Constraints (2023) 28:277–319
https://doi.org/10.1007/s10601-023-09343-6

Super-reparametrizations of weighted CSPs: properties and optimization perspective

Tomáš Dlask[1] · Tomáš Werner[1] · Simon de Givry[2]

Accepted: 21 February 2023 / Published online: 16 May 2023

Abstract
The notion of reparametrizations of Weighted CSPs (WCSPs) (also known as equivalence-preserving transformations of WCSPs) is well-known and finds its use in many algorithms to approximate or bound the optimal WCSP value. In contrast, the concept of super-reparametrizations (which are changes of the weights that keep or increase the WCSP objective for every assignment) was already proposed but never studied in detail. To fill this gap, we present a number of theoretical properties of super-reparametrizations and compare them to those of reparametrizations. Furthermore, we propose a framework for computing upper bounds on the optimal value of the (maximization version of) WCSP using super-reparametrizations. We show that it is in principle possible to employ arbitrary (under some technical conditions) constraint propagation rules to improve the bound. For arc consistency in particular, the method reduces to the known Virtual AC (VAC) algorithm. We implemented the method for singleton arc consistency (SAC) and compared it to other strong local consistencies in WCSPs on a public benchmark. The results show that the bounds obtained from SAC are superior for many instance groups.

Keywords Weighted CSP · Super-reparametrization · Linear programming · Constraint propagation

1 Introduction

In the *weighted constraint satisfaction problem (WCSP)* we maximize the sum of (weight) functions over many discrete variables, where each function depends only on a (usually

✉ Tomáš Dlask
dlaskto2@fel.cvut.cz

Tomáš Werner
werner@fel.cvut.cz

Simon de Givry
simon.de-givry@inrae.fr

1 Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Karlovo náměstí 13, 12000 Prague, Czech Republic

2 Université Fédérale de Toulouse, ANITI, INRAE, UR 875, 31320 Castanet-Tolosan, France

small) subset of the variables. A popular approach to tackle this NP-hard combinatorial optimization problem is via its linear programming (LP) relaxation [1–5]. The dual of this LP relaxation [2, 5, 6] can be interpreted as follows. Feasible dual solutions correspond to *reparametrizations* (also known as equivalence-preserving transformations [6]) of the WCSP objective function, which are obtained by moving weights between weight functions so that the WCSP objective function is preserved. The dual LP relaxation then seeks to find such a reparametrization of the initial WCSP that minimizes an *upper bound* on the WCSP objective value by reparametrizations. For some instances, the minimal upper bound is equal to the maximal value of the WCSP objective (i.e., the LP relaxation is tight) but, in general, there is a gap between them. The precise form of the dual LP differs slightly from author to author.

For larger instances, solving the LP relaxation to global optimality is too costly. Therefore, the upper bound is usually minimized suboptimally by performing reparametrizations only locally. Stopping points of these suboptimal methods are usually characterized by various levels of local consistency of the CSP formed by the *active tuples* (i.e., the tuples with the maximum weight in each weight function individually) of the reparametrized WCSP. This is consistent with the fact that a necessary (but not sufficient) condition for global optimality of the dual LP relaxation is that the active-tuple CSP has a non-empty local consistency closure. The level of local consistency at optimum depends on the space of allowed reparametrizations: if weights can move only between pairs of weight functions of which one is unary or nullary, it is arc consistency (AC); if weights can move between two weight functions of any arity, it is pairwise consistency (PWC). These suboptimal methods can be divided into two main classes.

The first class, popular in computer vision and machine learning, is known as *convex message passing* [2, 7–12]. These methods repeat a simple local operation and can be seen as block-coordinate descent with exact updates satisfying the so-called relative interior rule [13]. At fixed points, the active-tuple CSP has non-empty AC (or PWC) consistency closure. These methods yield good upper bounds but are too slow to be applied in each node of a branch-and-bound search.

The second class has been called *soft local consistency* methods in constraint programming [6], due to its similarity to local consistencies in the ordinary CSP. One type of these methods moves only integer weights between weight functions (assuming all initial weights are integer) and is efficient enough to be maintained during search. Its most advanced representant is *existential directional arc consistency (EDAC)* algorithm [14]. The other type allows moving fractional weights, which can lead to better bounds but is more costly, hence usually not suitable to be applied during search. Its representants are the *virtual arc consistency (VAC)* algorithm [6, 15] and the very similar Augmenting DAG algorithm [2, 16, 17]. These methods are based on the following fact: whenever the active-tuple CSP has an empty AC closure, there exists a reparametrization of the WCSP that decreases the upper bound. Thus, each iteration of these algorithms first applies the AC algorithm to the active-tuple CSP and if domain wipe-out occurs, it constructs a dual-improving direction by back-tracking the history of the AC algorithm, and finally reparametrizes the current WCSP by moving along this direction by a suitable step size. The VAC and Augmenting DAG algorithms converge to a non-unique state when the active-tuple CSP has a non-empty AC closure (which is called virtual arc consistency) but are typically faster than convex message passing methods.

