*

^b*Department of Mathematics and Computer Science, University of Calabria, Italy*

^c*DIMES, University of Calabria, Italy*

Abstract

Tree projections provide a unifying framework to deal with most structural decomposition methods of constraint satisfaction problems (CSPs). Within this framework, a CSP instance is decomposed into a number of sub-problems, called views, whose solutions are either already available or can be computed efficiently. The goal is to arrange portions of these views in a tree-like structure, called tree projection, which determines an efficiently solvable CSP instance equivalent to the original one. However, deciding whether a tree projection exists is NP-hard. Solution methods have therefore been proposed in the literature that do not require a tree projection to be given, and that either correctly decide whether the given CSP instance is satisfiable, or return that a tree projection actually does not exist. These approaches had not been generalized so far to deal with CSP extensions tailored for optimization problems, where the goal is to compute a solution of maximum value/minimum cost. The paper fills the gap, by exhibiting a fixed-parameter polynomial-time algorithm that either disproves the existence of tree projections or computes an optimal solution, with the parameter being the size of the expression of the objective function to be optimized over all possible solutions (and not the size of the whole constraint formula, used in related works). Tractability results are also established for the problem of returning the best K solutions. Finally, parallel algorithms for such optimization problems are proposed and analyzed.

Given that the classes of acyclic hypergraphs, hypergraphs of bounded treewidth, and hypergraphs of bounded generalized hypertree width are all covered as special cases of the tree projection framework, the results in this paper directly apply to these classes. These classes are extensively considered in the CSP setting, as well as in conjunctive database query evaluation and optimization.

Keywords: Constraint Satisfaction Problems, AI, Optimization Problems, Structural Decomposition Methods, Tree Projections, Parallel Models of Computation, Conjunctive Queries, Query Optimization, Database Theory.

*Corresponding author

Email addresses: gottlob@cs.ox.ac.uk (Georg Gottlob), ggreco@mat.unical.it (Gianluigi Greco), scarcello@dimes.unical.it (Francesco Scarcello)

Preprint submitted to Elsevier

November 14, 2017

1. Introduction

1.1. Optimization in Constraint Satisfaction Problems

Constraint satisfaction is a central topic of research in Artificial Intelligence, and has a wide spectrum of concrete applications ranging from configuration to scheduling, plan design, temporal reasoning, and machine learning, just to name a few.

Formally, a constraint satisfaction problem (for short: CSP) instance is a triple $\mathcal{I} = \langle \text{Var}, \mathcal{U}, \mathcal{C} \rangle$, where Var is a finite set of variables, \mathcal{U} is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \dots, C_q\}$ is a finite set of constraints (see, e.g., [1]). Each constraint C_v , with $v \in \{1, \dots, q\}$, is a pair (S_v, r_v) , where $S_v \subseteq \text{Var}$ is a set of variables called the *constraint scope*, and r_v is a set of assignments from variables in S_v to values in \mathcal{U} indicating the allowed combinations of values for the variables in S_v . A (partial) assignment from a set of variables $\mathcal{W} \subseteq \text{Var}$ to \mathcal{U} is explicitly represented by the set of pairs of the form X/u , where $u \in \mathcal{U}$ is the value to which $X \in \mathcal{W}$ is mapped. An assignment θ *satisfies* a constraint C_v if its restriction to S_v , i.e., the set of pairs $X/u \in \theta$ such that $X \in S_v$, occurs in r_v . A *solution* to \mathcal{I} is a (total) assignment $\theta : \text{Var} \mapsto \mathcal{U}$ for which q satisfying assignments $\theta_1 \in r_1, \dots, \theta_q \in r_q$ exist such that $\theta = \theta_1 \cup \dots \cup \theta_q$. Therefore, a solution is a total assignment that satisfies all the constraints in \mathcal{I} .

By *solving a CSP instance* we usually just mean finding any arbitrary solution. However, when assignments are associated with weights because of the semantics of the underlying application domain, we might instead be interested in the corresponding *optimization problem* of finding the solution of maximum or minimum weight (short: MAX and MIN problems), whose modeling is possible in several variants of the basic CSP framework, such as the *valued* and *semiring-based CSPs* [2]. Moreover, we might be interested in the TOP- K problem of enumerating the best (w.r.t. MAX or MIN) K solutions in form of a ranked list (see, e.g., [3, 4]),¹ or even in the NEXT problem of computing the next solution (w.r.t. such an ordering) following one that is at hand [5].

CSP instances, as well as their extensions tailored to model optimization problems, are computationally intractable. Indeed, even just deciding whether a given instance admits a solution is a well-known NP-hard problem, which calls for practically effective algorithms and heuristics, and for the identification of specific subclasses, called “islands of tractability”, over which the problem can be solved efficiently. In this paper, we consider the latter perspective to attack CSP instances, by looking at *structural properties* of constraint scopes.

1.2. Structural Decomposition Methods and Tree Projections

The avenue of research looking for islands of tractability based on structural properties originated from the observation that constraint satisfaction is tractable on *acyclic* instances (cf. [6, 7]), i.e., on instances whose associated hypergraph (whose hyperedges correspond one-to-one to the sets of variables in the given constraints) is acyclic.²

Motivated by this result, *structural decomposition methods* have been proposed in the literature as approaches to transform any given cyclic CSP into an equivalent acyclic

¹Related results on graphical models, conjunctive query evaluation, and computing homomorphisms on relational structures are transparently recalled hereinafter in the context of constraint satisfaction.

²There are different notions of hypergraph acyclicity. In the paper, we consider α -acyclicity, which is the most liberal one [8].

one by organizing its constraints or variables into a polynomial number of clusters and by arranging these clusters as a tree, called *decomposition tree*. The satisfiability of the original instance can be then checked by exploiting this tree, with a cost that is exponential in the cardinality of the largest cluster, also called *width* of the decomposition, and polynomial if the width is bounded by a constant (see [9] and the references therein). Similarly, by exploiting this tree, solutions can be computed even to CSP extensions tailored for optimization problems, again with a cost that is polynomial over bounded-width instances. For instance, we know that (in certain natural optimization settings) MAX is feasible in polynomial time over instances whose underlying hypergraphs are acyclic [10], have bounded *treewidth* [3], or have bounded *hypertree width* [11, 12].

Despite their different technical definitions, there is a simple framework encompassing all structural decomposition methods,³ which is the framework of the *tree projections* [14]. The basic idea of these methods is indeed to “cover” all the given constraints via a polynomial number of clusters of variables and to arrange these clusters as a tree, in such a way that the *connectedness condition* holds, i.e., for each variable $X \in Var$, the subgraph induced by the clusters containing X is a tree. In particular, any cluster identifies a subproblem of the original instance, and it is required that all solutions to this subproblem can either be computed efficiently, or are already available (e.g., from previous computations). A tree built from the available clusters and covering all constraints is called a *tree projection* [14, 15, 16, 17]. In particular, whenever such clusters are required to satisfy additional conditions, tree projections reduce to specific decomposition methods. For instance, if we consider candidate clusters given by all subproblems over $k + 1$ variables at most (resp., over any set of variables contained in the union of k constraints at most), then tree projections correspond to *tree decompositions* [18, 1] (resp., *generalized hypertree decompositions* [19, 16]), and k is their associated width.

Deciding whether a tree projection exists is NP-hard in general, that is, when a set of arbitrary clusters/subproblems is given [16]. Moreover, the problem remains intractable in some specific settings, such as (bounded width) generalized hypertree decompositions [16]. Therefore, designing tractable algorithms within the framework of tree projections is not an easy task. Ideally we would like to efficiently solve the instances without requiring that a tree projection be explicitly computed (or provided as part of the input). For standard CSP instances, algorithms of this kind have already been exhibited [15, 14, 20, 21]. These algorithms are based on *enforcing pairwise-consistency* [22], also known in the CSP community as *relational arc consistency* (or arc consistency on the dual graph) [1], *2-wise consistency* [23], and $R(*, 2)C$ [24]. Note that these algorithms are mostly used in heuristics for constraint solving algorithms. The idea is to repeatedly take—until a fixpoint is reached—any two constraints $C_i = (S_i, r_i)$ and $C_j = (S_j, r_j)$ and to remove from r_i all assignments θ_i that cannot be extended over the variables in S_j , i.e., for which there is no assignment $\theta_j \in r_j$ such that the restrictions of θ_i and θ_j over the variables in $S_i \cap S_j$ coincide. Here, the crucial observation is that the order according to which pairs of constraints are processed is immaterial, so that this procedure is equivalent to Yannakakis’ algorithm [7], which identifies a correct processing order based on the knowledge of a tree projection.⁴ Actually, it is even unnecessary to know that a tree

³The notion of *submodular* width [13] does not fit this framework, as it is not purely structural.

⁴The algorithm has been originally proposed for acyclic instances. For its application within the tree projection setting, the reader is referred to [21].

projection exists at all, because any candidate solution can be certified in polynomial time. Indeed, these algorithms are designed in a way that, whenever some assignment is computed that is subsequently found not to be a solution, then the (promised) existence of a tree projection is disproved. We define these algorithms computing certified solutions as *promise-free*, with respect to the existence of a tree projection (cf. [20, 21]). We note that, so far, this kind of solution approach has not been generalized in the literature to deal with CSP extensions tailored for optimization problems.

1.3. Contributions

All previous algorithms proposed in the literature for computing the best CSP solutions in polynomial time [3, 11, 25, 26, 27, 28, 12, 29, 30] (or, more generally, for optimizing functions in different application domains—see, e.g., [31]) require the knowledge of some suitable tree projection, which provides at each node a list of potentially good partial evaluations with their associated values to be propagated within a dynamic programming scheme. The main conceptual contribution of the present paper is to show that this knowledge is not necessary, since promise-free algorithms can be exhibited in the tree projection framework even when dealing with optimization problems.

More formally, we consider a setting where the given CSP instance \mathcal{I} is equipped with a valuation function \mathcal{F} to be maximized over the feasible solutions. The function is built from basic weight functions defined on subsets of variables occurring in constraint scopes, combined via some binary operator \oplus .⁵ Moreover, we assume that a set of subproblems \mathcal{V} is given together with their respective solutions. Then, within this setting,

- ▷ We provide a *fixed-parameter polynomial-time* algorithm [32] for MAX that either computes a solution (if one exists) having the best weight according to \mathcal{F} , or says that no tree projection can be built by using the available subproblems in \mathcal{V} . In any case, the algorithm does not output any wrong answer, because the computed solutions are certified. More precisely, the algorithm runs in time $f(\kappa) \times n^c$, where the parameter κ is the number of basic functions occurring in \mathcal{F} , n is the size of the input, and c is a fixed natural number. Thus, the running time has no exponential dependency on the input, but possibly on the fixed parameter κ .
- ▷ We show that the TOP- K problem of returning the best K solutions over all possible solutions is fixed-parameter tractable, too. As we may have an exponential number of solutions (w.r.t. n), tractability means here having a promise-free algorithm that computes the desired output *with fixed-parameter polynomial delay*: The first solution is computed in fixed-parameter polynomial-time, and any other solution is computed within fixed-parameter polynomial-time after the previous one.
- ▷ Moreover, we complement the above research results, by studying the setting where a tree projection is given at hand. In this case, we show that the task of computing the best solutions over a set of output variables is not only feasible in polynomial time (as we already know from the literature pointed out above), but it is even

⁵In fact, our results are designed to hold in a more general setting where different binary operators may be used together in the definition of more complex valuation functions. However, for the sake of presentation, we shall mainly focus on a single operator, in the spirit of the (standard) valued and semiring-based CSP settings.

possible to define parallel algorithms that can exploit the availability of machines with multiple processors.

Concerning our main technical contributions, we stress here that different kinds of fixed-parameter polynomial-time algorithms can be defined for the problems of interests when varying the underlying parameter of interest. For instance, a trivial choice would be to consider the overall number of constraints involved in the CSP at hand. In fact, our parameter is very often much smaller, so that our algorithms can be useful in all those applications where the optimization function consists of few basic functions, while the number of constraints is large (which makes infeasible computing any tree projection).

Organization. The rest of the paper is organized as follows. Section 2 illustrates some basic notions about CSPs and their structural properties. The formal framework for equipping CSP instances with optimization functions is introduced in Section 3. Our fixed-parameter tractability results are illustrated in Section 4. Parallel algorithms are presented in Section 5. Relevant related works are discussed in Section 6, and concluding remarks are drawn in Section 7.

2. Preliminaries

Logic-Based Modeling of Constraint Satisfaction. Let $\mathcal{I} = \langle \text{Var}, \mathcal{U}, \mathcal{C} \rangle$ be a CSP instance, with $\mathcal{C} = \{(S_1, r_1), (S_2, r_2), \dots, (S_q, r_q)\}$. Following [33], we shall exploit throughout the paper the logic-based characterization of \mathcal{I} as a pair (Φ, DB) , which simplifies the illustration of structural tractability results. In particular, Φ is the *constraint formula* (associated with \mathcal{I}), i.e., a conjunction of atoms of the form $r_1(\mathbf{u}_1) \wedge \dots \wedge r_q(\mathbf{u}_q)$ where \mathbf{u}_v , for each $v \in \{1, \dots, q\}$, is obtained by listing all the variables in the scope S_v . The set of variables in Φ is denoted by $\text{vars}(\Phi)$, while the set of atoms occurring in Φ is denoted by $\text{atoms}(\Phi)$. Moreover, DB is the *constraint database*, i.e., a set of ground atoms encoding the allowed tuples of values for each constraint, built as follows. For each constraint index $v \in \{1, \dots, q\}$ and for each assignment $\theta_v \in r_v$, DB contains the ground atom $r_v(a_1, \dots, a_{|S_v|})$ where if X_i is the i -th variable in the list \mathbf{u}_v , then $a_i = \theta_v(X_i)$ holds for each $i \in \{1, \dots, |S_v|\}$. No further ground atom is in DB.

In the following, for any set \mathcal{W} of variables and any assignment θ , $\theta[\mathcal{W}]$ denotes the partial assignment obtained by restricting θ to the variables in \mathcal{W} . Therefore, a (total) substitution θ is a solution to (Φ, DB) if $\theta[S_v] \in r_v$ holds for each $v \in \{1, \dots, q\}$. The set of all solutions to the CSP instance (Φ, DB) is denoted by Φ^{DB} . Moreover, for any set \mathcal{W} of variables, $\Phi^{\text{DB}}[\mathcal{W}]$ denotes the set $\{\theta[\mathcal{W}] \mid \theta \in \Phi^{\text{DB}}\}$.

Structural Properties of CSP Instances. The structure of a constraint formula Φ is best represented by its associated hypergraph $\mathcal{H}_\Phi = (N, H)$, where $N = \text{vars}(\Phi)$, i.e., variables are viewed as nodes, and where $H = \{S_1, \dots, S_q\}$, i.e., for each atom in Φ , H contains a hyperedge including all its variables. For any hypergraph \mathcal{H} , we denote the sets of its nodes and of its hyperedges by $\text{nodes}(\mathcal{H})$ and $\text{edges}(\mathcal{H})$, respectively.

A hypergraph \mathcal{H} is *acyclic* if it has a *join tree* [34]. A *join tree* JT of \mathcal{H} is a labeled tree (V, E, χ) , where for each vertex $v \in V$, it holds that $\chi(v) \in \text{edges}(\mathcal{H})$, and where the following conditions are satisfied:

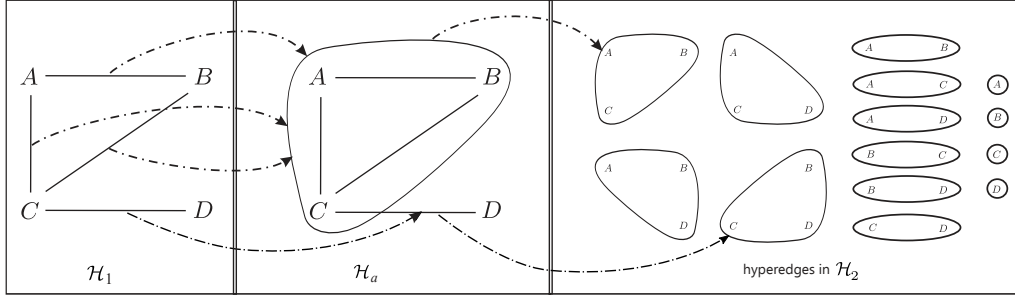


Figure 1: Structures discussed in Example 2.3.

Covering Condition: $\forall h \in \text{edges}(\mathcal{H})$, for some vertex v of JT , $h = \chi(v)$ holds;

Connectedness Condition: for each pair of vertices v_1, v_2 in V such that $\chi(v_1) \cap \chi(v_2) = \mathcal{W} \neq \emptyset$, v_1 and v_2 are connected in JT (via edges from E) and $\mathcal{W} \subseteq \chi(v)$, for every vertex v in the unique path linking v_1 and v_2 in JT .

Note that this definition is apparently more liberal than the traditional one (in [34]), where there is a one-to-one correspondence between hyperedges and vertices of the join tree. We find it convenient to allow multiple occurrences of the same hyperedge in the χ labels of different vertices of JT , but it is straightforward to show that a standard join tree may be obtained from JT by repeatedly contracting edges of the form $\{v_1, v_2\}$, where $\chi(v_1) \subseteq \chi(v_2)$ (until such a one-to-one correspondence is met).

Decomposition Methods. Structural decomposition methods have been proposed in the literature in order to provide a measure of the degree of acyclicity of hypergraphs, and in order to generalize positive computational results from acyclic hypergraphs to nearly-acyclic ones. Despite their different technical definitions, there is a simple framework encompassing all known (purely structural) decomposition methods. The framework is based on the concept of *tree projection* [14], which is recalled below.

Definition 2.1. For two hypergraphs \mathcal{H}_1 and \mathcal{H}_2 , we say that \mathcal{H}_2 *covers* \mathcal{H}_1 , denoted by $\mathcal{H}_1 \leq \mathcal{H}_2$, if each hyperedge of \mathcal{H}_1 is contained in at least one hyperedge of \mathcal{H}_2 . Let $\mathcal{H}_1 \leq \mathcal{H}_2$. Then, a *tree projection* of \mathcal{H}_1 with respect to \mathcal{H}_2 is an acyclic hypergraph \mathcal{H}_a such that $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. Whenever such a hypergraph \mathcal{H}_a exists, we say that the pair $(\mathcal{H}_1, \mathcal{H}_2)$ has a tree projection. \square

Example 2.2. Consider the hypergraph \mathcal{H}_1 depicted in Figure 1, and the hypergraph \mathcal{H}_2 whose hyperedges are listed on the right of the same figure. Note that \mathcal{H}_1 is (just) a graph and it contains a cycle over the nodes A , B , and C .

The acyclic hypergraph \mathcal{H}_a shown in the middle is a tree projection of \mathcal{H}_1 w.r.t. \mathcal{H}_2 . For instance, note that the cycle is “absorbed” by the hyperedge $\{A, B, C\}$, which is in its turn trivially contained in a hyperedge of \mathcal{H}_2 . \triangleleft

Following [21], tree projections can be used to solve any CSP instance (Φ, DB) whenever we have (or we can build) an additional pair $(\mathcal{V}, \text{DB}')$ such that:

- \mathcal{V} is a set of atoms (hence, corresponding to a set of constraint scopes). Each atom in \mathcal{V} clusters together the variables of a subproblem whose solutions are assumed to be available in the *constraint database* DB' and that can be exploited in order to answer the original CSP instance (Φ, DB) . Atoms in \mathcal{V} will be called *views*, and \mathcal{V} will be called *view set*. It is required that, for each atom $q \in \text{atoms}(\Phi)$, \mathcal{V} contains a *base view* w_q with the same list of variables as q .
- DB' is a constraint database that satisfies the following conditions:
 - (i) $w_q^{\text{DB}'} \subseteq q^{\text{DB}}$ holds for each base view $w_q \in \mathcal{V}$; that is, base views should be at least as restrictive as atoms in the constraint formula;
 - (ii) $w^{\text{DB}'} \supseteq \Phi^{\text{DB}}[w]$ holds for each $w \in \mathcal{V}$; that is, any view cannot be more restrictive than the constraint formula, otherwise correct solutions may be deleted by performing operations involving such views.

Such a database DB' is said *legal* for \mathcal{V} w.r.t. Φ and DB .

The pair $(\mathcal{V}, \text{DB}')$ is used as follows. Let $\mathcal{H}_{\mathcal{V}}$ denote the view hypergraph precisely containing, for each view w in \mathcal{V} , one hyperedge over the variables in w . We look for a *sandwich formula of Φ w.r.t. \mathcal{V}* , that is, a constraint formula Φ_a such that $\text{atoms}(\Phi_a)$ includes all base views and \mathcal{H}_{Φ_a} is a tree projection of \mathcal{H}_{Φ} w.r.t. $\mathcal{H}_{\mathcal{V}}$. By exploiting the sandwich formula Φ_a , solving Φ can be reduced to answering an acyclic instance, hence to a task which is feasible in polynomial time. Indeed, by projecting the assignments of any legal database DB' over the (portions of the) views used in Φ_a , a novel database DB'_a can be obtained such that $\Phi_a^{\text{DB}'_a} = \Phi^{\text{DB}}$ [14].

Most structural decomposition methods of constraint satisfaction problems can be viewed as special instances of this approach, where the peculiarities of each method lead to different ways of building the additional view set \mathcal{V} , with its associated database DB' . For instance, the methods based on *generalized hypertree decompositions* [19, 16] and *tree decompositions* [18], for a constant *width* k , fit into the framework as follows:

k -width generalized hypertree decompositions: The method uses a set $h\text{-}\mathcal{V}_k$ of views including, for each subformula Φ' of Φ with $\text{atoms}(\Phi') \subseteq \text{atoms}(\Phi)$ and $|\text{atoms}(\Phi')| \leq k$, a view that is built over the set of all variables on which these atoms are defined (hence, base views are obtained for $k=1$) and whose assignments in the corresponding constraint database $h\text{-}\text{DB}'_k$ are all solutions to (Φ', DB) .

k -width tree decompositions: The method uses the set \mathcal{V}_k of views consisting of the base views plus all the views that can be built over all possible sets of at most $k+1$ variables. In the associated constraint relations in DB'_k , base views consist of the assignments in the corresponding atoms in DB , whereas each of the remaining views contains all possible assignments that can be built over them, hence $|U|^{k+1}$ assignments at most, where U is the size of the largest domain over the selected $k+1$ variables.

Example 2.3. Consider a CSP instance (Φ, DB) such that $\Phi = r_1(A, B) \wedge r_2(B, C) \wedge r_3(A, C) \wedge r_4(C, D)$ and where $r_i(0, 0)$ and $r_i(1, 1)$ are the only two ground atoms in DB , for each $i \in \{1, 2, 3, 4\}$. The constraint hypergraph associated with Φ is precisely the hypergraph \mathcal{H}_1 illustrated in Figure 1. Since $\mathcal{H}_{\Phi} = \mathcal{H}_1$ is not acyclic, our goal is to

apply a structural decomposition method for transforming the original instance Φ into a novel acyclic one Φ_a that “covers” all constraints in Φ and is equivalent to it. To this end, let us consider the application on (Φ, DB) of the tree decomposition method with k being the associated width, resulting in the pair $(\mathcal{V}_k, \text{DB}'_k)$. For instance, the base view $w_{r_4(C,D)}(C, D)$ is in \mathcal{V}_k and its associated tuples in DB'_k are $w_{r_4(C,D)}(0, 0)$ and $w_{r_4(C,D)}(1, 1)$. Moreover, for each natural number $k > 1$, a view having the form $w_{A,B,C}(A, B, C)$ is in \mathcal{V}_k and the associated tuples in DB'_k are $w_{A,B,C}(\alpha, \beta, \gamma)$, with $\alpha, \beta, \gamma \in \{0, 1\}$. In particular, for $k = 2$, the hypergraph $\mathcal{H}_{\mathcal{V}_2}$ precisely coincides with the hypergraph \mathcal{H}_2 (whose hyperedges are) illustrated in Figure 1. Consider then the constraint formula

$$\Phi_a = w_{A,B,C}(A, B, C) \wedge w_{r_1(A,B)}(A, B) \wedge w_{r_2(B,C)}(B, C) \wedge w_{r_3(A,C)}(A, C) \wedge w_{r_4(C,D)}(C, D),$$

and note that \mathcal{H}_{Φ_a} coincides with the acyclic hypergraph \mathcal{H}_a , which is a tree projection⁶ of \mathcal{H}_Φ w.r.t. $\mathcal{H}_{\mathcal{V}_2}$. Then, solving (Φ, DB) is equivalent to solving (Φ_a, DB'_a) where DB'_a is just the restriction of DB'_2 over the atoms in $\text{atoms}(\Phi_a)$. \triangleleft

3. Valuation Functions and Basic Results

In this section, we illustrate a formal framework for equipping constraint formulas with valuation functions suited to express a variety of optimization problems. Moreover, we introduce and analyze a notion of embedding as a way to represent and study the interactions between constraint scopes and valuation functions.

In the following we assume that a domain \mathcal{U} of values, a constraint formula Φ , and a set \mathbb{D} of weights totally ordered by a relation \geq are given. Moreover, on the set \mathbb{D} , we define max and min as the operations returning any \geq -maximum and the \geq -minimum weight, respectively, over a given set of weights.

3.1. Formal Framework

Let $\mathcal{W} \subseteq \text{vars}(\Phi)$ be a set of variables. Then, a function f associating each assignment $\theta : \mathcal{W} \mapsto \mathcal{U}$ with a weight $f(\theta) \in \mathbb{D}$ is called a *weight function* (for Φ), and we denote by $\text{vars}(f)$ the set \mathcal{W} on which it is defined. If an assignment $\theta' : \mathcal{W}' \mapsto \mathcal{U}$ with $\mathcal{W}' \supseteq \text{vars}(f)$ is given, then we write $f(\theta')$ as a shorthand for $f(\theta'[\text{vars}(f)])$.

Definition 3.1. Let \oplus be a closed, commutative, and associative binary operator over \mathbb{D} being, moreover, distributive over max. A *valuation function* \mathcal{F} (for Φ over \oplus) is an expression of the form $f_{\mathbf{u}_1}^1 \oplus \dots \oplus f_{\mathbf{u}_m}^m$, with $m \geq 1$. The set of all weight functions occurring in \mathcal{F} is denoted by $\text{wfs}(\mathcal{F})$. For an assignment $\theta : \text{vars}(Q) \mapsto \mathcal{U}$, $\mathcal{F}(\theta)$ is the weight $f_{\mathbf{u}_1}^1(\theta) \oplus \dots \oplus f_{\mathbf{u}_m}^m(\theta)$. \square

As an example, note that valuation functions built for $\oplus = '+'$ basically⁷ correspond to those arising in the classical setting of *weighted* CSPs, where combination of values in

⁶The fact that \mathcal{H}_a is a tree projection of \mathcal{H}_Φ w.r.t. $\mathcal{H}_{\mathcal{V}_2}$ witnesses that the treewidth of \mathcal{H}_Φ is 2. In general, a tree projection of \mathcal{H}_Φ w.r.t. $\mathcal{H}_{\mathcal{V}_k}$ exists if and only if \mathcal{H}_Φ has treewidth k at most (see [21, 35]).

⁷For more information on weighted CSPs, see Section 6.

the constraints come associated with a cost and the goal is to find a solution minimizing the sum of the costs over the constraints.

A constraint formula Φ equipped with a valuation function \mathcal{F} is called a (*constraint optimization formula*), and is denoted by $\Phi_{\mathcal{F}}$. For an optimization formula $\Phi_{\mathcal{F}}$ and a constraint database DB, we define the total order $\succeq_{\mathcal{F}}$ over the assignments in Φ^{DB} such that for each pair θ_1 and θ_2 in Φ^{DB} , $\theta_1 \succeq_{\mathcal{F}} \theta_2$ if and only if $\mathcal{F}(\theta_1) \geq \mathcal{F}(\theta_2)$.

For a set $O \subseteq \text{vars}(\Phi)$, we also define $\succeq_{\mathcal{F},O}$ as the total order over the assignments in $\Phi^{\text{DB}}[O]$ as follows. For each pair θ_1 and θ_2 of assignments in $\Phi^{\text{DB}}[O]$, $\theta_1 \succeq_{\mathcal{F},O} \theta_2$ if and only if there is an assignment $\theta'_1 \in \Phi^{\text{DB}}$ with $\theta_1 = \theta'_1[O]$ such that $\theta'_1 \succeq_{\mathcal{F}} \theta'_2$ holds, for each assignment $\theta'_2 \in \Phi^{\text{DB}}$ such that $\theta_2 = \theta'_2[O]$. Note that $\succeq_{\mathcal{F},O}$ reflects a descending order over real numbers. To have an ascending order, we can consider operators distributing over min (rather than over max), and define the order $\succeq_{\mathcal{F},O}^{\text{asc}}$ such that $\theta_1 \succeq_{\mathcal{F},O}^{\text{asc}} \theta_2$ if and only if $\theta_2 \succeq_{\mathcal{F},O} \theta_1$. Our results are presented by focusing, w.l.o.g., on $\succeq_{\mathcal{F},O}$ only.

Two problems that naturally arise with constraint optimization formulas are stated next. The two problems receive as input an optimization formula $\Phi_{\mathcal{F}}$, a set $O \subseteq \text{vars}(\Phi)$ of distinguished variables, and a constraint database DB:

MAX($\Phi_{\mathcal{F}}, O, \text{DB}$): Compute an assignment $\theta \in \Phi^{\text{DB}}[O]$ such that there is no assignment $\theta' \in \Phi^{\text{DB}}[O]$ with $\theta' \succ_{\mathcal{F},O} \theta$;⁸ Answer NO SOLUTION, if $\Phi^{\text{DB}} = \emptyset$.

TOP-K($\Phi_{\mathcal{F}}, O, \text{DB}$): Compute a list $(\theta_1, \dots, \theta_{K'})$ of distinct assignments from $\Phi^{\text{DB}}[O]$, where $K' = \min\{K, |\Phi^{\text{DB}}[O]|\}$, and where for each $j \in \{1, \dots, K'\}$, there is no assignment $\theta' \in \Phi^{\text{DB}}[O] \setminus \bigcup_{i=1}^{j-1} \{\theta_i\}$ with $\theta' \succ_{\mathcal{F},O} \theta_j$. Note that the parameter K is an additional input of the problem TOP-K and, as usual, is assumed to be given in binary notation. This means, in particular, that the answers to this problem can be exponentially many when compared to the input size.

3.1.1. Structured Valuation Functions

Our results are actually given in the more general setting of the *structured* valuation functions, where different binary operators can be used in the same constraint formula, and the order according to which the basic weight functions have to be processed is syntactically guided by the use of parentheses (in this case the evaluation order of operators does matter, in general).

Formally, a *structured* valuation function \mathcal{F} is either a weight function or an expression of the form $(\mathcal{F}_\ell \oplus_b \mathcal{F}_r)$, where \mathcal{F}_ℓ and \mathcal{F}_r are in turn structured valuation functions, and \oplus_b is a binary operator with the same properties as the operator in Definition 3.1.

Clearly enough, any structured valuation function built with one binary operator can be transparently viewed as a standard valuation function by just omitting the parenthesis. On the other hand, given a valuation function \mathcal{F} , we can easily build the set $\text{svf}(\mathcal{F})$ of all possible equivalent structured valuation functions, by just considering all possible legal ways of adding parenthesis to \mathcal{F} .

Working on the elements of $\text{svf}(\mathcal{F})$ appears to be easier in the algorithms we shall illustrate in the following sections and, accordingly, our presentation will be focused on structured valuation functions. We will discuss in Section 4.4 how to move from structured valuation functions to equivalent standard valuation functions.

⁸As usual, the fact that $\theta_1 \succeq \theta_2$ and $\theta_2 \not\succeq \theta_1$ hold is denoted by $\theta_1 \succ \theta_2$.

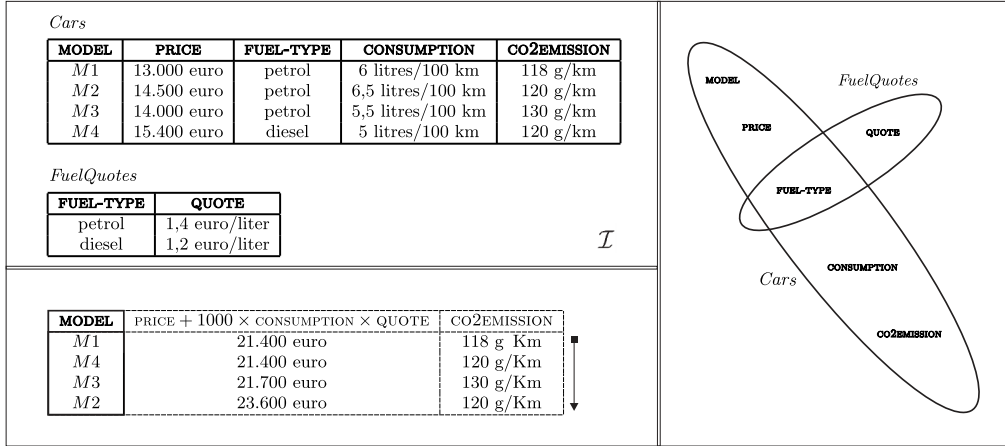


Figure 2: Illustration of Example 3.2.

Example 3.2. Consider a simple configuration scenario defined in terms of the CSP instance (Φ, DB) , where

$$\Phi = \text{Cars}(\text{MODEL}, \text{PRICE}, \text{FUEL-TYPE}, \text{CONSUMPTION}, \text{CO2EMISSION}) \wedge \text{FuelQuotes}(\text{FUEL-TYPE}, \text{QUOTE}),$$

and where the atoms in DB are those shown in Figure 2 using an intuitive graphical notation. Note that the instance is trivially satisfiable.

In fact, we are usually not interested in finding just *any* solution in this setting, but would rather like to single out one that matches as much as possible our preferences over the possible configurations. For instance, we might be interested in computing (the solution corresponding to) a car minimizing the sum of its price plus the cost that is expected to be paid, for the given quotation of the fuel, to cover 100.000 kilometers. Moreover, for cars that are equally ranked w.r.t. this first criterion, we might want to give preference to cars minimizing the emission of CO_2 . For a sufficiently large constant B (which can be treated in a symbolic way), this requirement can be modeled via the function $\mathcal{F} = ((B \times \mathcal{F}_1) + \mathcal{F}_2)$ such that

$$\begin{aligned} \mathcal{F}_1 &= (f_{(\text{PRICE})} + (1000 \times (f_{(\text{CONSUMPTION})} \times f_{(\text{QUOTE})}))) \\ \mathcal{F}_2 &= f_{(\text{CO2EMISSION})}, \end{aligned}$$

where $f_{(X)}(\theta) = \theta[X]$ is the identity weight function on each variable X and where any real number is viewed as a constant weight function. \triangleleft

3.2. Structured Valuation Functions and Embeddings

It is easily seen that structured valuation functions introduce further dependencies among the variables, which are not reflected in the basic hypergraph-based representation of the underlying CSP instances. Therefore, when looking at islands of tractability for constraint optimization formulas, this observation motivates the definition of a novel form of structural representation where the interplay between functions and constraint scopes is made explicit. In order to formalize this structural representation, we introduce the concept of *parse tree* of a structured valuation function.

Definition 3.3. Let \mathcal{F} be a structured valuation function. Then, the *parse tree* of \mathcal{F} is a labeled rooted tree $PT(\mathcal{F}) = (V, E, \tau)$, where τ maps vertices either to variables or to binary operators, defined inductively as follows:

- If \mathcal{F} is a weight function f , then $PT(\mathcal{F}) = (\{f\}, \emptyset, \tau_f)$ where $\tau_f(f) = \text{vars}(f)$. That is, $PT(\mathcal{F})$ has no edges and a unique (root) node f , labeled by the variables occurring in f .
- Assume that $\mathcal{F} = (\mathcal{F}_\ell \oplus \mathcal{F}_r)$ with $PT(\mathcal{F}_\ell) = (V_\ell, E_\ell, \tau_\ell)$ and $PT(\mathcal{F}_r) = (V_r, E_r, \tau_r)$, and with p_ℓ and p_r being the root nodes of $PT(\mathcal{F}_\ell)$ and $PT(\mathcal{F}_r)$, respectively. Then, $PT(\mathcal{F}) = (V_\ell \cup V_r \cup \{p\}, E_\ell \cup E_r \cup \{\{p, p_\ell\}, \{p, p_r\}\}, \tau_p)$ is rooted at a fresh node p , with the labeling function τ_p such that $\tau_p(p) = \oplus$ and its restrictions over V_ℓ and V_r coincide with τ_ℓ and τ_r , respectively.

Let O be a set of variables, and let p be the root of $PT(\mathcal{F})$. Then, the *output-aware parse tree* of \mathcal{F} w.r.t. O is the labeled tree $tree(\mathcal{F}, O) = (V \cup \{O\}, E \cup \{\{O, p\}\}, \tau_O)$ rooted at a fresh node O , and where τ_O is such that $\tau_O(O) = O$ and its restriction over V coincides with τ . \square

Now, we define the concept of embedding as a way to characterize how the parse tree of a structured function interacts with the constraints of an acyclic constraint formula.

Definition 3.4. Let \mathcal{F} be a structured valuation function for a constraint formula Φ , let \mathcal{H}_a be an acyclic hypergraph with $\text{vars}(\Phi) \subseteq \text{nodes}(\mathcal{H}_a)$, let $O \subseteq \text{vars}(\Phi)$ be a set of variables, and let $tree(\mathcal{F}, O) = (V_O, E_O, \tau_O)$ be the associated output-aware parse tree.

We say that the pair (\mathcal{F}, O) can be *embedded in* \mathcal{H}_a if there is a join tree $JT = (V, E, \chi)$ of \mathcal{H}_a and an injective mapping $\xi : V_O \mapsto V$, such that every vertex $p \in V_O$ is associated with a vertex $\xi(p)$ of JT , called *p-separator*, which satisfies the following conditions:

- (1) $\tau_O(p) \cap \text{nodes}(\mathcal{H}_a) \subseteq \chi(\xi(p))$, i.e., the variables occurring in p occur in the labeling of $\xi(p)$, too; and
- (2) there is no pair q, q' of vertices adjacent to p in $tree(\mathcal{F}, O)$ such that their images $\xi(q)$ and $\xi(q')$ are not separated by $\xi(p)$, i.e., such that they occur together in some connected component of the forest $(V \setminus \xi(p), \{e \in E \mid e \cap \xi(p) = \emptyset\})$.

The mapping ξ is called an *embedding of* (\mathcal{F}, O) *in* \mathcal{H}_a , and JT is its witness. \square

Intuitively, condition (1) states that any leaf node p of the parse tree (i.e., with $\tau_O(p) \cap \text{nodes}(\mathcal{H}_a) \neq \emptyset$) is mapped into a p -separator whose χ -labeling covers the variables involved in the domain of the underlying weight function. Moreover, it requires that the root node is mapped into a node whose χ -labeling covers the output variables in O . On the other hand, condition (2) guarantees that the structure of the parse tree is “preserved” by the embedding. This is explained by the example below.

Example 3.5. Recall the setting of Example 3.2 and the output-aware parse tree shown on the left of Figure 3. Observe that \mathcal{H}_Φ is acyclic, as it is witnessed by the join tree depicted on the right. Moreover, note that the figure actually shows that there is an embedding of $(\mathcal{F}, \{\text{MODEL}\})$ in \mathcal{H}_Φ that maps each node to (the hyperedge containing the variables of) the atom *Cars*, except for the leaf $f_{(\text{QUOTE})}$ which is mapped to *FuelQuotes*. Note also that the root is mapped to *Cars*, which indeed covers the output variable *MODEL*, and that mapping constant functions is immaterial. \triangleleft

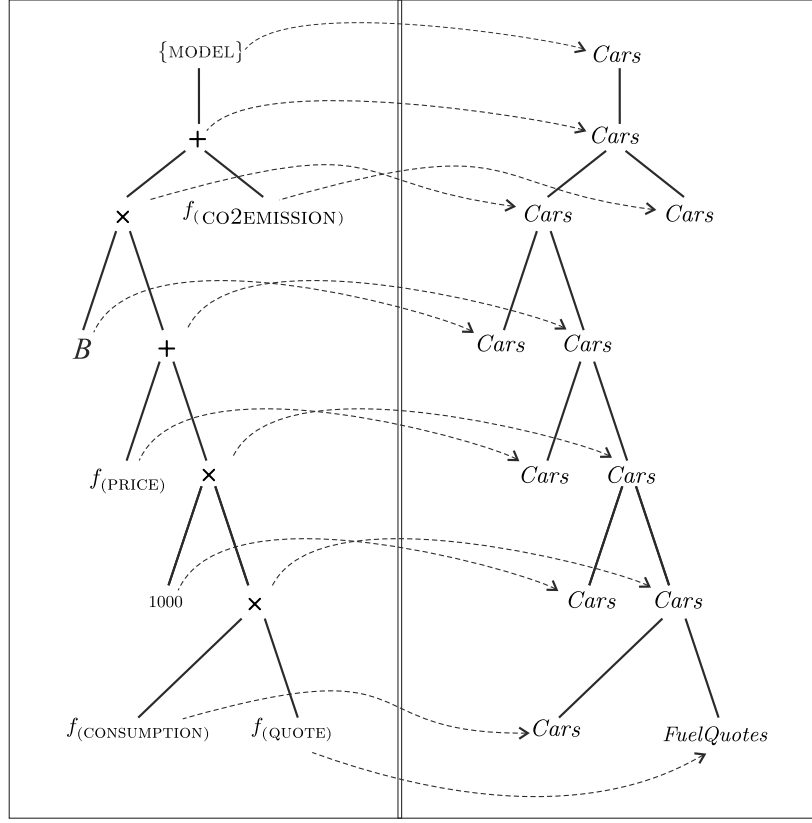


Figure 3: The join tree (right) and the parse tree (left) in the setting of Example 3.5.

3.3. Properties of Embeddings for Structured Valuation Functions

We shall now analyze some relevant properties of embeddings, which are useful for providing further intuitions on this notion and will be used in our subsequent explanations.

Let \mathcal{F} be a structured valuation function for a constraint formula Φ , let \mathcal{H}_a be an acyclic hypergraph with $\text{vars}(\Phi) \subseteq \text{nodes}(\mathcal{H}_a)$, and let $O \subseteq \text{vars}(\Phi)$ be a set of variables. Let $\xi : V_O \mapsto V$ be any injective mapping and denote by $V_\xi \subseteq V$ its image. Thus, for each vertex $v \in V_\xi$, its inverse $\xi^{-1}(v)$ is the vertex in the parse tree whose image under ξ is precisely v . In the following, let us view JT as a tree rooted at the vertex $\xi(O)$, where O is the root of $\text{tree}(\mathcal{F}, O)$. Moreover, in any rooted tree, we say that a vertex v is a descendant of v' , if either v is a child of v' , or v is a descendant of some child of v' .

Theorem 3.6. *Assume that ξ is an embedding of (\mathcal{F}, O) in \mathcal{H}_a , with $JT = (V, E, \chi)$ being its witness. Let v and v' be two distinct vertices in V_ξ . Then, v is a descendant of v' in JT if and only if $\xi^{-1}(v)$ is a descendant of $\xi^{-1}(v')$ in $\text{tree}(\mathcal{F}, O)$.*

Proof. We prove the property by structural induction, from the root to the leaves of JT . In the base case, v' is the root of JT and thus $\xi^{-1}(v')$ is the root of $\text{tree}(\mathcal{F}, O)$. In this case, the result is trivially seen to hold. Now assume that the property holds on any

vertex $v'' \in V_\xi$ in the path connecting the root and a vertex v' . That is, v is a descendant of v'' in JT if and only if $\xi^{-1}(v)$ is a descendant of $\xi^{-1}(v'')$ in $tree(\mathcal{F}, O)$. We show that the property holds on v' , too. To this end, we first claim the following.