In the soft-consistency terminology, global optima of the dual LP relaxation have been called *optimally soft arc consistent (OSAC)* WCSPs [6, 18]. In this sense, EDAC and VAC are relaxations of OSAC. But note that OSAC can no longer be considered a local consistency, since no algorithm using only local operations is known to enforce it[1].

Reparametrizations in general cannot enforce stronger local consistencies of the active-tuple CSP than PWC. This can be seen as follows: if the active-tuple CSP has a non-empty PWC closure but violates some stronger local consistency (hence it is unsatisfiable), there exists no reparametrization that would decrease the upper bound and possibly make the active-tuple CSP satisfy the stronger local consistency. The only way to achieve stronger local consistencies (such as k-consistencies) of the active-tuple CSP by reparametrizations is to introduce new weight functions (of possibly higher arities) and then move weights between these new weight functions and the existing weight functions. This allows constructing a hierarchy of progressively tighter LP relaxations of the WCSP [11, 21–23], including the Sherali-Adams hierarchy [24].

In this paper, we study a different LP-based approach, namely an LP formulation of the WCSP, which was proposed in [25] but never pursued later. It differs from the above well-known LP relaxation and does not belong to the hierarchy of LP relaxations obtained by introducing new weight functions of higher arities. This LP formulation minimizes the same upper bound on the WCSP objective value but this time over *super-reparametrizations* of the initial WCSP objective function, which are changes of the weights that either preserve or increase the WCSP objective value for every assignment. This LP formulation has an exponential number of inequality constraints (representing super-reparametrizations) and is exact, i.e., its minimal value is always equal to the maximal value of the WCSP objective.

We propose to solve this LP suboptimally by a local search method, which is based on the following key observation: whenever the active-tuple CSP is unsatisfiable, there exists a super-reparametrization (but possibly no reparametrization) that decreases the upper bound. The direction of this super-reparametrization is a *certificate of unsatisfiability* of the active-tuple CSP, which can be constructed from the history of the CSP solver. Note that this approach strictly generalizes the VAC algorithm: if the active-tuple CSP has a non-empty AC closure but is unsatisfiable, the VAC algorithm is stuck (because no reparametrization can decrease the upper bound) but our algorithm can decrease the bound by a super-reparametrization. The cost for this greater generality is that super-reparametrizations may preserve neither the WCSP objective value for some assignments nor the set of optimal assignments, but they can nevertheless provide valid, and possibly tighter, upper bounds on the WCSP optimal value.

After formulating this general framework, we focus on the case when the unsatisfiability of the active-tuple CSP is proved by local consistencies stronger than AC/PWC. In particular we use *singleton arc consistency (SAC)*, which is interesting because it does not have bounded support [26] and therefore it would be difficult to achieve by introducing new weight functions of higher arity. We show how to construct a certificate of unsatisfiability of a CSP from the history of the SAC algorithm. Our algorithm then interleaves AC and SAC: we always keep decreasing the upper bound by reparametrizations until the active-tuple CSP has non-empty AC closure, and only then decrease the bound by a super-reparametrization if the SAC closure of active-tuple CSP is found empty. In experiments we show that on many WCSP instances, this algorithm yields better bounds than state-of-the-art soft local consistency methods in reasonable runtime. Note, we report only the achieved upper bounds but do not use them in branch-and-bound search, which would be beyond the scope of our paper.

To the best of our knowledge, super-reparametrizations have not been utilized or studied except for [25] and [27]. In [25], super-reparametrizations were used to obtain tighter

[1] And it is unlikely that such an algorithm exists, since it has been proved [19, 20] that finding global optimum of the LP relaxation of the WCSP is not easier than solving the general linear programming problem.

bounds using a specialized cycle-repairing algorithm and were identified in [27] as a property satisfied by all formulations of the linear programming relaxations based on reparametrizations. However, [27] focuses almost solely on the relation between different formulations of reparametrizations, instead of super-reparametrizations. To fill in this gap, we theoretically analyze the associated optimization problem and also the properties of super-reparametrizations.

Compared to the previous version of this paper [28], we improved the current paper in the following ways:

- Most importantly, we include a study on the theoretical properties of super-reparametrizations and compare them to those of reparametrizations (Section 5).
- Although our implementation remains limited to WCSPs of arity 2, we present all of our theoretical results for WCSPs of any arity.
- We include a geometric interpretation that provides intuitive insights and thus simplifies understanding of our method (Section 4.1.1).
- In addition to making our code publicly available, we add more information on implementation details to improve reproducibility (Section 4.4).
- We also analyze the cone of non-negative weighted CSPs and prove that it is dual to the marginal polytope (Section 3.2).
- Unsurprisingly, we show that some decision problems connected to our approach and super-reparametrizations are NP-hard (in Section 6).