Claim 3.7. *Let q_1, \dots, q_m be a path in $tree(\mathcal{F}, O)$ such that: (i) q_i is a child of q_{i-1} , for each $i \in \{2, \dots, m\}$; and (ii) $\xi(q_i)$ is a descendant in JT of $\xi(q_1)$, for each $i \in \{2, \dots, m-1\}$. Then, $\xi(q_i)$ is a descendant of $\xi(q_{i-1})$, for each $i \in \{2, \dots, m\}$.*

Proof. We prove the property by induction. Consider first the case where $i = 2$. The fact that $\xi(q_2)$ is a descendant of $\xi(q_1)$ is immediate by (ii). Then, assume that the property holds up to an index i , with $i \in \{2, \dots, m-1\}$. We show that it holds on $i+1$, too. Indeed, by inductive hypothesis, we know that $\xi(q_i)$ is a descendant of $\xi(q_{i-1})$, which is in turn a descendant of $\xi(q_1)$ by (ii). Consider the vertex q_i , and recall that since ξ is an embedding, $\xi(q_i)$ disconnects $\xi(q_{i+1})$ from $\xi(q_{i-1})$. This means that $\xi(q_{i+1})$ is a descendant of $\xi(q_i)$. \diamond

We now resume the main proof.

(only-if) Assume that v is a descendant of v' . Hence, v is a descendant of some vertex $v'' \in V_\xi$ and we can apply the inductive hypothesis to derive that $\xi^{-1}(v)$ and $\xi^{-1}(v')$ are both descendant of $\xi^{-1}(v'')$ in $tree(\mathcal{F}, O)$. Assume, for the sake of contradiction, that $\xi^{-1}(v)$ is not a descendant of $\xi^{-1}(v')$. We distinguish two cases.

In the first case $\xi^{-1}(v')$ is a descendant of $\xi^{-1}(v)$. This means that there is a path $\xi^{-1}(v'') = q_1, \dots, \xi^{-1}(v), \dots, q_m = \xi^{-1}(v')$. Note that on this path we can apply the inductive hypothesis in order to conclude that $\xi(q_i)$ is a descendant of $\xi(q_1)$, for each $i \in \{2, \dots, m\}$. Therefore, we are in the position to apply Claim 3.7, and we conclude that $\xi(q_i)$ is a descendant of $\xi(q_{i-1})$, for each $i \in \{2, \dots, m\}$. In particular, by transitivity, we get that $\xi(q_m)$ is a descendant of $\xi(\xi^{-1}(v))$. That is, v' is a descendant of v . Contradiction.

The only remaining possibility is that there are two distinct vertices q and q' that are children of a vertex \bar{q} , which is a descendant of $\xi^{-1}(v'')$ or is precisely $\xi^{-1}(v'')$, and such that $\xi^{-1}(v)$ (resp., $\xi^{-1}(v')$) is a descendant of q (resp., q') or coincides with q (resp., q') itself. Consider then the paths $\xi^{-1}(v''), \dots, \bar{q}, q, \dots, v$ and $\xi^{-1}(v''), \dots, \bar{q}, q', \dots, v'$. Similarly to the case discussed above, note that on each of them we can apply Claim 3.7. Thus, we get that $\xi(q)$ and $\xi(q')$ are descendant of $\xi(\bar{q})$. Moreover, either v is a descendant of $\xi(q)$, or $\xi(q)$ coincides with v . Similarly, either v' is a descendant of $\xi(q')$, or $\xi(q')$ coincides with v' . Finally, recall that ξ is an embedding, and hence $\xi(\bar{q})$ must disconnect $\xi(q)$ and $\xi(q')$. Therefore, it disconnects v and v' , too. Contradiction with the fact that v is a descendant of v' .

(if) Assume that $\xi^{-1}(v)$ is a descendant of $\xi^{-1}(v')$ in $tree(\mathcal{F}, O)$. Assume, for the sake of contradiction, that v is not a descendant of v' . Because of the only-if part, we are guaranteed that v' is in any case not a descendant of v . Therefore, there is a vertex \bar{v} disconnecting v and v' and such that v and v' are both descendant of \bar{v} . We can now apply the inductive hypothesis on the vertex $v'' \in V_\xi$ in the path connecting \bar{v} and the root, and that is the closest to \bar{v} (possibly coinciding with it). Therefore, we know that $\xi^{-1}(v)$ and $\xi^{-1}(v')$ are both descendant of $\xi^{-1}(v'')$. Hence, $\xi^{-1}(v')$ occurs in the path connecting $\xi^{-1}(v'')$ and $\xi^{-1}(v)$. Consider then the path

$\xi^{-1}(v'') = q_1, \dots, \xi^{-1}(v'), \dots, q_m = \xi^{-1}(v)$. Because of the inductive hypothesis, we know that $\xi(q_i)$ is a descendant of $\xi(q_1)$, for each $i \in \{2, \dots, m\}$. Therefore, we are in the position of applying Claim 3.7 and, by transitivity, we derive that $\xi(q_m)$ is a descendant of $\xi(\xi^{-1}(v'))$. That is, v is a descendant of v' . Contradiction. \square

In words, the above result tells us that embeddings preserve the descendant relationship. In fact, preserving this relationship suffices for an embedding to exist.

Theorem 3.8. *Let $\xi : V_O \mapsto V$ be an injective function satisfying condition (1) in Definition 3.4 for $\text{tree}(\mathcal{F}, O) = (V_O, E_O, \tau_O)$ and for a join tree $JT = (V, E, \chi)$ of \mathcal{H}_a . Assume that for each pair v, v' of distinct vertices in V_ξ , v is a descendant of v' in JT if and only if $\xi^{-1}(v)$ is a descendant of $\xi^{-1}(v')$ in $\text{tree}(\mathcal{F}, O)$. Then, there is an embedding ξ' of (\mathcal{F}, O) in \mathcal{H}_a (with witness JT), which can be built in polynomial time from ξ .*

Proof. Based on $\xi : V_O \mapsto V$, we build a function $\xi' : V_O \mapsto V$ as follows. Let x be any node in V_O . If x is a leaf node in $\text{tree}(\mathcal{F}, O)$ or it is the root, then we set $\xi'(x) = \xi(x)$. Otherwise, i.e., if x is an internal node with children y and z , then we first observe that, by hypothesis, $\xi(y)$ and $\xi(z)$ are both descendant of $\xi(x)$. Moreover, $\xi(y)$ (resp., $\xi(z)$) is not a descendant of $\xi(z)$ (resp., $\xi(y)$), and therefore there is a vertex \bar{v} in V possibly coinciding with $\xi(x)$ such that $\xi(y)$ and $\xi(z)$ occur in different components of $(V \setminus \bar{v}, \{e \in E \mid e \cap \bar{v} = \emptyset\})$ as descendants of \bar{v} —in particular, whenever $\bar{v} \neq \xi(x)$, we have that \bar{v} is a descendant of $\xi(x)$. For the node x , we now define $\xi'(x) = \bar{v}$.

Note that ξ' trivially satisfies condition (1) in Definition 3.4, as ξ' differs from ξ only over non-leaf nodes different from the root. We claim that ξ' satisfies condition (2), too.

Recall that, by Definition 3.3, the output-aware parse tree is binary, and its root is the only vertex having one child. Consider next any vertex $p \in V_O$ with parent s and children q and q' . Indeed, if p is the root or a leaf, then condition (2) in Definition 3.4 trivially holds. By the above construction (setting $x = p$), we know that $\xi(q)$ and $\xi(q')$ occur in different components of $(V \setminus \xi'(p), \{e \in E \mid e \cap \xi'(p) = \emptyset\})$ as descendants of $\xi'(p)$. Moreover, whenever $\xi'(q) \neq \xi(q)$, we are guaranteed that $\xi'(q)$ is a descendant of $\xi(q)$ (again by the above construction, this time setting $x = q$). Similarly, either $\xi'(q') = \xi(q')$, or $\xi'(q')$ is a descendant of $\xi(q')$. Hence, $\xi'(q)$ and $\xi'(q')$ occur in different components of $(V \setminus \xi'(p), \{e \in E \mid e \cap \xi'(p) = \emptyset\})$ as descendants of $\xi'(p)$.

Consider now the parent s and the child q —the same line of reasoning applies to the child q' . By hypothesis, we know that $\xi(p)$ occurs in the path connecting $\xi(s)$ and $\xi(q)$, with $\xi(q)$ being a descendant of $\xi(s)$. Moreover, by construction of ξ' (over s, p , and q), we have that: either $\xi'(s)$ occurs in the path connecting $\xi(s)$ and $\xi(p)$, or $\xi'(s) = \xi(s)$; either $\xi'(p)$ occurs in the path connecting $\xi(p)$ and $\xi(q)$, or $\xi'(p) = \xi(p)$; and $\xi'(q)$ is either a descendant of $\xi(q)$, or coincides with $\xi(q)$. Therefore, $\xi'(p)$ occurs in the path connecting $\xi'(s)$ and $\xi'(q)$. That is, $\xi'(s)$ and $\xi'(q)$ occur in different components of $(V \setminus \xi'(p), \{e \in E \mid e \cap \xi'(p) = \emptyset\})$.

By putting all together, we have shown that condition (2) in Definition 3.4 holds on any vertex $p \in V_O$. \square

4. Structural Tractability in the Tree Projection Setting

The concept of embedding has been introduced in Section 3 as a way to analyze the interactions of valuation functions with acyclic instances. However, the concept can be

easily coupled with the tree projections framework in order to be applied to instances that are not precisely acyclic. This coupling is formalized below.

Definition 4.1. Let \mathcal{F} be a structured valuation function for a constraint formula Φ , let $O \subseteq \text{vars}(\Phi)$ be a set of variables, and let \mathcal{V} be a view set for Φ . We say that (\mathcal{F}, O) can be *embedded in* (Φ, \mathcal{V}) if there is a sandwich formula Φ_a of Φ w.r.t. \mathcal{V} such that (\mathcal{F}, O) can be embedded in the acyclic hypergraph \mathcal{H}_{Φ_a} . If $O = \emptyset$, then we just say that \mathcal{F} can be embedded in (Φ, \mathcal{V}) . \square

Example 4.2. Recall the setting of Example 2.3 and the valuation function $\mathcal{F} = f_a$, where $f_a(\theta) = \theta(A)$, for each solution θ . It is immediate to check that \mathcal{F} can be embedded in (Φ, \mathcal{V}_2) . Indeed, consider the sandwich formula Φ_a depicted in Figure 1, and note that (\mathcal{F}, \emptyset) can be embedded in \mathcal{H}_{Φ_a} . In fact, as \mathcal{F} consists of a weight function only, this is witnessed by any join tree JT of \mathcal{H}_{Φ_a} because we can always build an embedding that maps f_a to a vertex q of JT such that $\chi(q) \supseteq \{A\}$. Note that, for this kind of valuation functions, checking the existence of an embedding always reduces to checking the existence of a tree projection. \triangleleft

Recall that deciding whether a pair of hypergraphs has a tree projection is an NP-complete problem [16], so that the notion of embedding can hardly be exploited in a constructive way when combined with tree projections. However, we show in this section that the knowledge of an embedding (and of a tree projection) is not necessary to compute the desired answers. Indeed, a *promise-free* algorithm can be exhibited that is capable of returning a solution to MAX (and TOP- K) or to check that the given instance is not embeddable. The algorithm is in fact rather elaborated, and we start by illustrating some useful properties that can help the intuition.

Hereinafter, let Φ be a constraint formula, DB a constraint database, O a set of variables, and \mathcal{F} a structured valuation function, all of them being provided as input to our reasoning problems. Moreover, to deal with the setting of tree projections, we assume that a view set \mathcal{V} for Φ plus a constraint database DB' that is legal for \mathcal{V} w.r.t. Φ and DB are provided. Accordingly, to emphasize the role played by these structures, the problems of interest will be denoted as MAX($\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}'$) and TOP- K ($\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}'$).

4.1. Useful Properties of Embeddings

For a constraint database DB and an atom a , the set a^{DB} will be also denoted by $\text{rel}(a, \text{DB})$. Substitutions in $\text{rel}(a, \text{DB})$ will be also viewed as the ground atoms in DB to which they are unambiguously associated. If q is an atom, $\Phi^{\text{DB}}[q]$ denotes $\Phi^{\text{DB}}[\text{vars}(q)]$.

Without loss of generality, assume that, for each weight function $f_i \in \text{wfs}(\mathcal{F})$, \mathcal{V} contains a *function view* w_{f_i} over the variables in $\text{vars}(f_i)$. Indeed, if $w_{f_i} \notin \mathcal{V}$, then we can just define w_{f_i} as the projection over $\text{vars}(f_i)$ of any view $w \in \mathcal{V}$ such that $\text{vars}(w) \supseteq \text{vars}(f_i)$. In particular, if such a view w does not exist, then we can immediately conclude that \mathcal{F} cannot be embedded in (Φ, \mathcal{V}) .

Let us define $\overline{\text{DB}}$ as the constraint database obtained (in polynomial time) by *enforcing pairwise consistency* on DB' w.r.t. \mathcal{V} [22]. The method consists of repeatedly applying, till a fixpoint is reached, the following constraint propagation procedure: Take any pair w and w' of views in \mathcal{V} , and delete from DB any (ground atom associated with an) assignment θ in $\text{rel}(w, \text{DB})$ for which no assignment $\theta' \in \text{rel}(w', \text{DB})[\text{vars}(w) \cap \text{vars}(w')]$ exists with $\theta' \subseteq \theta$. In words, the procedure removes, for each view w , all its associated

assignments θ that cannot be extended to some assignment in each of the remaining views. In the database terminology, this is called a *semijoin* operation over w and w' .

The crucial property enjoyed by the database $\overline{\text{DB}}$, which we shall intensively use in our elaborations, is recalled below.

Proposition 4.3 ([21]). *Assume there exists a tree projection \mathcal{H}_a of \mathcal{H}_Φ with respect to $\mathcal{H}_\mathcal{V}$. Then, $w^{\overline{\text{DB}}}[h] = \Phi^{\text{DB}}[h]$ holds, for every $w \in \mathcal{V}$ and $h \subseteq \text{vars}(w)$ such that there is a hyperedge h_a of \mathcal{H}_a with $h \subseteq h_a$.*

For any partial assignment $\theta : \mathcal{W} \mapsto \mathcal{U}$, where $\mathcal{W} \subseteq \text{vars}(\Phi)$, and for any constraint optimization formula $\Phi_{\mathcal{F}}$, denote by $\max_{\mathcal{F}}(\theta)$ the maximum weight that any assignment $\theta' \in \Phi^{\text{DB}}$ with $\theta'[\mathcal{W}] = \theta$ can get according to \mathcal{F} , that is, $\max_{\mathcal{F}}(\theta) = \max(\{\mathcal{F}(\theta') \mid \theta' \in \Phi^{\text{DB}} \text{ and } \theta'[\mathcal{W}] = \theta\} \cup \{\perp\})$, where \perp denotes the minimum weight in the codomain of the valuation function.

Let p be any vertex of $\text{tree}(\mathcal{F}, O)$ occurring in the parse tree $PT(\mathcal{F})$. Let \mathcal{F}_p denote the subexpression of \mathcal{F} whose parse tree is the subtree rooted at p . Note that if $\mathcal{F}_p = f_i$ holds for some weight function f_i , then $\forall \theta \in w^{\overline{\text{DB}}}_{f_i}, f_i(\theta) = \max_{\mathcal{F}_p}(\theta)$ holds by construction. Assume now that p is a (non-leaf) vertex labeled by \oplus , and let ℓ and r be its children. Then, the maximum weight of $\mathcal{F}_p = (\mathcal{F}_\ell \oplus \mathcal{F}_r)$ is bounded by the aggregation via \oplus of the maximum weights that can be achieved over its \mathcal{F}_ℓ and \mathcal{F}_r .

Lemma 4.4. $\max_{\mathcal{F}_\ell \oplus \mathcal{F}_r}(\theta) \leq \max_{\mathcal{F}_\ell}(\theta) \oplus \max_{\mathcal{F}_r}(\theta)$ holds for each partial assignment θ .

Proof. Let $\bar{\theta}$ be an assignment such that $\max_{\mathcal{F}_\ell \oplus \mathcal{F}_r}(\theta) = \mathcal{F}_\ell(\bar{\theta}) \oplus \mathcal{F}_r(\bar{\theta})$. Then, the result follows by the properties of \oplus and since $\mathcal{F}_\ell(\bar{\theta}) \leq \max_{\mathcal{F}_\ell}(\theta)$ and $\mathcal{F}_r(\bar{\theta}) \leq \max_{\mathcal{F}_r}(\theta)$. \square

Assume now that there exists a tree projection \mathcal{H}_a of \mathcal{H}_Φ with respect to $\mathcal{H}_\mathcal{V}$, and that the pair (\mathcal{F}, O) can be *embedded in* \mathcal{H}_a . Let $JT = (V, E, \chi)$ be the join tree of \mathcal{H}_a and $\xi : V_O \mapsto V$ be the injective mapping of Definition 3.4. Then, we show that the inequality in Lemma 4.4 is tight on separators, thus the operation of choosing the best sets of partial assignments computed in the subtrees distributes over \oplus .

Lemma 4.5. $\max_{\mathcal{F}_\ell \oplus \mathcal{F}_r}(\theta) = \max_{\mathcal{F}_\ell}(\theta) \oplus \max_{\mathcal{F}_r}(\theta)$ holds for each $\theta \in w^{\overline{\text{DB}}}[q_a]$, where q_a is a p -separator for some $p \in V_O$ and $w \in \mathcal{V}$ is a view with $\text{vars}(w) \supseteq \text{vars}(q_a)$.

Proof. Recall that $\xi(p)$ separates its children ℓ and r in the join tree JT of \mathcal{H}_{Φ_a} . Observe that, by Lemma 3.6, the images of all vertices of the subtree rooted at ℓ (resp., r) belong to the same connected component C_ℓ (resp., C_r) of $JT \setminus \xi(p)$. In particular, $C_\ell \neq C_r$. Thus, from the connectedness condition of join trees, every variable that C_ℓ and C_r have in common must be included in $\chi(\xi(p))$. In fact, any partial assignment $\theta \in w^{\overline{\text{DB}}}[q_a]$ provides a value for all these variables. Thus, all possible extensions of θ to C_ℓ are independent of their extensions to C_r , so that they can be freely combined. Hence, we can safely obtain $\max_{\mathcal{F}_\ell \oplus \mathcal{F}_r}(\theta)$ by computing \oplus over the maximum weights obtained for θ looking at \mathcal{F}_ℓ and \mathcal{F}_r in a separate way. \square

In the light of Lemma 4.4 and Lemma 4.5, it is not difficult to define a bottom-up algorithm that, given the tree projection \mathcal{H}_a and the embedding ξ , processes from the leaves to the root each vertex p of (\mathcal{F}, O) computing the maximum weights that can be achieved by combining the results coming from its children, and using some view covering the p -separator.

Input: A constraint formula Φ ; a structured valuation function \mathcal{F} ;
a view set \mathcal{V} for Φ ;
a constraint database DB' legal for \mathcal{V} w.r.t. Φ and DB ; and,
a set of variables $O \subseteq \text{vars}(\Phi)$;

Output: An assignment, NO SOLUTION, or FAIL;

begin
 $\text{DB}_1 := \text{EnforcePairwiseConsistency}(\mathcal{V}, \text{DB}')$;
if some database relation is empty **then Output** NO SOLUTION;
Let p_1, \dots, p_s be a topological ordering of the vertices of $\text{tree}(\mathcal{F}, O)$;
Initialize the sets of candidate separators $\text{sep}_1, \dots, \text{sep}_s$,
and let DB_1 be the resulting constraint database;
for $i:=1$ **to** $s-1$ **do**
 $\text{DB}'_i := \text{evaluate}(\text{sep}_i, \text{DB}_i)$;
if sep_i is empty **then Output** FAIL;
 $\text{DB}_{i+1} := \text{propagate}(\text{sep}_i, \text{DB}'_i)$
 $\text{DB}_{s+1} := \text{evaluate}(\text{sep}_s, \text{DB}_s)$;
if sep_s is empty **then Output** FAIL;
else
let $\text{rel} = [wX]^{\text{DB}_{s+1}}$, where $\text{sep}_s = \{[wX]\}$;
Output any assignment from $\{\theta \in \text{rel} \mid \theta[X] = \max\{\text{rel}[X]\}\}$;
end.

Figure 4: **Algorithm Compute-Max.**

However, because deciding whether there is a tree projection (and compute one, if one exists) is NP-hard [16], such a naïve approach to solve MAX and TOP- K is impractical for large constraint formulas. We need a method that is able to perform the computation even when an embedding is not given.

4.2. Algorithm Compute-Max

Our approach to solve MAX and TOP- K even without the knowledge of (a sandwich formula and of) an embedding is based on the Algorithm **Compute-Max** shown in Figure 4.

Assume that the vertices p_1, \dots, p_s of $\text{tree}(\mathcal{F}, O)$ are numbered according to some topological ordering of this tree (from leaves to root). As no tree projection and no embedding are known, we miss the relevant information about which views behave as separators. This is dealt with in **Compute-Max** by maintaining, for each vertex p_i , a set sep_i of views that are *candidates* to be p_i -separators in some tree projections and w.r.t. to some embedding. These sets are managed as follows.

Initialization. Define DB_1 as the database obtained by enforcing pairwise-consistency on DB' w.r.t. \mathcal{V} . Let p_i be a vertex of $\text{tree}(\mathcal{F}, O)$, and consider three cases:

LEAF NODE: p_i is a leaf associated with the weight function f_i . Then, sep_i contains only an “augmented” view $[w_{f_i} X_i^{(w_{f_i})}]$ over a fresh relation symbol, and over all variables in $\text{vars}(w_{f_i})$ plus the fresh variable $X_i^{(w_{f_i})}$. Accordingly, DB_1 is enlarged to contain

the relation:

$$rel([w_{f_i} X_i^{(w_{f_i})}], DB_1) = \{\theta \cup \{X^{(w_{f_i})}/f_i(\theta)\} \mid \theta \in w_{f_i}^{DB_1}\}.$$

Thus, the auxiliary variable $X_i^{(w_{f_i})}$ is meant to store the weight of the function $f_i = \mathcal{F}_{p_i}$ for each assignment of the view w_{f_i} . Note that the sample set sep_i includes just one (augmented) function view, as w_{f_i} can always be used as a p_i -separator.

INTERNAL NODE: p_i is a non-leaf vertex having two children named p_r and p_t in the given ordering. Then, for each $w \in \mathcal{V}$, $w' \in sep_r$, and $w'' \in sep_t$, the set sep_i includes the augmented view $[w X_r^{(w')} X_t^{(w'')}]$, over the variables in $vars(w)$ plus the fresh variables $X_r^{(w')}$ and $X_t^{(w')}$. Accordingly, DB_1 is enlarged to contain the relation:

$$rel([w X_r^{(w')} X_t^{(w'')}] , DB_1) = \{\theta \cup \{X_r^{(w')}/noval, X_t^{(w'')}/noval\} \mid \theta \in w^{DB_1}\}.$$

Intuitively, augmented views store the weights derived during the computation for functions \mathcal{F}_{p_r} and \mathcal{F}_{p_t} . Initially, we consider the constant *noval*, meaning that no weight is currently available. Note that for internal nodes we need to keep all the possible views in \mathcal{V} as candidates for being p_i -separators.

ROOT: $p_i = p_s$ is the root whose only child is p_{s-1} . Then, let us chose any view $w_O \in \mathcal{V}$ such that $O \subseteq vars(w_O)$, which exists for otherwise there would be no embedding. For each view $w \in sep_{s-1}$, the set sep_s includes the augmented view $[w_O X_{s-1}^{(w)}]$, whose relations in DB_1 are:

$$rel([w_O X_{s-1}^{(w)}], DB_1) = \{\theta \cup \{X_{s-1}^{(w)}/noval\} \mid \theta \in w_O^{DB_1}[O]\}.$$

During the initialization, DB_1 is modified so as to include augmented views. However, the projection of each augmented view over the variables occurring in Φ gives precisely the original underlying view. Thus, Lemma 4.4 and Lemma 4.5 hold over (the modified) DB_1 and the augmented views. During the computation, a sequence of such constraint databases DB_2, \dots, DB_s is constructed. For each of them, the equivalence with DB_1 when considering projections over the variables in the original views is guaranteed.

Main Loop. After their initialization, views in sep_i are incrementally processed, from $i = 1$ to $i = s$, via the functions *evaluate* and *propagate*. Both functions receive as input a candidate separator and a current constraint database, and produce as output a novel constraint database. The functions

Step $DB'_i := evaluate(sep_i, DB_i)$. The goal of this step is to evaluate functions over the candidates in sep_i and to filter out those that cannot be p_i -separators. When invoked according to the topological ordering, it will be guaranteed that the active domain in DB_i of any variable of any augmented view in sep_i does not include *noval*, because functions associated with the children have been previously evaluated. We distinguish three cases:

LEAF NODE: In this case, no operation is required, as sep_i contains one good augmented view, by initialization.

INTERNAL NODE: For every $w \in \mathcal{V}$, recall that sep_i contains the view $[wX_r^{(w')}X_t^{(w'')}]$, for each pair $w' \in sep_r$ and $w'' \in sep_t$. Let \oplus be the label of p_i , and for any assignment $\bar{\theta} \in w^{\text{DB}_1}$, let $marg(\bar{\theta}, w, w', w'')$ be the maximum of $\theta[X_r^{(w')}] \oplus \theta[X_t^{(w'')}]$ taken over all the assignments $\theta \in [wX_r^{(w')}X_t^{(w'')}]^{\text{DB}_i}$ such that $\bar{\theta} = \theta[w]$. This is often called marginalization of $[wX_r^{(w')}X_t^{(w'')}]$ w.r.t. $X_r^{(w')} \oplus X_t^{(w'')}$. Let $best(\bar{\theta}, w)$ denote the minimum weight of $marg(\bar{\theta}, w, w', w'')$ over all the possible pairs of views w' and w'' , and define for w the augmented view $[wX_i^{(w)}]$, whose associated relation in DB_i is:

$$rel([wX_i^{(w)}], \text{DB}_i) = \{\theta \cup \{X_i^{(w)} / best(\bar{\theta}, w)\} \mid \bar{\theta} \in w^{\text{DB}_1}\}.$$

Then, sep_i is modified by including only all augmented views of the form $[wX_i^{(w)}]$.

The rationale of this step can be understood by first recalling that every assignment in a p_i -separator is associated with the largest weight over its possible extensions to full answers according to \mathcal{F}_{p_i} (cf. Lemma 4.4 and Lemma 4.5). In fact, when analyzing the algorithm, we shall show that good candidates to act as p_i -separators are those having the minimum marginalized weights for each one of their assignments, which therefore motivates the definition of the term $best(\bar{\theta}, w)$. Based on this fact, we actually delete from sep_i every view whose maximum weight over all its assignments is not the minimum over the maximum weights of all other views. This way $|sep_i| \leq |\mathcal{V}|$, and the maximum weight stored somewhere in the constraint database is bounded by its real maximum over the answers of the given constraint formula. Therefore, no space explosion may occur, neither in terms of number of samples nor in terms of size of the weights stored in the views.

ROOT: We drop all views from sep_s , but one view (if any) of the form $[w_OX_{s-1}^{(w)}]$ such that for each $\theta \in [w_OX_{s-1}^{(w)}]^{\text{DB}_s}$, $\theta[X_{s-1}^{(w)}] \leq \theta'[X_{s-1}^{(w')}]$ holds over any $[w_OX_{s-1}^{(w')}] \in sep_s$, and any $\theta' \in [w_OX_{s-1}^{(w')}]^{\text{DB}_s}$ with $\theta'[w_O] = \theta[w_O]$.

Step $\text{DB}_{i+1} := propagate(sep_i, \text{DB}'_i)$. Let p_j be the parent of p_i . In this step, we propagate the information of the views in sep_i into sep_j . For any variable X , let $dom(X)$ denote its active domain in DB'_i . Then, for each view $[wX_i^{(w)}] \in sep_i$, propagation is implemented via the following steps (1)–(6):

- (1) initialize a set $\mathcal{V}_i = \{[wX_i^{(w)}]\}$;
- (2) add to \mathcal{V}_i all augmented views $[w_bX_i^{(w)}]$ for each $w_b \in \mathcal{V}$, and to DB'_i their corresponding relations of the form $rel([w_bX_i^{(w)}], \text{DB}'_i) = w_b^{\text{DB}'_i} \times dom(X_i^{(w)})$; that is, these views are not restrictive w.r.t. X_i because all its possible weights are considered;
- (3) add to \mathcal{V}_i all the views of the form $[w'X_i^{(w)}X_r^{(w'')}]$ which are stored in sep_j . Denote by R the projection of $rel([w'X_i^{(w)}X_r^{(w'')}]^{\text{DB}'_i})$ over all variables but $X_i^{(w)}$, and update $rel([w'X_i^{(w)}X_r^{(w'')}]^{\text{DB}'_i})$ to be the relation containing all assignments θ such that $\theta[w'X_r^{(w'')}] \in R$ and $\theta[X_i^{(w)}] \in dom(X_i^{(w)})$. Repeat the step for the symmetrical case of those views having the form $[w'X_r^{(w'')}]X_i^{(w)}$.

- (4) update DB'_{i+1} with the result of $EnforcePairWiseConsistency(\mathcal{V}_i, DB'_i)$;
- (5) remove from DB'_{i+1} the relations added at step (2).
- (6) replace each view $[w'X_i^{(w)}X_r^{(w'')}] \in sep_j$ by its marginalization w.r.t. $X_i^{(w)}$, that is, remove from $rel([w'X_i^{(w)}X_r^{(w'')}], DB'_{i+1})$ any assignment θ for which there is an assignment θ' in the same relation with $\theta[w'X_r^{(w'')}] = \theta'[w'X_r^{(w'')}]$ and $\theta[X_i^{(w)}] < \theta'[X_i^{(w)}]$. Repeat for all views of the form $[w'X_r^{(w'')}]X_i^{(w)}$.

Note that the goal of steps (1)–(4) above is to filter views of the form $[w'X_i^{(w)}X_r^{(w'')}]$ that are stored in sep_j , by keeping the assignments that agree with the weights stored in the view $[wX_i^{(w)}]$. This is done by enforcing local consistency via the augmented views $[w_bX_i^{(w)}]$ added at step (2). In particular, such augmented views (as well as the target view $[w'X_i^{(w)}X_r^{(w'')}]$) do not constrain the weights for the variable $X_i^{(w)}$, but just propagate the information in $[wX_i^{(w)}]$, as they are initialized by associating the whole active domain $dom(X_i^{(w)})$ with each assignment in the corresponding original views of \mathcal{V} . In particular, as their role is just to propagate the information from sample $[wX_i^{(w)}]$ to sample $[w'X_i^{(w)}X_r^{(w'')}]$, they are eventually removed in step (5) from the constraint database. Finally, note that the same assignment can be propagated with different associated weights. The final ingredient (used at the end of each “evaluate step”) is to retain the assignment with the minimum associated weight.

Concluding Step. If after the last invocation of the evaluation step sep_s contains one view, then output any of its assignments having the maximum associated weight.

4.3. “Depromisization” and Analysis Overview

The analysis of **Compute-Max** is rather technical, and its details are deferred to the Appendix. Observe that algorithm **Compute-Max** is a *promise* algorithm, in that it is guaranteed to correctly return a solution to MAX under the hypothesis that some embedding exists (the “promise”). Actually, we can show that its correctness does not require that the constraint optimization formula can be embedded in a tree projection of the entire CSP instance. Indeed, we can show that it suffices that an embedding exists for some *homomorphically equivalent* subformula.

Remark 4.6. We consider the usual computational setting where each mathematical operation costs 1 time unit. However, all the algorithms described in the following are such that the total size of the weights computed during their execution is polynomially bounded w.r.t. the combined size of the input and the size of the value of any optimal solution (assuming that the promise holds). This is a sensitive issue because, in the adopted computational setting, one may compute in polynomial-time weights of size exponential w.r.t. the input size.

To state our main result, recall first that, whenever Φ' is a subformula of Φ , i.e., $atoms(\Phi') \subseteq atoms(\Phi)$, we say that Φ' is *homomorphically equivalent* to Φ , denoted by $\Phi' \equiv_h \Phi$, if there is a homomorphism from Φ to Φ' , i.e., mapping $h : vars(Q) \mapsto vars(Q')$ such that for each $r_i(\mathbf{u}_i) \in atoms(Q)$, it holds that $r_i(h(\mathbf{u}_i)) \in atoms(Q')$. For any set O of variables, denote by $atom(O)$ a fresh atom over the variables in O .

Theorem 4.7. *Algorithm **Compute-Max** runs in polynomial time. It outputs NO SOLUTION, only if $\Phi^{\text{DB}} = \emptyset$. Moreover, it computes an answer (if any) to $\text{MAX}(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$, with \mathcal{F} being a structured valuation function, if (\mathcal{F}, O) can be embedded in (Φ', \mathcal{V}) for some subformula Φ' of $\Phi \wedge \text{atom}(O)$ such that $\Phi' \equiv_{\text{h}} \Phi \wedge \text{atom}(O)$. It outputs FAIL, only if this condition does not hold.*

Interestingly, **Compute-Max** can be used as a subroutine for an algorithm that incrementally builds a solution in Φ^{DB} , hence yielding a promise-free algorithm, i.e., an algorithm that either computes a correct solution or disproves some given promise (which is NP-hard to be checked), in our case the existence of an embedding. The full proof of the following result is given in the Appendix, but a proof idea is discussed below.

Theorem 4.8. *There is a polynomial-time algorithm for structured valuation functions that either solves $\text{MAX}(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$, or disproves that \mathcal{F} can be embedded in (Φ, \mathcal{V}) .*

Proof Idea. Given a variable $X \in \text{vars}(\Phi)$, we invoke **Compute-Max** with $O = \{X\}$ and with a modified set of views \mathcal{V}_X , which are obtained by augmenting each original view in \mathcal{V} with the variable X (and by modifying the original legal database DB' accordingly). Note that $(\mathcal{F}, \{X\})$ can be embedded in (Φ, \mathcal{V}_X) if and only if \mathcal{F} can be embedded in (Φ, \mathcal{V}) . By the application of Theorem 4.7 on the modified instance, if **Compute-Max** returns NO SOLUTION (resp., FAIL), then we can terminate the computation, by returning that there is no solution, i.e., $\Phi^{\text{DB}} = \emptyset$ (resp., the promise that \mathcal{F} can be embedded in (Φ, \mathcal{V}) is disproved—observe that the equivalent promise that $(\mathcal{F}, \{X\})$ can be embedded in (Φ, \mathcal{V}_X) is indeed more stringent than the one in Theorem 4.7 for \mathcal{V}_X and $O = \{X\}$). Therefore, let us assume that we get an assignment θ_X with an associated weight z . The variable X is then deleted from the constraint formula and from the views, and the original constraint database is modified so as to keep only assignments where X is fixed to θ_X .

The process is then iterated over all the variables. It can be shown that the promise is disproved if in some subsequent step **Compute-Max** does not return the same weight z , or if at the end of the computation the assignment we have computed, say θ , is not a solution or $\mathcal{F}(\theta) \neq z$. Otherwise, θ can be returned as a certified solution to MAX. \square

The above is the basis for getting the corresponding tractability result for TOP- K . Note that, since Φ^{DB} may have an exponential number of assignments, tractability of enumerating such assignments means here having algorithms that list them *with polynomial delay* (WPD): An algorithm M solves WPD a computation problem P if there is a polynomial $p(\cdot)$ such that, for every instance of P of size n , M discovers whether there are no solutions in time $O(p(n))$; otherwise, it outputs all desired solutions in such a way that a new solution is computed within $O(p(n))$ time after the previous one. Note that, in general, an algorithm running WPD may well use exponential time and space.

Note, moreover, that in the result below, when the algorithm discovers that the promise does not hold, i.e., when \mathcal{F} cannot be embedded in (Φ, \mathcal{V}) , then it stops the computation but we are still guaranteed that all solutions returned so far (which might be even exponentially many) constitute a solution to TOP- K' , for some $K' \leq K$. That is, the algorithm computes a (possibly empty) certified prefix of a solution to TOP- K . Again, the proof of the following result is elaborated in the Appendix.

Theorem 4.9. *There is a polynomial-delay algorithm for structured valuation functions that either solves $\text{TOP-}K(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$, or disproves that \mathcal{F} can be embedded in (Φ, \mathcal{V}) ; in the latter case, before terminating, it computes a (possibly empty) certified prefix of a solution.*

Proof Idea. The result can be established by exploiting a method proposed by Lawler [36] for ranking solutions to discrete optimization problems. In fact, the method has been already discussed in the context of inference in graphical models [3] and in conjunctive query evaluation [10]. Reformulated in the CSP context, for a CSP instance over n variables, the idea is to first compute the optimal solution (w.r.t. the functions specified by the user), and then recursively process n constraint databases, obtained as suitable variations of the database at hand where the current optimal solution is no longer a solution (and no relevant solution is missed). By computing the optimal solution over each of these new constraint databases, we get n candidate solutions that are progressively accumulated in a priority queue over which operations (e.g., retrieving any minimal element) take logarithmic time w.r.t. its size. Therefore, even when this structure stores data up to *exponential space* (so that its construction required overall exponential time), basic operations on it are still feasible in polynomial time. The procedure is repeated until K (or all) solutions are returned. Thus, whenever the MAX problem of computing the optimal solution is feasible in polynomial time (over the instances generated via this process), we can solve with polynomial delay the TOP- K problem of returning the best K -ranked ones. In fact, we can show that the constraint database can always be updated according to the approach by Lawler [36], while being still in the position of applying Theorem 4.8, and hence by iteratively solving TOP- K . \square

4.4. Results for Evaluation Functions

Our analysis has been conducted so far over *structured* valuation functions, whose syntactic form plays a crucial role with respect to the existence of an embedding in some tree projection. However, when we focus instead on valuation functions only (built over a single binary operator \oplus) the specific form of the constraint formula should not matter because \oplus is by definition a commutative and associative operator. Therefore, to deal properly with this setting, we adopt a more semantic approach in which the only sensitive issue is the existence of a tree projection. To formalize the result, recall that, for a valuation function \mathcal{F} , $\text{svf}(\mathcal{F})$ denotes the set of all equivalent structured valuation functions.

Theorem 4.10. *Let Φ be a constraint formula, let O be a set of variables, let \mathcal{F} be a valuation function, and let \mathcal{V} be a view set for Φ . Then, the following statements are equivalent:*

- (1) *There is a function $\mathcal{F}' \in \text{svf}(\mathcal{F})$ such that $(\mathcal{F}', \emptyset)$ can be embedded in (Φ, \mathcal{V}) ;*
- (2) *There is a tree projection \mathcal{H}_a of \mathcal{H}_{Φ} w.r.t. $\mathcal{H}_{\mathcal{V}}$ such that:*
 - (a) *there is a hyperedge $h \in \text{edges}(\mathcal{H}_a)$ with $h \supseteq O$;*
 - (b) *for each weight function $f \in \text{wfs}(\mathcal{F})$, there is a hyperedge $h_f \in \text{edges}(\mathcal{H}_a)$ such that $h_f \supseteq \text{vars}(f)$.*

Moreover whenever (2) holds and the tree projection \mathcal{H}_a is given, then a function \mathcal{F}' as in (1) can be built in polynomial time.

Proof. The fact that (1) \Rightarrow (2) is immediate by the definition of embedding. Therefore, let us focus on showing that (2) \Rightarrow (1) holds, too.

Assume that \mathcal{H}_a is a tree projection of \mathcal{H}_Φ w.r.t. $\mathcal{H}_\mathcal{V}$ satisfying the conditions stated in (2). Consider a join tree $JT = (V, E, \chi)$ of \mathcal{H}_a and let p_O be a vertex in V such that $\chi(p_O) \supseteq h$, which exists by (2).(a). Let us root (to simplify the exposition below) the tree JT at p_O . Recall that $wfs(\mathcal{F})$ is the set of all weight functions occurring in \mathcal{F} , and for each $f \in wfs(\mathcal{F})$, let $p_f \in V$ be any vertex such that $\chi(p_f) = h_f$, which exists by (2).(b). Based on JT , we build a novel join tree $\bar{JT} = (\bar{V}, \bar{E}, \bar{\chi})$ as follows: for each function $f \in wfs(\mathcal{F})$, we create a new node \bar{p}_f such that $\bar{\chi}(\bar{p}_f) = \chi(p_f)$ and we add this node in \bar{JT} as a child of p_f . Moreover, we create a new node \bar{p}_O whose only child is p_O and such that $\bar{\chi}(\bar{p}_O) = \chi(p_O)$. This new node will act as the root of \bar{JT} . All the other nodes, edges, and labeling remain the same as in JT . Finally, we process \bar{JT} in order to make it binary. To this end, if a vertex p in \bar{JT} has children c_1, \dots, c_n with $n > 2$, then we modify \bar{JT} by removing the edges connecting p and c_i , for each $i \in \{2, \dots, n\}$, by adding a novel vertex p' as a child of p and by appending c_2, \dots, c_n as children of p' . The label of p' is defined as the label of p , so that the connectedness condition still holds on the modified join tree. The transformation is repeated till \bar{JT} is made binary.

Given the join tree \bar{JT} , consider the following algorithm that recursively builds $\mathcal{F}(\bar{JT})$. Let p be the vertex that is the closest to the root of \bar{JT} and such that p has two children c_1 and c_2 and the subtrees rooted at them each contains a vertex of the form \bar{p}_f , for some weight function $f \in wfs(\mathcal{F})$. Note that, since \bar{JT} is binary, either the vertex p is univocally determined or it does not exist at all. In particular, in this latter case, let f be the only weight function such that \bar{p}_f occurs in \bar{JT} (w.l.o.g., the function contains at least one weight function, and this will be recursively guaranteed) and define $\mathcal{F}(\bar{JT}) := f$. In the former case, define $\mathcal{F}(\bar{JT}) := \mathcal{F}(\bar{JT}_1) \oplus \mathcal{F}(\bar{JT}_2)$, where \bar{JT}_1 and \bar{JT}_2 are the trees rooted at c_1 and c_2 , respectively.

Note that $\mathcal{F}(\bar{JT})$ clearly belongs to $suf(\mathcal{F})$. Moreover, consider the function ξ such that $\xi(O) = \bar{p}_O$; $\xi(f) = \bar{p}_f$, for each $f \in wfs(\mathcal{F})$; and, for each internal node v of the parse tree, $\xi(v)$ is mapped to the node p selected in the above algorithm when processing the subexpression corresponding to the subtree rooted at v . Note that ξ is injective, and by the recursive construction, for each pair v, v' of distinct vertices in the image of ξ , v is a descendant of v' in \bar{JT} if and only if $\xi^{-1}(v)$ is a descendant of $\xi^{-1}(v')$ in $tree(\mathcal{F}(\bar{JT}), O)$. Then, we can apply Theorem 3.8 and conclude that an embedding ξ' of $(\mathcal{F}(\bar{JT}), O)$ in (Φ, \mathcal{V}) can be built in polynomial time from ξ . \square

Note that, since the “(2) \Rightarrow (1)” of the above result is constructive, we immediately get the following by Theorem 4.8 and Theorem 4.9.

Corollary 4.11. *Whenever a tree projection of \mathcal{H}_Φ w.r.t. $\mathcal{H}_\mathcal{V}$ is given, the problems $\text{MAX}(\Phi_\mathcal{F}, O, \text{DB}, \mathcal{V}, \text{DB}')$ and $\text{TOP-K}(\Phi_\mathcal{F}, O, \text{DB}, \mathcal{V}, \text{DB}')$ are tractable on classes of constraint optimization formulas $\Phi_\mathcal{F}$ where \mathcal{F} is any valuation function with $|vars(f)| = 1$, for each $f \in wfs(\mathcal{F})$.*

Proof. Assume that $|vars(f)| = 1$, for each $f \in wfs(\mathcal{F})$. By Theorem 4.10, we can use the given tree projection to build in polynomial time a function $\mathcal{F}' \in ef(\mathcal{F})$ such that $(\mathcal{F}', \emptyset)$ can be embedded in (Φ, \mathcal{V}) . The result follows by Theorem 4.8 and Theorem 4.9. \square

Interestingly, if a tree projection is not given, then we are still able to end up with a useful result. Indeed, we can provide a fixed-parameter polynomial-time algorithm, where *the parameter is the size of the valuation function*, measured as the number of occurrences of weight functions. This algorithm may be useful in those applications where the number of weight functions is small, while the number of constraints is large (as it is often the case in CSPs). Note that this fixed-parameter tractability result should not be confused with tractability results where the parameter is the size of the whole constraint formula (equivalently, the size of the hypergraph to be decomposed), which may be useful only when the instance consists of a few constraints only. In such cases, however, the results presented in this paper are not needed, because if the parameter is the size of the constraint formula, then one can compute in fixed-parameter polynomial-time a tree projection [21], and then use known techniques for computing optimal solutions on acyclic instances.

Theorem 4.12. *Consider the problem $\text{MAX}(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$ over valuation functions and parameterized by the size of such valuation functions. Then, there is a fixed-parameter polynomial-time algorithm that either solves the problem, or disproves that there exists some $\mathcal{F}' \in \text{svf}(\mathcal{F})$ that can be embedded in (Φ, \mathcal{V}) .*

Proof. Consider the following algorithm: For any $\mathcal{F}' \in \text{svf}(\mathcal{F})$, call the algorithm of Theorem 4.8. As soon as some invocation does not disprove the promise that \mathcal{F}' can be embedded in (Φ, \mathcal{V}) , then we can return the answer we have obtained. To conclude, observe that we perform at most $|\text{svf}(\mathcal{F})|$ iterations, and that $|\text{svf}(\mathcal{F})|$ depends only on the number of weight functions occurring in \mathcal{F} . \square

From the above theorem, we obtain the corresponding tractability result for TOP- K , as in the proof of Theorem 4.9. In particular, because we may ask for an exponential number of solutions, fixed-parameter tractability means here having a promise-free algorithm that computes the desired output *with fixed-parameter polynomial-delay*. The first solution is computed in fixed-parameter polynomial-time, and any other solution is computed within fixed-parameter polynomial-time from the previous one.

Theorem 4.13. *Consider the problem $\text{TOP-}K(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$ over valuation functions and parameterized by the size of such valuation functions. Then, there is a fixed-parameter polynomial-delay algorithm that either solves the problem, or disproves that there exists some $\mathcal{F}' \in \text{svf}(\mathcal{F})$ that can be embedded in (Φ, \mathcal{V}) ; in the latter case, before terminating, it computes a (possibly empty) certified prefix of a solution.*

5. Parallel Algorithms

In the previous sections we have seen that the desired best solutions can be computed by just enforcing local consistency, without any explicit computation of the equivalent sandwich acyclic instance associated with some tree projection of the given CSP. However, in most practical applications, in particular when constraint relations contain a large number of tuples of allowed values (with respect to the number of constraints), having such an acyclic instance allows us to solve the given instance much more efficiently.

In particular, without using acyclicity, we need a quadratic number of operations for enforcing local consistency between each pair of constraints, contrasted with a linear

number of such operations if the procedure is guided by any join tree of the given instance. From a computational complexity point of view, [37] proved that even establishing arc-consistency is P-complete and hence not parallelizable, while [38] showed that evaluating Boolean acyclic instances is LOGCFL-complete, hence inside P. Combining the latter result with the techniques of [39], it can be seen easily that even establishing global consistency in acyclic instances is in (functional) LOGCFL. It is known that all problems in this class are highly parallelizable. Indeed, any problem in LOGCFL is solvable in logarithmic time by a concurrent-read concurrent-write *parallel random access machine* (CRCW PRAM) with a polynomial number of processors, or in \log^2 -time by an exclusive-read exclusive-write (EREW) PRAM with a polynomial number of processors.

In this section we provide parallel algorithms for the computation of the best solutions over a set of output variables O of a given constraint optimization formula $\Phi'_{\mathcal{F}}$ over a constraint database DB' , where \mathcal{F} is a valuation function. These algorithms are significant extensions of a parallel algorithm originally given in [40].

We assume that a tree projection \mathcal{H}_a of (Φ', \mathcal{V}) (for some set of subproblems \mathcal{V}) is given in input, too. By the results in [41], we assume w.l.o.g. that the number of hyperedges of \mathcal{H}_a is at most the number of variables occurring in Φ' , so that \mathcal{H}_a cannot be much larger than the original hypergraph of Φ' . Because the valuation function is built with a single operator, say \oplus , the embedding problem is trivial and any tree projection, in particular \mathcal{H}_a , can be used to solve the optimization problem. Therefore, a sequential algorithm can be obtained easily by using Lemma 4.5 and a dynamic programming algorithm as in [11]. However, it is not a-priori clear how to compute such solutions in parallel with a guaranteed performance that is independent of the shape of the tree projection at hand, and this is precisely the goal of this section. The algorithms proposed in this section generalize the DB-SHUNT parallel algorithm for evaluating Boolean conjunctive queries to relational databases presented in [38], which is in its turn based on a *tree contraction* technique which closely resembles the method of Karp and Ramachandran for the *expression evaluation problem* [42].

We refer the interested reader to [43], for a nice review of the literature on parallel approaches to solving CSPs. In particular, the importance of enforcing consistency (at different levels) in CSP solvers is pointed out. There are different models of parallelism, some more suited to distributed computations (useful when nodes are associated with difficult subproblems), and others to specialized parallel machines (see, e.g., Valiant's bulk synchronous parallel (BSP) and multi-BSP computational models [44], which can simulate the PRAM model, or the Immediate Concurrent Execution (ICE) abstraction, described in [45] for a general-purpose many-core explicit multi-threaded (XMT) computer architecture).

In this paper, we abstract from low-level details of the model and, following [38], we assume a shared-memory parallel EREW DB-machine, where relational algebra operations are machine primitives. In one parallel step of the DB-machine, each machine's processor can perform a constant number of relational algebra operations on the database relations at hand. Transformations of constraint scopes and input different from constraint relations are considered costless as long as they are polynomial.⁹ The efficiency

⁹In fact, such further operations occurring in the proposed algorithms are feasible in logarithmic space and thus are parallelizable, in their turn.

of a computation of the machine will be measured according to the following cost parameters: (a) the number of parallel steps; (b) the number of relational processors employed in each step, i.e., processors able to perform operations of relational algebra and related operations such as relational assignment statements (e.g., $r := s$ where r and s are relation variables); (c) the size of the working constraint database. We refer the interested reader to [46] for a connection of our approach (called ACQ there) with Valiant’s BSP and in particular to Map-Reduce models.

5.1. Enforcing Global Consistency in Parallel Given a Tree Projection

We first focus on the problem, of independent interest, of computing the (maximal) globally consistent sub-instance of the given constraint optimization formula. Recall that a CSP instance \mathcal{I} with m constraints is said *globally consistent*, or equivalently *m -wise consistent*, or $R(*, m)$ C-consistent, if, for every constraint C occurring in \mathcal{I} , every tuple t of values in its constraint relation can be extended to a full solution of \mathcal{I} . More formally, there exists a satisfying assignment θ for \mathcal{I} such that t is given by θ applied (or restricted) to the variables in the scope of C .

First observe that, by using the given tree projection \mathcal{H}_a and the DB-SHUNT algorithm described in [38], we can compute easily with a logarithmic number of parallel steps an equivalent acyclic instance having the same solutions of the original one. Just consider every hyperedge h of \mathcal{H}_a and the constraints having scopes $s \subseteq h$ as a single acyclic instance (think of a join tree with h as its root label and all such constraints as direct children). We thus assume in the following that such an acyclic instance $\mathcal{I} = (\Phi, O, \text{DB})$ equivalent to the original instance has been already computed from Φ' , with Φ being an acyclic constraint formula, and DB its constraint database. We will omit the specification of the output variables O if it is understood that we are interested in solutions over all variables. W.l.o.g., every atom occurring in Φ is constant-free and does not contain any pair of variables with the same name.

Moreover, for an instance \mathcal{I} as above, the algorithms use an additional tree T , called an *e-Join Tree* for \mathcal{I} , defined as follows: each vertex p of T is a constraint occurring in \mathcal{I} , whose scope and constraint relation are denoted, respectively, by $\text{Scope}_T(p)$ and $\text{rel}(p)$; each constraint in Φ occurs as some vertex in T ; for each pair of vertices p, p' in T , the variables in $\text{Scope}_T(p) \cap \text{Scope}_T(p')$ occur in the scopes of all vertices in the path connecting p and p' in T . Moreover, the variables of $\text{Scope}_T(p)$ are partitioned in two distinguished sets $R_T(p)$ and $I_T(p)$.

Note that the above tree is essentially a join tree of the given acyclic instance. By using known results on join trees [38], it easily follows that, given \mathcal{I} , we can compute in linear time or in logspace (and hence in parallel) an e-Join Tree T for \mathcal{I} , whose number of vertices is linear in the number of constraints occurring in \mathcal{I} , such that: T is a strictly-binary tree, i.e., every non-leaf vertex has precisely two children; and for every vertex p , $I_T(p) = \emptyset$ (hence, $R_T(p)$ is equal to the original constraint scope occurring in \mathcal{I}).

We assume that such a computation has already been done and we next focus on the evaluation of the CSP instance \mathcal{I} with such an e-Join Tree T at hands. We remark that, from the resulting e-Join Tree T , we can immediately get, with at most one additional parallel step, the desired globally consistent constraint occurring in the original (possibly non acyclic) instance (Φ', DB') .

We next describe an algorithm that computes with (at most) a logarithmic number of parallel relational operations the (unique) maximal sub-instance of \mathcal{I} that is globally

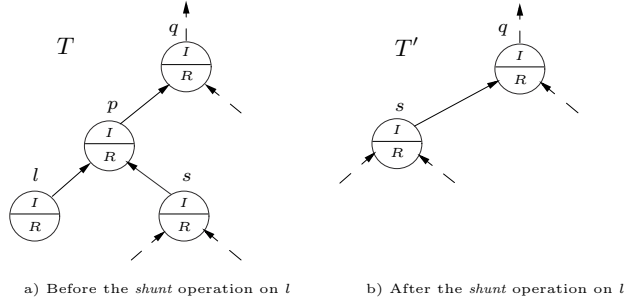


Figure 5: *Shunt* operation applied to leaf l

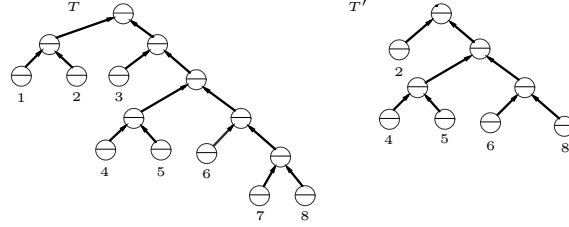


Figure 6: Shunt operation applied to leaves 1,3 and 7 in parallel

consistent, which is hereafter called the *globally-consistent reduct* of \mathcal{I} . Note that the naive parallel algorithm uses a linear number of parallel operations that depends on the tree-shape and that, in general, there is no guarantee that any balanced or similar good-shaped join tree exists for the given instance. Think, e.g., of a given CSP instance whose associated hypergraph is just a line.

Our algorithm transforms the e-join tree in stages, in such a way that the n -vertices tree T is contracted into a 3-vertices one in $\lceil \log n \rceil - 1$ stages. At each stage, a local operation, called *shunt*, is applied in parallel to half of the leaves of T . Let l be a leaf of an e-join tree T , p the parent of l , s the other child of p , and q the parent of p (see Figure 5). The shunt operation applied to l results in a new contracted tree T' in which l and p are deleted, s is suitably transformed into a fresh constraint and takes the place of vertex p (i.e., it becomes child of q). Intuitively, as p is deleted, the variables occurring in both p and q must be kept in the new e-join tree, in order to guarantee the soundness of the procedure. To this end, whenever such variables do not occur in s , they are added to the scope of s (precisely, in $I_T(s)$). This way, after the application of shunt, by means of $I_T(s)$, the new constraint at s stores the “witnesses” of the constraint tuples from p that are consistent with both s and l , and that are relevant to extend solutions to q (and up in the tree).

The leaves of T are numbered from left to right. At each iteration, the shunt operation is applied to the odd numbered leaves of T in a parallel fashion; to avoid concurrent changes on the same constraint, left and right leaves are processed in two distinct steps (Figure 6). Thus, after each iteration the number of leaves is halved, and the tree-contraction ends within the desired bound.

To compute the globally-consistent reduct in parallel, we employ a two-phase tech-

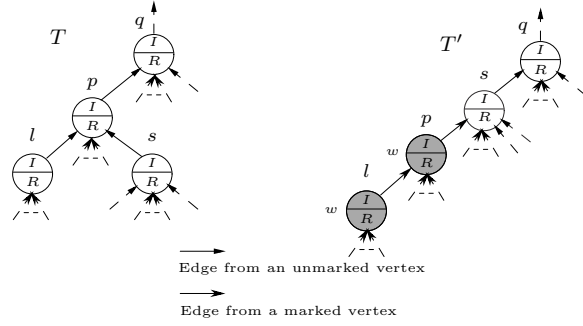


Figure 7: Execution of the $shunt(w)$ operation on l

nique. In the ascending phase, a shunt based procedure contracts the e-join tree; in the descending phase, the original tree shape is rebuilt by applying a sort of reverse shunt operation.

In the ascending phase, unlike the classical tree-contraction algorithm [38], no vertex is deleted but all processed vertices are modified and marked, for a subsequent use in the descending phase. A marked vertex is logically deleted for the ascending procedure and no shunt operation can be applied on it any more (during this phase). The mark is just a natural number encoding the step of the procedure that marked that vertex.

In our algorithm, a shunt operation performed at step w on the (current) e-join tree T , denoted $shunt(w)$, proceeds as follows. We say that a leaf of T is unmarked (at step w) if it is an unmarked vertex having no unmarked child (hence, such a vertex it is not an actual leaf of the current tree, though it is a leaf of the sub-tree of T induced by its unmarked vertices). Let l be an unmarked leaf of T , p the parent of l , s the other unmarked child of p , and q the parent of p (as in Figure 7). The shunt operation applied to l results in a new e-join tree T' in which l and p are marked with w , s takes the place of vertex p (i.e., it becomes a child of q in T'), and p becomes a child of s . The scopes of p and l remain unchanged in T' ; while the scope of s is transformed as follows:

$$R_{T'}(s) = R_T(s) \quad I_{T'}(s) = (R_T(q) \cap Scope_T(p)) \setminus R_T(s)$$

The constraint relations of p , s , and l in T' are just the projections over their scopes of the solutions of the subproblem comprising the constraints l , p , and s , i.e., in the relational framework, the projection of the relation $C = rel(l) \bowtie (rel(p) \bowtie rel(q)) \bowtie rel(s)$.

Additionally, the shunt operation produces and stores an updated version of vertex s of T , named s_w , that will be used in the descending phase to rebuild possible p - s relationships coming from attributes in $I_T(s)$. This new constraint s_w is stored into an additional storage (it is not in the e-join tree), its scope is $Scope_{T'}(s_w) = Scope_T(s)$ (the same as the old s) and its constraint relation $rel(s_w)$ is the projection over this scope of the same relation C .

As soon as the ascending phase is terminated and the tree is contracted to a tree of depth 1, a descending phase gets started which re-expands the e-join tree by applying in parallel a reverse shunt operation.

The *reverse shunt* (r -*shunt*) operation unmarks (and updates) the vertices having the highest mark, say, w in the e-join tree. Let l and p be two vertices with mark w of an

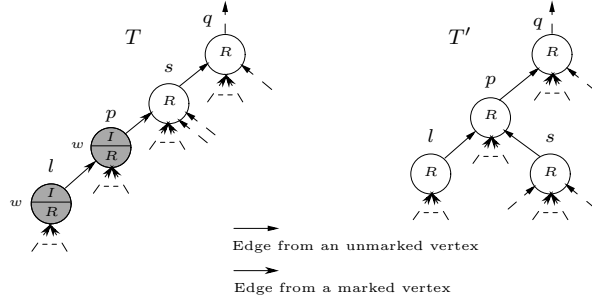


Figure 8: Execution of the $r\text{-shunt}(w)$ operation on vertices l and p

e-join tree T , such that p is the parent of l in T (note that l and p have been necessarily marked at the same step of the ascending procedure). Moreover, let s be the (unmarked) parent of p , and q , in turn, the (unmarked) parent of s (see Figure 8). The $r\text{-shunt}(w)$ operation applied on l and p results in a new e-join tree T' in which the marks of l and p have been removed, p takes the place of s as a child of q , and s becomes a child of p . The scopes and the constraint relations associated to the vertices p and l in T' are the following:

$$\begin{aligned}
 R_{T'}(p) &= R_T(p); & I_{T'}(p) &= \emptyset; & (Scope_{T'}(p) &= R_T(p);) \\
 R_{T'}(l) &= R_T(l); & I_{T'}(l) &= \emptyset; & (Scope_{T'}(l) &= R_T(l);) \\
 Rel_{T'}(p) &= \prod_{R_T(p)} (rel(q) \times rel(p) \times (Rel(s_w) \times rel(s))) \\
 Rel_{T'}(l) &= \prod_{R_T(l)} (Rel_{T'}(p) \times rel(l))
 \end{aligned}$$

Intuitively, the new constraint for p is computed from its “frozen” version (which was marked at step w of the ascending phase) by enforcing consistency with both q and s . Actually, concerning s , p is made consistent with the constraint relation $rel(s_w) \times rel(s)$, because s_w stores the variables shared by p and s in the ascending phase that were in $I_T(s)$ and are not present in the current scope of s . The relation for l is then obtained by enforcing consistency with the new p relation (note that the scope of p contains all variables that l shares with other unmarked vertices of the e-join tree).

The GLOBAL-CONSISTENCY algorithm is shown in Figure 9. The algorithm consists of two main phases. The ascending phase is similar to DB-SHUNT [38]: at each stage of the **while** loop performed in this phase, a shunt operation is applied to half of the unmarked leaves which get marked. After $\lceil \log n \rceil - 1$ **while** iterations only three vertices remain unmarked. Then, instruction (5) computes the relation associated to the root of the tree, which is the final relation for this vertex and will be returned in the output database. Instruction (6) begins the descending phase, by fixing the values for the relations associated to the two children of the root. The descending phase “propagates” the correct relation values from the unmarked vertices to the marked vertices. The **for** loop unmarks the vertices marked during the ascending phase. In particular, an execution of instruction (7.a) unmarks all vertices marked by an execution of either instruction (4.b) or (4.d) of the ascending phase. Once a vertex is unmarked, its relation is fixed to

Input: An acyclic CSP instance (Φ, DB) and an e-Join Tree T .
Output: The globally-consistent reduct of (Φ, DB) .

begin

- (1) Let λ be the number of leaves in T ;
- (2) Label the leaves of T in order from left to right as $1, \dots, \lambda$;
- (3) $\text{mark} := 0$;
 (* Ascending Phase *)
- (4) **while** $\text{depth}(T) > 1$ **do**
 - (a) $\text{mark} := \text{mark} + 1$;
 - (b) **in parallel** apply the $\text{shunt}(\text{mark})$ operation to all unmarked odd leaves that are the left children of their parent, and that have depth greater than 1;
 - (c) $\text{mark} := \text{mark} + 1$;
 - (d) **in parallel** apply the $\text{shunt}(\text{mark})$ operation to all unmarked odd leaves that are the right children of their parent, and that have depth greater than 1;
 - (e) shift out the rightmost bit in the labels of all remaining unmarked leaves;**end** (* while *)
 Let p be the root of T , and let p' and p'' be the children of p in T
- (5) $\text{rel}(p) := \text{rel}(p') \bowtie \text{rel}(p) \bowtie \text{rel}(p'')$;
 (* Descending Phase *)
- (6) $\text{rel}(p') := \prod_{R_T(p')} (\text{rel}(p) \bowtie \text{rel}(p'))$; $I_T(p') := \emptyset$;
 $\text{rel}(p'') := \prod_{R_T(p'')} (\text{rel}(p) \bowtie \text{rel}(p''))$; $I_T(p'') := \emptyset$;
- (7) **for** $w := \text{mark}$ **downto** 1 **do**
 - (a) **in parallel** apply $r\text{-shunt}(w)$ to all pairs of adjacent vertices with mark w**end** (* for *)
- (8) **output** all constraints stored in the vertices of T .

end.

Figure 9: GLOBAL-CONSISTENCY ALGORITHM.

the final value. Upon unmarking, the vertex is located exactly in the same tree position it had in the ascending phase (at the time when it was marked). Unmarking is performed by the r -shunt operations; each of them applies to a vertex l and to its parent p , both having the highest mark w in the current e-join tree. The relation for p is computed by eliminating from its old value all tuples which do not agree to its (unmarked) adjacent vertices. The relation for l is then obtained by a semi-join to its parent p . At the end of the **for** loop all vertices are unmarked and the associated constraints are returned.

Given an e-join tree T , we denote by $\text{verts}(T)$ the set of the vertices of T , and by $\text{verts}(T, s)$ the subtree rooted at vertex s of T . Moreover, define $M\text{verts}(T, s)$ as the set of vertices in the subtrees of T rooted at the marked children of s . Given a subproblem of the given CSP instance encoded as a set of vertices/constraints V of the e-join tree T , we will denote by $\bowtie V$ (or just V , if no confusion arises) the set of all its satisfying assignments, obtained as the (natural) join of all the constraint relations occurring in V .

Theorem 5.1. *The GLOBAL-CONSISTENCY algorithm is correct.*

Proof. Given an acyclic CSP instance \mathcal{I} , we denote by $cr(\mathcal{I})$ the globally-consistent reduct of \mathcal{I} . Moreover, given a constraint p , $cr(p, \mathcal{I})$ is the constraint occurring in the globally-consistent reduct, also called the consistent reduct of p w.r.t. \mathcal{I} . Importantly, note that, by construction, the e-join trees keep the peculiar connectedness property of ordinary join trees during the computation. By the same arguments used for proving the soundness of DB-shunt [38], it turns out that at the end of the ascending phase the constraint associated with the root r of the e-join tree T is precisely $cr(r, \mathcal{I})$.

Observe that, at each step of the computation, $cr(p, \mathcal{I})$ is equal to the projection on $R_T(p)$ of the join of the constraint relations of all (marked and unmarked) vertices of the e-join tree at hand, i.e. $\bowtie \text{verts}(T)$. Indeed, this property clearly holds at the beginning of the computation by definition of globally-consistent reduct, and it is easy to verify that the join of all constraint relations corresponding to the nodes of the e-join trees is an invariant of the algorithm. In fact, a tuple is discarded from a constraint relation at some step *only if* it is inconsistent with some other constraint.

By using an inductive argument we prove that, at any step of the descending phase, for each unmarked vertex p of the current e-join tree T , its constraint relation is equal to $cr(p, \mathcal{I})$. (*Basis.*) We already pointed out that, at the end of the ascending phase, the root r holds $cr(r, \mathcal{I})$. Moreover, it can be shown that, after the execution of instruction (6), $rel(p) = cr(p, \mathcal{I})$ holds for each child p of the root. That is, after the execution of instruction (6) (the basis step of the descending phase), the root and its two children, which are the only unmarked vertices in the current e-join tree, contain precisely their respective consistent reducts for \mathcal{I} .

(*Induction step.*) Assume the property holds for the e-join tree T computed after some executions of the loop at step 7. Consider one of the parallel steps executed on T by instruction (7.a), say, the execution of $r\text{-shunt}(w)$ on vertices l and p , where p is the parent of l , and both vertices have the highest mark w of T . Let s be the parent of p , and q the parent of s , as shown in Figure 8. By the induction hypothesis, for the unmarked vertices s and q , $rel(s) = cr(s, \mathcal{I})$ and $rel(q) = cr(q, \mathcal{I})$. By applying $r\text{-shunt}(w)$ on l , we get a new e-join tree T' where p replaces s as a child of q and s becomes a child of p (the other one is l). First, it can be shown that $Rel_{T'}(p) = cr(p, \mathcal{I})$. To this end, note that the consistent reduct of p can be obtained as $cr(p, \mathcal{I}) = \prod_{R_T(p)}(UT_q \bowtie rel(q) \bowtie T_s)$, where $UT_q = \text{verts}(T) \setminus \text{verts}(T_s)$. Moreover, we use the fact that all variables occurring in both UT_q and T_s belong to $\text{Scope}_T(q)$, and that $rel(q)$ is the consistent reduct of q and hence it is clearly consistent with $\bowtie UT_q$. It follows that $cr(p, \mathcal{I}) = \prod_{R_T(p)}(rel(q) \bowtie T_s)$. By means of a further elaboration on this expression, using a similar (though more involved) argument, we get $cr(p, \mathcal{I}) = \prod_{R_T(p)}(rel(q) \bowtie rel(p) \bowtie SP \bowtie rel(s))$, where $SP = \text{verts}(MT_s) \setminus \text{verts}(T_p)$. Recall that, at the execution of the $shunt(w)$, we stored a relation $Rel(s_w)$, which can be shown to be consistent with the full subproblem encoded by SP . Moreover, $Sch(s_w) \subseteq (\text{Scope}_T(p) \cup \text{Scope}_T(s))$ by construction, so that we can prove $cr(p, \mathcal{I}) = \prod_{R_T(p)}(rel(q) \bowtie rel(p) \bowtie rel(s) \bowtie rel(s_w))$. Because $\text{Scope}_T(s) \subseteq Sch(s_w)$ and $(\text{Scope}_T(q) \cap Sch(s_w)) \subseteq \text{Scope}_T(p)$, this constraint relation is equal to the result of the semi-join based computation in $r\text{-shunt}(w)$. By similar arguments, we get that $r\text{-shunt}(w)$ gives to us the constraint reduct for node l , too. \square

Theorem 5.2. *On a parallel DB-machine with c processors, given a CSP instance \mathcal{I} with an e-join tree for it having n vertices, the GLOBAL-CONSISTENCY algorithm performs*

- (a) *a sequence of at most $4(\lceil \log c \rceil + 2\lceil n/4c \rceil)$ parallel shunt operations;*
- (b) *by using $O(n)$ intermediate relations, having size $O(d^2)$, where d is the size of the largest constraint relation.*

Proof. Assume first that the number of processors c is half the number of leaves of the e-join tree, which is known to be $\lceil n/2 \rceil$, because it is a strictly-binary tree. Then each parallel operation can be executed by a different processor, and the ascending phase of GLOBAL-CONSISTENCY performs $2\lceil \log n/4 \rceil$ shunt operations, because after each run of instructions 4a and 4b the number of leaves is reduced by half.

For the general case with $c < \lceil n/4 \rceil$: at the first iteration of the loop, we can equally divide the parallel shunt operations among the c processors, which means $2\lceil n/4c \rceil$ operations. In the subsequent steps, the number of leaves (hence of needed processors) halves at each step. Then, to complete the loop, we need $2(2\lceil n/4c \rceil - 1)$ operations until $c = \lceil n/4 \rceil$, followed by further $2\lceil \log c \rceil$ operations, as above.

Moreover, we note that this phase uses $O(n)$ intermediate relations stored in the e-join tree, and we claim that the size of each such relation requires at most $O(d^2)$ space.

The claim is proven by induction, assuming that for each vertex v , at each step, both $\|\prod_{R_T(v)} rel(v)\| \leq d$ and $\|\prod_{I_T(v)} rel(v)\| \leq d$ hold. The claim trivially holds when the ascending phase starts. Consider the generic step depicted in Figure 5 where, after the shunt operation, the vertex s , which is a child of vertex p in the current tree T , becomes a child of vertex q in the new tree T' . Recall that, by definition, $I_{T'}(s) \subseteq R_T(q)$. Because of the semijoin operation in the computation of the relation C and the fact that, by induction, $\|\prod_{R_T(q)} rel(q)\| \leq d$, we get $d \geq \|\prod_{R_T(q)} C\| \geq \|\prod_{I_{T'}(s)} C\|$. Moreover, $\|\prod_{R_T(s)} rel(s)\|$ is monotonically decreasing during the procedure. Then, the size of the relation $rel(s)$ in the e-join tree T' is at most d^2 , as it is at most the cartesian product $\|\prod_{I_{T'}(s)} C \times \prod_{R_{T'}(s)} rel(s)\| \leq d^2$. Note that all other vertices in T' have the same schema they have in T , and thus their stored relations, after the additional join operations computed in C , may only lose tuples with respect to T . For the sake of completeness, we note that the computation of the whole relation C is in fact not necessary to obtain the new relations in T' . Indeed, note that T encodes an acyclic query, so that the new relations for the vertices l, p , and s_w can be computed easily in linear space by using the classical semijoin algorithm for enforcing global consistency over the acyclic subquery induced by vertices l, p, q , and s . After this procedure, the new relation for vertex s in T' can be computed by evaluating $\|\prod_{Scope_{T'}(s)} rel(p) \bowtie rel(s)\|$.

The descending phase is even more efficient than the ascending phase. Indeed, the relation sizes decrease monotonically in the descending phase (only semi-joins and projections are applied). With respect to the constraints appearing in the given e-join tree, the algorithm uses additional intermediate relations to store the s_w vertices; however, since there is only one additional relation for each shunt operation performed in the ascending phase, the total number of intermediate relations is still linear in the size of the e-join tree (and hence in the number of constraints of the given instance). Moreover, the number of (parallel) steps performed in the descending phase is exactly the same as in the ascending phase. The same number of processors used in the ascending phase

Input: An acyclic CSP instance $(\Phi_{\mathcal{F}}, O, \text{DB})$ and an e-Join Tree T .
Output: The best solutions of $(\Phi_{\mathcal{F}}, O, \text{DB})$.
begin
 Let λ be the number of leaves in T ;
 (1) Label the leaves of T in order from left to right as $1, \dots, \lambda$;
 (2) **while** $\text{depth}(T) > 1$ **do**
 (a) **in parallel** apply the *shunt* operation to all odd numbered leaves that
 are the left children of their parent, and have depth greater than 1;
 (b) **in parallel** apply the *shunt* operation to all odd numbered leaves that
 are the right children of their parent, and have depth greater than 1;
 (c) shift out the rightmost bit in the labels of all remaining leaves;
 end (* while *)
 Let p be the root of T , and let p' and p'' be the children of p in T
 (3) **output** $\prod_O^{\oplus}(\text{rel}(p) \bowtie^{\oplus} \text{rel}(p') \bowtie^{\oplus} \text{rel}(p''))$
end.

Figure 10: The parallel algorithm ACSP.

is clearly sufficient to perform the descending phase, and also the number of relational operations is the same (apart from a constant factor). \square

5.2. Computing Max

Figure 10 shows the parallel algorithm ACSP for computing the best solutions of a given (already made) acyclic instance $(\Phi_{\mathcal{F}}, O, \text{DB})$, given an e-join tree T of a tree projection for the given instance, and where \mathcal{F} is a valuation function for Φ over \oplus . The e-join tree T is assumed to hold a global consistent instance, possibly computed by using the algorithm described in the previous section. Algorithm ACSP outputs a relation with the solutions θ over O , each one annotated with a value $\text{val}(\theta) \in \mathbb{D}$, which is the best possible value that can be obtained by extending the partial substitution θ to a full solution of the given instance. All tuples leading to the maximum value can thus be obtained immediately from this output relation. Clearly enough, such best solutions will be also the best solutions (w.r.t. \mathcal{F}) of the original (possibly non-acyclic) constraint formula (Φ', DB', O) .

Algorithm ACSP proceeds similarly to the ascending phase of the previous algorithm, but the output variables are suitably preserved during all the computation, by propagating them towards the root of T . To this end, the scope $\text{Scope}_T(p)$ of any vertex p is now partitioned in three (instead of two) distinguished set of variables $R_T(p)$, $I_T(p)$, and $O_T(p)$, where the new set $O_T(p)$ contains output variables. For each vertex p of T , $O_T(p)$ is initialized to \emptyset .

The major novelty is due to the necessity of dealing with the optimization problem, which requires an extension of the classical relational operators so to manage the values provided by the weighting functions in $\text{wfs}(\mathcal{F})$. Every tuple θ of every vertex of the e-join tree T is associated with a value $\text{val}(\theta) \in \mathbb{D}$. Without loss of generality, we assume that

\mathbb{D} contains a neutral value for \oplus , denoted by 0, such that $a \oplus 0 = 0 \oplus a = a$, for all $a \in \mathbb{D}$.¹⁰

The e-join tree T is initialized as follows: Every vertex s whose associated constraint occurs in the constraint formula with some weigh function f , is such that every tuple $\theta \in \text{rel}(s)$ has value $\text{val}(\theta) = f(\theta)$; for every other relation in the e-join tree, all tuples have value 0.

Given two vertices R_1 and R_2 , define the extended join operation $R_1 \bowtie^\oplus R_2$ as the set of tuples

$$\{\theta \in R_1 \bowtie R_2, \text{ with value } \text{val}(\theta) = \max\{\text{val}(\theta') \oplus \text{val}(\theta'') \mid \theta = \theta' \cup \theta'', \theta' \in R_1, \theta'' \in R_2\}\},$$

and the extended projection operation $\prod_X^\oplus R$ of a relation R over a set of variables X as

$$\{\theta \in \prod_X R, \text{ with value } \text{val}(\theta) = \max\{\text{val}(\theta') \in R \mid \theta'[X] = \theta\}.$$

Then, the shunt operation is redefined as follows. Let l be a leaf of an e-join tree T , p the parent of l , s the other child of p , and q the parent of p . The *shunt operation* applied to l results in a new contracted e-join tree T' in which l and p are deleted, s is transformed (as specified below) and takes the place of vertex p (i.e., it becomes child of q). The scope and the constraint relation of the transformed version of s in T' are specified below.

$$\begin{aligned} R_{T'}(s) &= R_T(s) \\ I_{T'}(s) &= (R_T(q) \cap \text{Scope}_T(p)) \setminus R_T(s) \\ O_{T'}(s) &= ((\text{Scope}_T(l) \cup \text{Scope}_T(p) \cup \text{Scope}_T(s)) \cap O) \setminus (R_{T'}(s) \cup I_{T'}(s)) \\ Rel_{T'}(s) &= \prod_{\text{Scope}_{T'}(s)}^\oplus (\text{rel}(l) \bowtie^\oplus \text{rel}(p) \bowtie^\oplus \text{rel}(s)) \end{aligned}$$

Thus, each output variable occurring in a deleted constraint (p or l) is kept in s (if it does not belong to $R_{T'}(s) \cup I_{T'}(s)$, then it is added to $O_{T'}(s)$).

Theorem 5.3. *The algorithm ACSP is correct.*

Proof. The proof can be derived by a similar line of reasoning as for the previous algorithm; only a few remarks are needed. By the new definition of shunt, no output variable disappears from the e-join tree during the computation: whenever the last constraint containing an output variable X is deleted, X is stored in an O_T attribute of another vertex. Thus, when the last iteration of instruction (3) is executed, all variables in O are still present in the tree (i.e., it holds that $O \subseteq \text{Scope}_T(p) \cup \text{Scope}_T(p') \cup \text{Scope}_T(p'')$). Moreover, if a variable X occurs in $O_T(p)$ for some vertex p of an e-join tree T generated during the computation of Algorithm ACSP, then X does not appear in the scope $\text{Scope}_T(q)$ of any other vertex q of T . Thus, O_T variables are not playing as join attributes in any shunt operation performed in the algorithm. They are used only for storing and preserving,

¹⁰Indeed, the neutral element is just used in the proposed algorithm to manage operations involving tuples with no assigned value. It can be easily simulated by dealing explicitly with such case.

during the computation, the assignments over output variables that will eventually be output as solutions.

Finally, at the end of the algorithm, the root holds a relation where the value $val(\theta) \in \mathbb{D}$ of each tuple θ is the best possible value that can be obtained by extending it to a full solution of the given instance. Indeed recall that \mathcal{F} is a valuation function for Φ over a single operator \oplus so that there is always a natural embedding for any tree projection, with every vertex that can play the role of a separator. Then, the statement easily follows by the definition of the extended operators \bowtie^\oplus and \prod^\oplus , and by Lemma 4.5. \square

Theorem 5.4. *On a parallel DB-machine with c processors, given a globally consistent CSP instance (Φ, O, DB) with an e-join tree for it having n vertices, the best solutions (over the variables O) can be computed by performing*

- (a) *a sequence of at most $2(\lceil \log c \rceil + 2\lceil n/4c \rceil)$ parallel shunt operations;*
- (b) *by using $O(n)$ intermediate relations, having size $O(vd^2)$, where d is the size of the largest constraint relation, and v the size of the solutions.*

Proof. Use Algorithm ACSP on (Φ, O, DB) and its e-join tree: the while loop of instruction 2 is very similar to the ascending phase of the previous algorithm, so that it is easy to see that property (a) of the theorem stems from the proof of Theorem 5.2.

Concerning property (b), compare the algorithm for enforcing global consistency in Figure 9 with ACSP, where the extra output variables that are kept in the e-join tree T may increase the size of the intermediate relations. If the cardinality of the set O is bounded by a fixed constant, such a size is bounded by a polynomial of the input size. However, in general the set O is arbitrary, possibly the whole set of variables occurring in the CSP instance. Therefore, to get the desired output-polynomial bound (property b), we use the crucial property that the given instance is global consistent, so that each tuple of values which is stored in the output variables during the while loop of ACSP will eventually be part of the solutions (no tuple can be deleted during the while loop). It follows that the size of the (partial) assignments over the output variables in the intermediate relations cannot exceed the size v of the result. \square

Because the number of solutions over the output variables O can be exponential with respect to the size of the input, if we are interested in computing just one best solution, the complexity can be made smaller by using $O = \emptyset$ in ACSP and then building the desired solution backward with a final top-down step.

Corollary 5.5. *On a parallel DB-machine with c processors, given a globally consistent CSP instance (Φ, O, DB) with an e-join tree for it having n vertices, $\text{MAX}(\Phi_{\mathcal{F}}, O, DB)$ can be computed by performing*

- (a) *a sequence of at most $4(\lceil \log c \rceil + 2\lceil n/4c \rceil)$ parallel shunt operations;*
- (b) *by using $O(n)$ intermediate relations, having size $O(d^2)$, where d is the size of the largest constraint relation.*

6. Further Related Work

In this section, in addition to the references already given, we discuss further literature that is closely related to our research, by emphasizing our specific modeling choices.

Soft Constraints. Soft constraints are classical constraints [1] enriched with the ability of associating either with the entire constraint or with each assignment of its variables a weight (meant to encode, for instance, a level of preference or a cost). The use of soft constraints leads to generalizations of the basic CSP setting, such as *fuzzy* [47], *probabilistic* [48], *possibilistic* [49], *partial* [50], and *lexicographic* [51] CSPs. These extensions can be viewed as special instances of the general setting of *semiring*-based CSPs [26], where each assignment in a constraint is associated with a value taken from a domain \mathbb{D} over which a *constraint-semiring* $\langle \mathbb{D}, \oplus, \otimes, 0, 1 \rangle$ is defined¹¹. Intuitively, \otimes is a binary operator combining the values associated with the various constraints, while \oplus is a binary operator inducing a partial order \succeq_{\oplus} over \mathbb{D} such that $a \succeq_{\oplus} b$ if and only if $a = a \oplus b$ holds. The goal is to find an assignment whose total value is minimal w.r.t. this order.

In many cases of practical interest, the domain \mathbb{D} is already associated with a total order and, therefore, the semiring-based model can be reduced to the setting of *valued* CSPs [52]—see [2], for a formal comparison of the two settings. In a valued CSP, the domain is part of a valuation structure $\langle \mathbb{D}, \otimes, \geq \rangle$, where \geq is a total order over \mathbb{D} and where \otimes is a commutative, associative, and monotonic binary operator used (as usual) to combine the values. For instance, let $k \in \mathbb{N} \cup \{\infty\}$ be an element taken from the set of the natural numbers extended with the positive infinity¹² and consider the valuation structure $\langle \{0, 1, \dots, k\}, \oplus, \geq \rangle$ where $a \oplus b = \min\{k, a + b\}$. This structure gives rise to the well-known *weighted* CSP setting: For each constraint (S_v, r_v) and for each assignment θ_v over the variables in S_v , if θ_v is not in r_v , then its value is k , which basically means that the given partial assignment is forbidden. Instead, if θ_v is in r_v , then its value is a non-negative cost. The goal is to check whether there is any \geq -minimal assignment whose associated value is different from ∞ , and to compute one if any. Despite their simplicity, weighted CSPs are expressive enough to model all valued CSPs over discrete valuation structures, provided that \otimes has a partial inverse [53].

Encodings. From the above discussion, it is easily seen that *hard* and soft constraints do not need to be explicitly distinguished in the formalization, since hard constraints can be enforced by just associating an infinite cost to the assignments that are not admissible (and then looking for solutions with minimum cost). However, algorithms exploited to process hard constraints often assume a relational representation where only the allowed tuples of values for each hard constraint are listed, while soft constraints algorithms assume a tabular representation where all possible assignments are listed together with their values. Therefore, encoding a hard constraint in terms of a soft one might lead to an exponential blow-up of the size of its representation. For an extreme example, consider a constraint (S_v, r_v) such that r_v does not contain any assignment. To encode this (hard) constraint in terms of a weighted CSP, we have to associate with all possible assignments over S_v the value ∞ . The number of these assignments is clearly exponential in the *arity* of S_v , i.e., $|S_v|$, and hence representing them in tabular form might quickly become unfeasible.

¹¹In particular, \oplus is closed, commutative, associative, idempotent, 0 is its unit element, and 1 is its absorbing element; \otimes is closed, commutative, associative, distributes over \oplus , 1 is its unit element, and 0 is its absorbing element.

¹²As usual, it holds that $a + \infty = \infty$ and $\infty \geq a$, for each $a \in \mathbb{N} \cup \{\infty\}$.

In fact, most of the works in the literature on soft CSPs do not care about the issue, because a *bounded-arity* setting is (implicitly) considered, i.e., the size of the largest constraint scope is assumed to be bounded by a fixed constant—when this constant is 2, then we get the classical setting of *binary* CSPs [54]. In the present paper, instead, we have not posed any arity bound on the given constraints, so that it was natural to avoid listing all possible assignments (cf. [55, 25]). Accordingly, we assumed that the input to our reasoning problems is given by a *standard* CSP instance \mathcal{I} , where only the assignments that are allowed are explicitly represented, plus an optimization function built on top of a valuation structure (over a set of binary operators) associating a value *only* with the assignments that are contained in some of the constraint relations of \mathcal{I} .

Decomposition Methods. One of the most important and deeply studied island of tractability for standard CSPs is the class of instances whose associated hypergraphs are *acyclic* [7, 56, 8]. Structural decomposition methods are approaches for extending the good results about this class to relevant classes of *nearly acyclic* structures (see, e.g., [57, 58, 59] and the references therein). On CSP instances having bounded arity, the *tree decomposition* [18, 60, 61] emerged to be the most powerful decomposition method [62]. Its natural counterpart over arbitrary instances is the (*generalized*) *hypertree* decomposition method [19, 16], but it has been observed that this method does not chart the frontier of tractability—for instance, further classes of tractable instances can be identified via the *fractional hypertree* method [63, 64] (see also the more recent results in [65]). All these methods (including fractional hypertree decompositions) fit into the framework of the tree projections [14, 15, 16], within which the results derived in the paper have been positioned.

Structural decomposition methods have been shown to play a crucial role even in presence of soft constraints, as introduced above. Indeed, the tractability of semiring-based and valued CSPs over structures having bounded treewidth has been shown in the literature [26, 27], and the effectiveness of the solution approaches has been practically validated, too (see, e.g., [66]). Actually, in a variety of automated reasoning areas, similar solution algorithms over structures having bounded treewidth have been proposed over the years. A unifying perspective of all of them has been provided by [25], where the concept of *graphical model* is introduced (essentially capturing a valuation structure) and where the bucket-tree elimination algorithm has been introduced to solve instances of this model having a tree-like structure. In [25] it has been also shown how these results can be extended if the notion of hypertree decomposition is used in place of the notion of tree decomposition. However, for this extension, the given valuation structure has to satisfy certain technical conditions¹³, which reduces its range of applicability.

Efficient solution algorithms that work on instances having bounded hypertree width and with arbitrary valuation structures have been more recently proposed by [12, 28]. In the present paper, we further generalized these algorithms to deal with complex optimization functions where more than just one aggregation operator is allowed. Moreover, unlike all the references reported above, we have not assumed that a decomposition is

¹³Values are real numbers, with 0 meant to encode that an assignment is not allowed/desirable. The binary operator \otimes must be absorbing relative to 0, i.e., $a \otimes 0 = 0$. For example, multiplication has this property while summation has not.

given (except for Section 5), which is a useful assumption when computing a decomposition is an NP-hard problem [16]. This has been obtained by designing promise-free algorithms that either compute correct certified solutions, or disprove the (promised) existence of a decomposition, in the spirit of [20]. Both generalizations are non trivial, and novel technical machineries have been required.

Constraint Propagation. From a technical viewpoint, our algorithms are based on procedures enforcing pairwise consistency [22], also known in the CSP community as *relational arc consistency* (or arc consistency on the dual graph) [1], *2-wise consistency* [23], and $R(*, 2)C$ [24], which are suitably adapted to deal with the propagation of the weights (in addition to the propagation of the information in the allowed assignments).

Constraint propagation is a fundamental technique in the context of CSPs (see, e.g., [1]), and a number of different propagation strategies have been proposed over the years for standard CSPs. Moreover, it is known since the very introduction of the setting of valued and semiring-based CSPs that suitable notions of consistency can be used for propagation in the constraint optimization framework, too. It has been observed in [26] that when the aggregation operator is idempotent, then *soft local consistency* terminates by producing an instance equivalent to the original one and in a way that the result does not depend on the order of application of the propagation rules—see also [67], for a general environment supporting different forms of soft propagations founded on these properties. The result excludes non-idempotent operators, such as the very basic summation, and hence it has a limited scope of applicability. However, for operators that are not necessarily idempotent but admit a partial inverse, a notion of *soft arc consistency* can be defined whose enforcement preserves at least the equivalence with the original instance [68]. The goal of soft arc consistency is to transform a problem into an equivalent one, by providing incrementally maintainable bounds which are crucial for branch and bound search [69]. Motivated by the fact that the fixpoint of the computation is not unique and may lead to different lower bounds, the problem of finding optimal sequences of soft arc consistency operations has been recently addressed by [70].

While our algorithms might be abstractly viewed as methods enforcing (suitable kinds) of soft consistency, it must be pointed out that they are completely orthogonal to the research illustrated above. The correctness of our propagation strategies (except for Section 5) only requires that the instance has a tree-like structure (without any further knowledge about it, but its existence) to find the optimal sequence of consistency operations leading not only to a bound, but in fact to the exact solution. There is no obvious way to extend our results to design algorithms for efficiently solving (or just enforcing bounds in) arbitrary instances.

7. Conclusion

A formal framework for constraint optimization has been proposed and analyzed. The computational complexity of reasoning problems related to computing the best solutions have been studied. In particular, structural tractability results have been derived within the general setting of tree projections. Transferring our theoretical findings into the design of a practical platform for constraint optimization is a natural avenue of further research. With respect to foundational analysis and theoretical contributions, instead,

efforts might be spent to study extensions of the framework supporting, for instance, forms of multi-criteria optimization.

Acknowledgments

Georg Gottlob’s work was supported by the EPSRC Programme Grant EP/M025268/ “VADA: Value Added Data Systems – Principles and Architecture”. The work of Gianluigi Greco was supported by the Italian Ministry for Economic Development under PON project “Smarter Solutions in the Big Data World” and by the Regione Calabria under POR project “Explora Process”.

References

- [1] R. Dechter, *Constraint Processing*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
- [2] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, Semiring-based CSPs and valued CSPs: Basic properties and comparison, in: *Over-Constrained Systems*, 1996, pp. 111–150.
- [3] N. Flerova, R. Marinescu, R. Dechter, Searching for the M best solutions in graphical models, *Journal of Artificial Intelligence Research* 55 (2016) 889–952.
- [4] A. Bulatov, V. Dalmau, M. Grohe, D. Marx, Enumerating homomorphisms, *Journal of Computer and System Sciences* 78 (2) (2012) 638–650.
- [5] R. Brafman, F. Rossi, D. Salvagnin, K. Venable, T. Walsh, Finding the next solution in constraint- and preference-based knowledge representation formalisms, in: *Proc. of KR’10*, 2010, pp. 425–433.
- [6] U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, *Information Sciences* 7 (1974) 95–32.
- [7] M. Yannakakis, Algorithms for acyclic database schemes, in: *Proc. of VLDB’81*, 1981, pp. 82–94.
- [8] R. Fagin, Degrees of acyclicity for hypergraphs and relational database schemes, *Journal of the ACM* 30 (3) (1983) 514–550.
- [9] G. Gottlob, G. Greco, N. Leone, F. Scarcello, Hypertree decompositions: Questions and answers, in: *Proc of PODS’16*, Association for Computing Machinery (ACM), 2016, pp. 57–74.
- [10] B. Kimelfeld, Y. Sagiv, Incrementally computing ordered answers of acyclic conjunctive queries, in: *Proc. of NGITS’06*, 2006, pp. 33–38.
- [11] G. Gottlob, G. Greco, F. Scarcello, Tractable optimization problems through hypergraph-based structural restrictions, in: *Proc. of ICALP’09*, 2009, pp. 16–30.
- [12] G. Greco, F. Scarcello, Structural tractability of constraint optimization, in: *Proc. of CP’11*, 2011, pp. 340–355.
- [13] D. Marx, Tractable hypergraph properties for constraint satisfaction and conjunctive queries, *Journal of the ACM* 60 (6) (2013) 42:1–42:51.
- [14] N. Goodman, O. Shmueli, The tree projection theorem and relational query processing, *Journal of Computer and System Sciences* 28 (1) (1984) 60–79.
- [15] Y. Sagiv, O. Shmueli, Solving queries by tree projections, *ACM Transactions on Database Systems* 18 (3) (1993) 487–511.
- [16] G. Gottlob, Z. Miklós, T. Schwentick, Generalized hypertree decompositions: NP-hardness and tractable variants, *Journal of the ACM* 56 (6) (2009) 1–32.
- [17] G. Greco, F. Scarcello, Greedy strategies and larger islands of tractability for conjunctive queries and constraint satisfaction problems, *Information and Computation* 252 (2017) 201–220.
- [18] N. Robertson, P. Seymour, Graph minors. ii. algorithmic aspects of tree-width, *Journal of Algorithms* 7 (3) (1986) 309–322.
- [19] G. Gottlob, N. Leone, F. Scarcello, Hypertree decompositions and tractable queries, *Journal of Computer and System Sciences* 64 (3) (2002) 579–627.
- [20] H. Chen, V. Dalmau, Beyond hypertree width: Decomposition methods without decompositions, in: *Proc. of CP’05*, 2005, pp. 167–181.
- [21] G. Greco, F. Scarcello, The power of local consistency in conjunctive queries and constraint satisfaction problems, *SIAM Journal on Computing* (2017) 1111–1145.

- [22] C. Beeri, R. Fagin, D. Maier, M. Yannakakis, On the desirability of acyclic database schemes, *Journal of the ACM* 30 (3) (1983) 479–513.
- [23] M. Gyssens, On the complexity of join dependencies, *ACM Transactions on Database Systems* 11 (1) (1986) 81–108.
- [24] S. Karakashian, R. J. Woodward, C. G. Reeson, B. Y. Choueiry, C. Bessiere, A first practical algorithm for high levels of relational consistency, in: *Proc. of AAAI’10*, 2010, pp. 101–107.
- [25] K. Kask, R. Dechter, J. Larrosa, A. Dechter, Unifying tree decompositions for reasoning in graphical models, *Artificial Intelligence* 166 (1-2) (2005) 165–193.
- [26] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint satisfaction and optimization, *Journal of the ACM* 44 (2) (1997) 201–236.
- [27] C. Terrioux, P. Jégou, Bounded backtracking for the valued constraint satisfaction problems, in: *Proc. of CP’03*, 2003, pp. 709–723.
- [28] G. Gottlob, G. Greco, Decomposing combinatorial auctions and set packing problems, *Journal of the ACM* 60 (4) (2013) 1–39.
- [29] M. Joglekar, R. Puttagunta, C. Ré, Aggregations over generalized hypertree decompositions, *CoRR* abs/1508.07532.
- [30] M. Abo Khamis, H. Q. Ngo, A. Rudra, Faq: Questions asked frequently, in: *Proc of PODS’16*, 2016, pp. 13–28.
- [31] S. Lauritzen, D. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, in: *Readings in Uncertain Reasoning*, Morgan Kaufmann Publishers Inc., 1990, pp. 415–448.
- [32] R. G. Downey, M. R. Fellows, *Parameterized Complexity*, Springer Publishing Company, Incorporated, 2012.
- [33] P. Kolaitis, M. Vardi, Conjunctive-query containment and constraint satisfaction, *Journal of Computer and System Sciences* 61 (2) (2000) 302–332.
- [34] P. Bernstein, N. Goodman, Power of natural semijoins, *SIAM Journal on Computing* 10 (4) (1981) 751–771.
- [35] G. Greco, F. Scarcello, Structural tractability of enumerating csp solutions, *Constraints* 18 (1) (2013) 38–74.
- [36] E. Lawler, A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem, *Management Science* 18 (7) (1972) 401–405.
- [37] S. Kasif, On the parallel complexity of discrete relaxation in constraint satisfaction networks, *Artificial Intelligence* 45 (3) (1990) 275–286.
- [38] G. Gottlob, N. Leone, F. Scarcello, The complexity of acyclic conjunctive queries, *Journal of the ACM* 48 (3) (2001) 431–498.
- [39] G. Gottlob, N. Leone, F. Scarcello, Computing LOGCFL certificates, *Theoretical Computer Science* 270 (1-2) (2002) 761–777.
- [40] G. Gottlob, N. Leone, F. Scarcello, Advanced parallel algorithms for acyclic conjunctive queries, *Tech. Rep. DBAI-TR-98/18*, Technical University of Vienna (1998).
- [41] G. Greco, F. Scarcello, Tree projections and structural decomposition methods: Minimality and game-theoretic characterization, *Theoretical Computer Science* 522 (2014) 95–114.
- [42] R. M. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science (Vol. A)*, MIT Press, Cambridge, MA, USA, 1990, pp. 869–941.
- [43] I. P. Gent, C. Jefferson, I. Miguel, N. C. Moore, P. Nightingale, P. Prosser, C. Unsworth, A preliminary review of literature on parallel constraint solving, in: *Proc. of PMCS’11*, 2011.
- [44] L. G. Valiant, A bridging model for multi-core computing, *Journal of Computer and System Sciences* 77 (1) (2011) 154–166.
- [45] U. Vishkin, Using simple abstraction to reinvent computing for parallelism, *Communications of the ACM* 54 (1) (2011) 75.
- [46] F. N. Afrati, M. R. Joglekar, C. Ré, S. Salihoglu, J. D. Ullman, GYM: A multiround distributed join algorithm, in: *Proc. of ICDT’17*, 2017, pp. 4:1–4:18.
- [47] D. Dubois, H. Fargier, H. Prade, The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction, in: *Proc. of FUZZ-IEEE’93*, 1993, pp. 1131–1136.
- [48] H. Fargier, J. Lang, Uncertainty in constraint satisfaction problems: A probabilistic approach, in: *Proc. of ECSQARU’93*, 1993, pp. 97–104.
- [49] T. Schiex, Possibilistic constraint satisfaction problems or “how to handle soft constraints?”, in: *Proc. of UAI’92*, 1992, pp. 268–275.
- [50] E. Freuder, R. Wallace, Partial constraint satisfaction, *Artificial Intelligence* 58 (1992) 21–70.

- [51] H. Fargier, J. Lang, T. Schiex, Selecting preferred solutions in fuzzy constraint satisfaction problems, in: Proc. of EUFIT'93, 1993.
- [52] T. Schiex, H. Fargier, G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in: Proc. of IJCAI'95, 1995, pp. 631–637.
- [53] M. Cooper, High-order consistency in valued constraint satisfaction, Constraints 10 (3) (2005) 283–305.
- [54] F. Bacchus, X. Chen, P. van Beek, T. Walsh, Binary vs. non-binary constraints, Artificial Intelligence 140 (1-2) (2002) 1–37.
- [55] D. Larkin, R. Dechter, Bayesian inference in the presence of determinism, in: Proc. of AISTATS'03, 2003.
- [56] R. Fagin, A. Mendelzon, J. Ullman, A simplified universal relation assumption and its properties, ACM Transactions on Database Systems 7 (3) (1982) 343–360.
- [57] G. Gottlob, N. Leone, F. Scarcello, A comparison of structural csp decomposition methods, Artificial Intelligence 124 (2) (2000) 243–282.
- [58] D. Cohen, P. Jeavons, M. Gyssens, A unified theory of structural tractability for constraint satisfaction problems, Journal of Computer and System Sciences 74 (5) (2008) 721–743.
- [59] G. Greco, F. Scarcello, On the power of structural decompositions of graph-based representations of constraint problems, Artificial Intelligence 174 (5-6) (2010) 382–409.
- [60] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence 38 (3) (1989) 353–366.
- [61] J. Flum, M. Frick, M. Grohe, Query evaluation via tree-decompositions, Journal of the ACM 49 (6) (2002) 716–752.
- [62] M. Grohe, The complexity of homomorphism and constraint satisfaction problems seen from the other side, Journal of the ACM 54 (1) (2007) 1–24.
- [63] M. Grohe, D. Marx, Constraint solving via fractional edge covers, ACM Transactions on Algorithms 11 (1) (2014) 1–20.
- [64] D. Marx, Approximating fractional hypertree width, ACM Transactions on Algorithms 6 (2) (2010) 1–17.
- [65] W. Fischl, G. Gottlob, R. Pichler, General and fractional hypertree decompositions: Hard and easy cases, CoRR abs/1611.01090 - short version to appear in Proc. of PODS 2018.
- [66] P. Jégou, C. Terrioux, Hybrid backtracking bounded by tree-decomposition of constraint networks, Artificial Intelligence 146 (1) (2003) 43–75.
- [67] S. Bistarelli, T. Frühwirth, M. Marte, F. Rossi, Soft constraint propagation and solving in constraint handling rules, Computational Intelligence 20 (2) (2004) 287–307.
- [68] M. Cooper, T. Schiex, Arc consistency for soft constraints, Artificial Intelligence 154 (1-2) (2004) 199–227.
- [69] J. Larrosa, T. Schiex, In the quest of the best form of local consistency for weighted csp, in: Proc. of IJCAI'03, 2003, pp. 239–244.
- [70] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, T. Werner, Soft arc consistency revisited, Artificial Intelligence 174 (7-8) (2010) 449–478.

A. Proof of Theorem 4.8

In this section, we analyze the correctness of **Compute-Max**. The proof is rather involved and is discussed incrementally.

Hereinafter, recall that Φ is a constraint formula, DB a constraint database, $O \subseteq \text{vars}(\Phi)$ a set of variables, \mathcal{F} a valuation function, \mathcal{V} a view set for Φ , and DB' a legal database instance for \mathcal{V} w.r.t. Φ and DB . Moreover, recall the following.

Proposition A.1 ([21]). *Let Φ be a constraint formula over DB , \mathcal{V} a view set for Φ , and DB' a legal database for \mathcal{V} w.r.t. Φ and DB . Let Φ_a be a sandwich formula of Φ' w.r.t. \mathcal{V} , where Φ' is a subformula of Φ such that $\Phi' \equiv_h \Phi$. Then,*

- (1) *A database DB'_a legal for $\text{atoms}(\Phi_a)$ w.r.t. Φ and DB can be computed in polynomial time.*

- (2) If DB' is locally consistent w.r.t. \mathcal{V} , then for every $w \in \mathcal{V}$ and every $h \subseteq \text{vars}(w)$ such that there is atom $q_a \in \text{atoms}(\Phi_a)$ with $h \subseteq \text{vars}(q_a)$, $w^{\text{DB}'}[h] = \Phi^{\text{DB}'}[h]$.
- (3) If Φ_a is a sandwich formula of Φ w.r.t. \mathcal{V} (and not just of Φ'), then $\Phi_a^{\text{DB}'} = \Phi^{\text{DB}'}_a$;

A.1. Promise Tractability

For any set of variables O , define $\text{atom}(O)$ to be a fresh atom (with a fresh relation symbol) over these variables, i.e., such that $O = \text{vars}(\text{atom}(O))$. We start the analysis of **Compute-Max** by considering the following promise:

(P1) (\mathcal{F}, O) can be embedded in (Φ', \mathcal{V}) for some subformula Φ' of $\Phi \wedge \text{atom}(O)$ such that $\Phi' \equiv_{\mathbf{h}} \Phi \wedge \text{atom}(O)$.

Note that **P1** is less stringent than assuming that (\mathcal{F}, O) can be embedded in (Φ, \mathcal{V}) . In fact, we will find it convenient to show that **Compute-Max** is correct (as a promise algorithm), even under **P1**. To this end, we first prove the following technical lemma, stating some crucial properties of separator views.

Lemma A.2. *Assume that **P1** holds. After the last execution of **evaluate** in **Compute-Max**, sep_s contains a view $[w_O X]$ with $\text{vars}(w_O) = O$ such that, for all $\theta \in [w_O X]^{\text{DB}'_s}$, $\theta[X] = \max_{\mathcal{F}}(\theta[O])$.*

Proof. Since (\mathcal{F}, O) can be embedded in (Φ', \mathcal{V}) for some subformula $\Phi' \equiv_{\mathbf{h}} \Phi \wedge \text{atom}(O)$, there exists a sandwich (acyclic) formula Φ_a equivalent to Φ' and $\Phi \wedge \text{atom}(O)$ and an embedding ξ that satisfies the conditions for being an embedding for some join tree $JT = (V, E, \chi)$ of the tree projection \mathcal{H}_{Φ_a} . In particular, we may assume w.l.o.g that, for every leaf p_j , $\chi(\xi(p_j)) = \text{vars}(w_{f_j})$ for some function view w_{f_j} , and that the root p_s is mapped via ξ to $\text{atom}(O)$, and hence $\chi(\xi(p_s)) = O$. Recall that (p_1, \dots, p_s) is a topological ordering of the vertices of $\text{tree}(\mathcal{F}, O)$, and hence p_s is the root of the tree.

Note that after local consistency, from Proposition A.1, for every $q \in \Phi_a$, $q^{\text{DB}_1} = \Phi^{\text{DB}}[q]$. Thus, all assignments in the relations for the output view and the function views are partial homomorphisms that can be extended to full answers of the given constraint formula.

The proof is by induction on $i = 1 \dots s$. For any view $v \in \text{sep}_i$, any set of variables $h \subseteq \text{vars}(v)$, and any assignment $\theta \in v^{\text{DB}'_i}$, let $\max(\theta, X, h, v^{\text{DB}'_i}) = \max\{\theta'[X] \mid \theta' \in v^{\text{DB}'_i} \wedge \theta'[h] = \theta[h]\}$. We show that the following properties hold for views in any set sep_i , where the first of the two entails the statement of the lemma:

1. sep_i contains some view $[wX]$ with $h_i \subseteq \text{vars}(w)$, where h_i are the variables of the p_i -separator, and for all $\theta \in [wX]^{\text{DB}'_i}$, $\max_{\mathcal{F}_{p_i}}(\theta[h_i]) = \max(\theta, X, h_i, [wX]^{\text{DB}'_i})$;
2. for each view $[w'X] \in \text{sep}_i$, for all $\theta \in [w'X]^{\text{DB}'_i}$, $\theta[X] \geq \max_{\mathcal{F}_{p_i}}(\theta[w'])$.¹⁴

(Basis: $i = 1$.) The vertex p_1 is a leaf of $\text{tree}(\mathcal{F}, O)$ and sep_1 contains one view $[w_{f_1} X_1^{(w_{f_1})}]$, whose database relation is $\{\theta \cup \{X_1^{(w_{f_1})}/f_1(t)\} \mid \theta \in w_{f_1}^{\text{DB}_1}\}$. Clearly, in this case, every assignment $\theta[w_{f_1}]$ has only one possible weight for $f_1(\theta)$, because $f_1 = \mathcal{F}_{p_1}$ is evaluated precisely on its variables $\text{vars}(w_{f_1})$. Therefore, $\theta[X_1^{(w_{f_1})}] = \max_{\mathcal{F}_{p_1}}(\theta)$ for

¹⁴Recall that $\max_{\mathcal{F}_{p_i}}(\theta[w']) = \perp$, whenever $\theta[w']$ cannot be extended to any full answer of Φ .

every assignment θ . Indeed, θ is an assignment of a function view and thus it is part of some answer of Φ . As a consequence, there are full answers in Φ^{DB} that extend θ , and all of them will get $\theta[X_1^{(w_{f_1})}]$ as their weight according to \mathcal{F}_{p_1} . Finally, recall that, being p_1 a leaf, we have $\chi(\xi(p_1)) = \text{vars}(f_1) = \text{vars}(w_{f_1})$.

(Inductive Step: $i = j + 1$.) Assume the statement holds until some $j \geq 1$. That is, we have executed function *evaluate* at step j , and we consider the execution of function *propagate*, in order to obtain the new constraint database DB_{j+1} . This means that we would like to propagate the weights of \mathcal{F}_{p_j} to the candidate separators in sep_r , where p_r is the parent of p_j in $\text{tree}(\mathcal{F}, O)$. By the inductive hypothesis, there exists some view $[wX_j^{(w)}] \in \text{sep}_j$ such that $h_j \subseteq \text{vars}(w)$ are the variables of the separator $\xi(p_j)$, and where $\forall \theta \in [wX_j^{(w)}]^{\text{DB}_j}$, $\max(\theta, X_j^{(w)}, h_j, [wX_j^{(w)}]^{\text{DB}_j}) = \max_{\mathcal{F}_{p_j}}(\theta[h_j])$. Let $h_r = \chi(\xi(p_r))$ be set of variables of the p_r -separator occurring in the tree projection and thus included in the variables of some atom a_r of the sandwich formula Φ_a . Since all views in \mathcal{V} are considered as candidates to be p_r -separators, sep_r contains some views of the form $[w_r X' X'']$, such that $h_r \subseteq \text{vars}(w_r)$. By construction—see the initialization step—among them there is also an augmented view having the form $[w_r X_j^{(w)} X']$. Then, there is a step in function *propagate* where we enforce local consistency on the pair $(\mathcal{V}_j, \text{DB}'_j)$ where \mathcal{V}_j includes $[wX_j^{(w)}]$ and $[w_r X_j^{(w)} X']$, together with all augmented views $[w_b X_j^{(w)}]$, for each $w_b \in \mathcal{V}$. Observe that the corresponding constraint database DB'_j is legal for \mathcal{V}_j w.r.t. $\Phi_+ = \Phi' \wedge [wX_j^{(w)}]$ and DB'' , where $\Phi'' = \bigwedge_{q \in \Phi'} [qX_j^{(w)}]$; and the relations in DB'' have the form $\text{rel}(q, \text{DB}) \times \text{dom}(X_j^{(w)})$, for any $q \in \Phi'$, and $\text{rel}([wX_j^{(w)}], \text{DB}'_j)$ for $[wX_j^{(w)}]$. Indeed, DB_1 was legal w.r.t. Φ' and DB , and the new variable $X_j^{(w)}$ occurs in the view $[wX_j^{(w)}]$ with the right weights, and in all other relations with all possible weights in its active domain (for every assignment of the original relation).

Consider the acyclic hypergraph \mathcal{H}_a^+ such that $\text{nodes}(\mathcal{H}_a^+) = \text{nodes}(\mathcal{H}_{\Phi_a}) \cup \{X_j^{(w)}\}$, and $\text{edges}(\mathcal{H}_a^+) = \{h \cup \{X_j^{(w)}\} \mid h \in \text{edges}(\mathcal{H}_{\Phi_a})\}$. Since the new variable occurs in all available views, and h_j is covered in \mathcal{H}_{Φ_a} , \mathcal{H}_a^+ is clearly a tree projection of \mathcal{H}_{Φ_+} w.r.t. $\mathcal{H}_{\mathcal{V}_j}$. In particular, because \mathcal{H}_{Φ_a} covers h_r , the tree projection \mathcal{H}_a^+ covers $h_r \cup \{X_j^{(w)}\}$, too. From Proposition A.1, after DB_{j+1} is obtained by enforcing local consistency on $(\mathcal{V}_j, \text{DB}'_j)$, $[w_r X_j^{(w)} X']^{\text{DB}_{j+1}}[h_r \cup \{X_j^{(w)}\}] = \Phi_+^{\text{DB}''}[h_r \cup \{X_j^{(w)}\}]$. Recall that all assignments in $\text{rel}([wX_j^{(w)}], \text{DB}'') [h_j \cup \{X_j^{(w)}\}]$ are correct by the inductive hypothesis. Therefore, the last statement means that $\text{rel}([w_r X_j^{(w)} X'], \text{DB}_{j+1})$ is such that each assignment θ in this relation holds in the variable $X_j^{(w)}$ the weight of any assignment θ' of the p_j separator that can be extended to a same full answer of Φ' as θ , i.e., such that there exists $\theta'' \in \Phi^{\text{DB}}$ with $\theta''[h_j] = \theta'[h_j]$ and $\theta''[h_r] = \theta[h_r]$. In particular some of these assignments will hold the correct maximal weight $\max_{\mathcal{F}_{p_j}}(\theta[h_r])$. Note that all variables C that determine the evaluation of \mathcal{F}_{p_j} are covered in the subtree rooted at the separator $\xi(p_j)$ (cf. Theorem 3.6). From the connectedness condition of JT , $(C \cap h_r) \subseteq h_j$, and thus the above possible extensions of θ are precisely those relevant for the evaluation of \mathcal{F}_{p_j} . It follows that the maximum weight $\max(\theta, X_j^{(w)}, h_r, [w_r X_j^{(w)} X']^{\text{DB}_{j+1}})$ over all assignments $[w_r X_j^{(w)} X']^{\text{DB}_{j+1}}[h_r]$ that agree with $\theta[h_r]$ is the maximum weight

for \mathcal{F}_{p_j} over all possible extensions of $\theta[h_r]$ to answers in Φ'^{DB} . Finally, recall that Φ' is a subformula of $\Phi \wedge \text{atom}(O)$, and clearly considering further constraints in possible extensions encoded by the other atoms in Φ cannot improve such a weight. Therefore, $\max(\theta, X_j^{(w)}, h_r, [w_r X_j^{(w)} X']^{\text{DB}_{j+1}})$ is indeed equal to $\max_{\mathcal{F}_{p_j}}(\theta[h_r])$.

Now, consider instead the propagation from any generic view $[w' X_j^{(w')}] \in \text{sep}_j$, not necessarily including the variables of some p_j -separator, to a generic view $[w'_r X_j^{(w')} X'] \in \text{sep}_r$. From the inductive hypothesis, for all $\theta \in [w' X_j^{(w')}]^{\text{DB}_{j'}}$, $\theta[X_j^{(w')}] \geq \max_{\mathcal{F}_{p_j}}(\theta[h_j])$, where $h_j = \text{vars}(w')$. We next show that for all $\theta' \in [w'_r X_j^{(w')} X']^{\text{DB}_{j'}}$, $\theta'[X_j^{(w')}] \geq \max_{\mathcal{F}_{p_j}}(\theta'[h])$, where $h = \text{vars}(w'_r)$. To this end, consider the same construction of \mathcal{V}_j , DB'_j , Φ_+ , and DB'' as above, but where we use everywhere w' instead of w , and h instead of h_j . Note that in this general case we do not know whether there exists a tree projection of \mathcal{H}_{Φ_+} w.r.t. $\mathcal{H}_{\mathcal{V}_j}$ that covers both h and h'_r , thus guaranteeing the correct propagation of weights stored in $X_j^{(w')}$. However we can always add a suitable set of views \mathcal{V}_+ to \mathcal{V} having this property, i.e., such that there exists a tree projection \mathcal{H}'_a of \mathcal{H}_{Φ_+} w.r.t. $\mathcal{H}_{\mathcal{V}_j \cup \mathcal{V}_+}$ that covers both h and h'_r . For each view $w_+ \in \mathcal{V}_+$, we add to DB'_j the most liberal relation $\text{dom}(X_1) \times \dots \times \text{dom}(X_z)$, if X_1, \dots, X_z are the variables occurring in w_+ . Clearly, the resulting constraint database, say DB''_j is legal for $\mathcal{V}_j \cup \mathcal{V}_+$ w.r.t. Φ_+ and DB'' . Assume we enforce local consistency on $(\mathcal{V}_j \cup \mathcal{V}_+, \text{DB}''_j)$ and let DB_+ be the resulting constraint database. Consider any assignment $\theta' \in w_r'^{\text{DB}_+}$. If this assignment cannot be extended to any full solution (e.g., this is definitely the case if it does not belong to $w_r'^{\text{DB}_+}$), then $\mathcal{F}_{p_j}(\theta') = \perp$ and the statement trivially holds. Then, assume by contradiction that $\theta' \in w_r'^{\text{DB}_+} \cap \Phi'^{\text{DB}}[h'_r]$ and that $\theta'[X_j^{(w')}] < \max_{\mathcal{F}_{p_j}}(\theta'[h'_r])$. Thus, there exists some assignment $\theta_m \in \Phi'^{\text{DB}}$ such that $\theta_m[h'_r] = \theta'[h'_r]$ and $\mathcal{F}_{p_j}(\theta_m) > \theta'[X_j^{(w')}]$. Note that there cannot exist any assignment $\theta \in w'^{\text{DB}_+} \cap \Phi'^{\text{DB}}[h]$ with $\theta_m[h] = \theta[h]$ and $\theta[X_j^{(w')}] \geq \max_{\mathcal{F}_{p_j}}(\theta[h]) \geq \mathcal{F}_{p_j}(\theta_m)$, otherwise such a weight would be propagated to the view w_r after the local consistency procedure, thanks to the tree projection \mathcal{H}_{Φ_+} . Indeed, from Proposition A.1, we get $w'^{\text{DB}_+}[h \cup X_j^{(w')}] = \Phi_+^{\text{DB}''}[h \cup X_j^{(w')}]$ and $w_r'^{\text{DB}_+}[h'_r \cup X_j^{(w')}] = \Phi_+^{\text{DB}''}[h'_r \cup X_j^{(w')}]$. However, some assignment $\theta'' \in w'^{\text{DB}_+} \cap \Phi'^{\text{DB}}[h]$ such that $\theta_m[h] = \theta''[h]$ must exist since θ_m is a full solution and thus must match some assignment in every view, from the view-consistency property of the legal database DB_+ . Moreover, from the inductive hypothesis, we know that $\theta''[X_j^{(w')}] \geq \max_{\mathcal{F}_{p_j}}(\theta''[h])$, which thus leads to a contradiction. To conclude the analysis of this step, recall that we considered an additional set of views \mathcal{V}_+ , to get the desired decomposition. However, if we remove such views and consider only the set \mathcal{V}_j , we get more combinations available in $w_r'^{\text{DB}'_j}[h'_r \cup X_j^{(w')}]$ and more weights (possibly wrong) for assignments $\theta' \in w_r'^{\text{DB}'_j}$ among which to select an even larger weight for $\theta'[X_j^{(w')}]$.

To conclude the proof of the inductive step, consider now the execution of function *evaluate* at step $i = j + 1$. If p_i is a leaf, then the proof is the same as the base case p_1 . Therefore, assume p_i is not a leaf, and let p_b and p_c its children, with $b < c$, without loss of generality. Since we are proceeding according to a topological ordering,

both $b < c < i$ holds, and thus function *Propagate* has been already executed for views in sep_b and sep_c . Let h_i be the set of variables of the separator $\xi(p_i)$ that is covered in the tree projection \mathcal{H}_{Φ_a} , and let $w_i \in \mathcal{V}$ be a view with $h_i \subseteq vars(w_i)$. From the inductive hypothesis, there exists two views $[w_b X_b^{(w_b)}] \in sep_b$ and $[w_c X_c^{(w_c)}] \in sep_c$ that propagated the correct weights for \mathcal{F}_{p_b} and \mathcal{F}_{p_c} to their parent p_i . More precisely, after the above discussion on function *Propagate*, it follows that the view $[w_i X_b^{(w_b)} X_c^{(w_c)}]$ is such that, $\forall \theta \in [w_i X_b^{(w_b)} X_c^{(w_c)}]^{\text{DB}_i}$, $\max(\theta, X_b^{(w_b)}, h_i, [w_i X_b^{(w_b)} X_c^{(w_c)}]^{\text{DB}_i}) = \max_{\mathcal{F}_{p_b}}(\theta[h_i])$ and $\max(\theta, X_c^{(w_c)}, h_i, [w_i X_b^{(w_b)} X_c^{(w_c)}]^{\text{DB}_i}) = \max_{\mathcal{F}_{p_c}}(\theta[h_i])$. However, since local consistency holds and h_i is covered by the tree projection at hand, such assignments $\theta[h_i]$ are precisely the assignments of the relation $rel(w_{p_i}, \overline{\text{DB}})$ of the atom w_{p_i} in the sandwich formula covering the p_i -separator, as in the statement of Lemma 4.5. Therefore, if \oplus_i is the operator labeling p_i in the parse tree, from this lemma get $\max_{\mathcal{F}_{p_i}}(\theta[h_i]) = \max_{\mathcal{F}_{p_b}}(\theta[h_i]) \oplus_i \max_{\mathcal{F}_{p_c}}(\theta[h_i]) = \theta[X_b^{(w_b)}] \oplus_i \theta[X_c^{(w_c)}]$. It follows that the desired combined maximum is achieved, for any projected assignment $\theta[h_i]$, on some assignment $\theta' \in [w_i X_b^{(w_b)} X_c^{(w_c)}]^{\text{DB}_i}$, with $\theta'[h_i] = \theta[h_i]$ and $t'[w_i] \in \Phi^{\overline{\text{DB}}}$, that gets maximum weights according to both \mathcal{F}_{p_b} and \mathcal{F}_{p_c} . Of course such a correct assignment is preserved under any step of the algorithm, and we know that the weights are correctly propagated via the tree-projection. Then, the marginalization step correctly selects the right maximal weights for every assignment according to \mathcal{F}_{p_i} .

Finally, observe that, from the inductive property proved for the two variables $X_b^{(w_b)}$ and $X_c^{(w_c)}$ in the discussion about *propagate* and from Lemma 4.4, it follows that the inductive statements hold for sep_i , too. Now consider the step where the selection of a “minimum view” (if any) is executed. Observe that, by the inductive hypothesis, wrong views may only get better maxima for their assignments. Therefore, for any set of augmented views $\mathcal{V}_w \subseteq sep_i$ with the same “base” view w , we may safely select for each assignment $\theta \in w^{\overline{\text{DB}}}$, its version with the lowest marginalized weight $X_i^{(w')}$ among all views in \mathcal{V}_w . More precisely, observe that such a selection does not alter the validity of the two inductive statements. Moreover, as far as augmented views over different base views are concerned, we may safely consider the absolute maximum. Clearly, the right one (if any) should be the smallest one, from the inductive statements. Therefore, looking at this maximum, we are able to discard augmented views that cannot contain the variables of an actual sep_i separator, and more importantly we enforce the actual maximum to be an upper bound for all weights computed in the algorithm for \mathcal{F}_{p_i} and, at the end, for \mathcal{F} . \square

We can now show the correctness of **Compute-Max** under **P1**.

Theorem 4.7. *Algorithm **Compute-Max** runs in polynomial time. It outputs NO SOLUTION, only if $\Phi^{\text{DB}} = \emptyset$. Moreover, it computes an answer to $\text{MAX}(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$, with \mathcal{F} being a structured evaluation function, if **P1** holds. It outputs FAIL, only if **P1** does not hold.*

Proof. Note first that, whenever the algorithm outputs NO SOLUTION, then the constraint database obtained by enforcing local pairwise-consistency is empty. Hence, we are guaranteed that there is no solution at all in Φ^{DB} .

Consider then the case where the algorithm does not output **NO SOLUTION**. In this case, correctness follows from Lemma A.2 applied to the root p_s , which contains a unique augmented view, after the selection of the “minimum” view in function *evaluate*. Such a view contains only corrected weights if (\mathcal{F}, O) can be embedded in (Φ', \mathcal{V}) for some subformula Φ' of $\Phi \wedge \text{atom}(O)$ such that $\Phi' \equiv_h \Phi \wedge \text{atom}(O)$. On the other hand, the algorithm outputs **FAIL** only when it recognizes that this condition does not hold, because there are no feasible separators at some step, or when it turns out that the formula has no answers at all. Observe that Algorithm **Compute-Max** performs $s - 1$ iterations of the **for** loop plus some further operations, each one feasible in polynomial time (where the mathematical operations cost 1). In particular, local consistency requires polynomial time w.r.t. the given set of views and the given constraint database, and note that we deal with at most $O(|\mathcal{V}|^2 \|\mathcal{F}\|)$ views in the algorithm, where $\|\mathcal{F}\|$ denotes the size of \mathcal{F} , which is an upper bound to the total number of vertices of its parse trees.

Finally, note that the size of the weights computed during the execution of the algorithm is polynomially bounded w.r.t. the size of the input. Indeed, recall from Lemma A.2 that we avoid the computation of wrong weights larger than the actual maximum, if the promise is true. Otherwise, we cannot guarantee anything: just think that the formula may be empty if there are no tree-projections at all (no matter of formula embeddings). \square

A.2. Larger Islands of Tractability

We now show how to solve **MAX** under a less stringent promise than **P1**, by using **Compute-Max** as an oracle. For any variable X , define $\text{atom}_X(\{X\})$ to be a fresh atom (with a fresh relation symbol) such that $\{X\} = \text{vars}(\text{atom}_X(\{X\}))$.

Consider the following promise:

(P2) \mathcal{F} can be embedded in (Φ', \mathcal{V}) for some subformula Φ' of $\Phi \wedge \bigwedge_{X \in O} \text{atom}_X(\{X\})$ such that $\Phi' \equiv_h \Phi \wedge \bigwedge_{X \in O} \text{atom}_X(\{X\})$.

It is easy to see that **P1** entails **P2**, as every tree projection covering $\text{atom}(O)$ also covers all atoms of the form $\text{atom}_X(\{X\})$, for any $X \in O$.

Theorem A.3. *There is a polynomial-time algorithm that solves $\text{MAX}(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$ if **P2** holds. It outputs **FAIL**, only if **P2** does not hold.*

Proof. Let \bar{X} be a variable in O . Let $\mathcal{V}_{\bar{X}}$ be the set of views $\{[w\bar{X}] \mid w \in \mathcal{V}, X \notin \text{vars}(w)\} \cup \{w \mid w \in V, \bar{X} \in \text{vars}(w)\}$. Let $\text{DB}'_{\bar{X}}$ be the constraint database including all relations of DB' for the views $w \in \mathcal{V}_{\bar{X}} \cap \mathcal{V}$, and the cartesian product $w^{\text{DB}'} \times \text{dom}(\bar{X})$, for each other view $[w\bar{X}]$. Observe that \mathcal{F} can be embedded in $(\Phi', \mathcal{V}_{\bar{X}})$, too. Then, since each view in $\mathcal{V}_{\bar{X}}$ includes \bar{X} in its sets of variables, it trivially follows that $(\mathcal{F}, \{\bar{X}\})$ can be embedded in $(\Phi', \mathcal{V}_{\bar{X}})$. Note that Φ' has the form $\Phi'' \wedge \bigwedge_{X \in O} \text{atom}_X(\{X\})$, where Φ'' is a subformula of Φ with $\Phi'' \equiv_h \Phi$. Thus, $\Phi'' \wedge \text{atom}_{\bar{X}}(\{\bar{X}\})$ is homomorphically equivalent to $\Phi \wedge \text{atom}_{\bar{X}}(\{\bar{X}\})$. And, of course, $(\mathcal{F}, \{\bar{X}\})$ can be embedded in $(\Phi'', \mathcal{V}_{\bar{X}})$.

Now, note that $\text{DB}'_{\bar{X}}$ is legal for $\mathcal{V}_{\bar{X}}$ w.r.t. Φ and DB . Thus, we are in the position to apply **Compute-Max** over $\Phi, \mathcal{F}, \mathcal{V}_{\bar{X}}, \text{DB}'_{\bar{X}}$, and with $O = \{\bar{X}\}$. By the application of Theorem 4.7 on the modified instance, if **Compute-Max** returns **NO SOLUTION** (resp., **FAIL**), then we can terminate the computation, by returning that there is no solution,

i.e., $\Phi^{\text{DB}} = \emptyset$ (resp., the promise **P1**—and, hence **P2**—is disproved). Therefore, let us assume that we get an assignment $\theta_{\bar{X}}$.

After that $\theta_{\bar{X}}$ is computed, we can update DB' in such a way that $\text{dom}(\bar{X}) = \{\theta_{\bar{X}}\}$, and repeat the process with another variable in O . Eventually, all the values obtained this way for variables in O form an assignment that is returned as output as a solution to MAX. Note that, at each step, **Compute-Max** can disprove the promise **P1** (and, hence **P2**), by returning **FAIL**. Moreover, our procedure may additionally disprove the property **P2**, whenever there is a pair of values $\theta_{\bar{X}}$ and $\theta_{\bar{X}'}$ associated with different maximum weights by **Compute-Max**. Finally, if some subsequent invocation returns **NO SOLUTION** (after that the first invocation did not return **NO SOLUTION** or **FAIL**), then we can conclude that there is no sandwich formula of Φ w.r.t. \mathcal{V} (by the results in [21, 35]), and hence the promise is again disproved. \square

By the above result and the approach by Lawler [36], we obtain the corresponding tractability result for TOP- K .

Theorem A.4. *There is an algorithm that solves TOP- $K(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$ with polynomial delay, if **P2** holds. It outputs **FAIL**, only if **P2** does not hold.*

Proof. Let $\bar{\Phi} = \Phi \wedge \bigwedge_{X \in O} \text{atom}_X(\{X\})$. Update \mathcal{V} by adding the base views associated with such novel atoms. Moreover, update DB and DB' by adding the relations associated with these views, each one containing the whole active domain $\text{dom}(X)$ of any variable $X \in O$. Note that $\Phi^{\text{DB}} = \bar{\Phi}^{\text{DB}}$ and that DB' is legal. By **P2**, \mathcal{F} can be embedded in (Φ', \mathcal{V}) for some subformula Φ' of $\bar{\Phi}$ with $\Phi' \equiv_{\text{h}} \bar{\Phi}$. Then, **P2** clearly holds for $\bar{\Phi}$, too. From Theorem A.3, $\text{MAX}(\bar{\Phi}_{\mathcal{F}}, O, \text{DB})$ can be computed in polynomial time. Therefore, we can apply Lawler's procedure [36] on $\bar{\Phi}$, by using **Compute-Max** as an oracle. In particular, at any step, the oracle will be called with a different constraint database obtained by changing the atom_X relations (as in [10]), and with the base views in the constraint database DB' changed accordingly. Note that, at each step, MAX is still feasible in polynomial time, as we just change the database instance over which it has to be solved, without affecting the structural promise. \square

A.3. Putting It All Together: Computing Certified Solutions

We can now complete the picture, by showing that the above results may be exploited to design an algorithm that always output reliable answers. Indeed, if it is not possible to compute a correct solution, the promise is disproved. In particular, the price of having certified solutions is now to consider the following more stringent promise:

(P0) \mathcal{F} can be embedded in (Φ, \mathcal{V}) .

Theorem 4.8. *There is a polynomial-time algorithm for structured valuation functions that either computes a solution to $\text{MAX}(\Phi_{\mathcal{F}}, O, \text{DB}, \mathcal{V}, \text{DB}')$, or disproves **P0**.*

Proof. Assume that **P0** holds. Then, we derive that \mathcal{F} can be embedded in $(\Phi \wedge \bigwedge_{X \in \text{vars}(\Phi)} \text{atom}_X(\{X\}), \mathcal{V})$, because any tree projection of Φ w.r.t. \mathcal{V} clearly covers every single variable in $\text{vars}(\Phi)$. Then, the promise **P2** holds on $\Phi \wedge \bigwedge_{X \in \text{vars}(\Phi)} \text{atom}_X(\{X\})$ and, thus, we can exploit Theorem A.3 to compute a solution to $\text{MAX}(\Phi_{\mathcal{F}}, \text{vars}(\Phi), \text{DB})$.

Assume that the procedure in the proof of Theorem A.3 succeeds (otherwise **P2** and thus **P0** are disproved). If the result is **NO SOLUTION**, then we are in fact guaranteed that there is no solution at all. Otherwise, let θ be its output. Let v_{max} be the one weight associated with all partial solutions at any invocation of **Compute-Max**. Then, in polynomial time we check that θ is in fact an answer in Φ^{DB} , and that $\mathcal{F}(t) = v_{max}$. If the former check fails, then we conclude that no tree projections exist (and thus no embedding may exist, as well). Otherwise, from the proof of Lemma A.2, $\max_{\mathcal{F}}(\theta[X]) \leq v_{max}$, for each variable $X \in vars(\Phi)$. Therefore, if $\mathcal{F}(\theta) = v_{max}$ we are sure that the result is correct, and else that the promise does not hold. \square

By following the same line of reasoning as in the proof of Theorem A.4, we get the corresponding result for **TOP-K**.

Theorem 4.9. *There is a polynomial-delay algorithm for structured valuation functions that either solves **TOP-K**($\Phi_{\mathcal{F}}, O, DB, \mathcal{V}, DB'$), or disproves that \mathcal{F} can be embedded in (Φ, \mathcal{V}) ; in the latter case, before terminating, it computes a (possibly empty) certified prefix of a solution.*