Structure We begin in Section 2 by formally defining the Weighted CSP, classical (crisp) CSP, and introducing the notation that will be used throughout the paper. Then, in Section 3, we formally define the optimization problem of minimizing an upper bound over reparametrizations and/or super-reparametrizations where we also state the sufficient and necessary optimality conditions. Next, Section 4 proposes a practical approach for approximate minimization of the upper bound over super-reparametrizations using constraint propagation. We also give experimental results comparing our approach with existing soft local consistencies. Additional properties of the underlying active-tuple CSPs (see definition later) and the sets of optimal (or also non-optimal) super-reparametrizations are given in Section 5. Section 6 presents the hardness results. We provide a detailed example demonstrating EDAC, VAC, and our proposed approach with SAC in Appendix.

2 Notation

Let V be a finite set of *variables* and D a finite *domain* of each variable. An *assignment* $x \in D^V$ assigns[2] a value $x_i \in D$ to each variable $i \in V$. Let $C \subseteq 2^V$ be a set of non-empty *scopes*, i.e., (V, C) can be seen as an undirected hypergraph. The triplet (D, V, C) defines the *structure* of a (weighted) CSP and will be fixed throughout the paper. By

$$T = \{(S, k) \mid S \in C,\ k \in D^S\} = \bigcup_{S \in C} T_S \quad \text{where} \quad T_S = \{(S, k) \mid k \in D^S\} \tag{1}$$

we denote the set of *tuples*, partitioned into sets T_S, $S \in C$. We say that an assignment $x \in D^V$ *uses* a tuple $t = (S, k) \in T$ if $x[S] = k$ where $x[S]$ denotes the restriction of x onto the set $S \subseteq V$, i.e., for $S = \{i_1, ..., i_{|S|}\}$ we have $x[S] = (x_{i_1}, ..., x_{i_{|S|}})$ (where the order of

[2] As usual, D^V denotes the set of all mappings from V to D, so $x \in D^V$ is the same as $x : V \to D$.

the components is defined by the total order on S inherited from some arbitrary fixed total order on V). Each assignment $x \in D^V$ uses exactly one tuple from each T_S.

An instance of the *constraint satisfaction problem (CSP)* is defined by the quadruple (D, V, C, A) where $A \subseteq T$ is the set of *allowed tuples* (while the tuples $T - A$ are *forbidden*). As the CSP structure (D, V, C) will be always the same, we will refer to the CSP instance only as A (in other words, in the sequel we identify CSP instances with subsets of T). An assignment $x \in D^V$ is a *solution* to a CSP $A \subseteq T$ if it uses only allowed tuples, i.e., $(S, x[S]) \in A$ for all $S \in C$. The set of all solutions to the CSP will be denoted by $\mathrm{SOL}(A) \subseteq D^V$. The CSP is *satisfiable* if $\mathrm{SOL}(A) \neq \emptyset$, otherwise it is *unsatisfiable*.

The *weighted constraint satisfaction problem (WCSP)*[3] seeks to find an assignment $x \in D^V$ that maximizes the function

$$\sum_{S \in C} f_S(x[S]) \tag{2}$$

where $f_S \colon D^S \to \mathbb{R}$, $S \in C$, are given *weight functions*. All the weights (i.e., the values of the weight functions) together can be seen as a vector $f \in \mathbb{R}^T$, such that for $t = (S, k) \in T$ we have $f_t = f_S(k)$. The WCSP instance is defined by the quadruple (D, V, C, f). However, as the structure (D, V, C) will be always the same, we will refer to WCSP instances only as f (in other words, we identify WCSP instances with vectors from $\mathbb{R}^T$).

Example 1 For example, if $V = \{1, 2, 3, 4\}$, $C = \{\{1\}, \{2\}, \{2, 3\}, \{1, 4\}, \{2, 3, 4\}\}$, and $D = \{\mathsf{a}, \mathsf{b}\}$, then we want to maximize the expression

$$f_{\{1\}}(x_1) + f_{\{2\}}(x_2) + f_{\{2,3\}}(x_2, x_3) + f_{\{1,4\}}(x_1, x_4) + f_{\{2,3,4\}}(x_2, x_3, x_4)$$

over $x_1, x_2, x_3, x_4 \in \{\mathsf{a}, \mathsf{b}\}$. We have, e.g.,

$$T_{\{2,3\}} = \{(\{2, 3\}, (\mathsf{a}, \mathsf{a})), (\{2, 3\}, (\mathsf{a}, \mathsf{b})), (\{2, 3\}, (\mathsf{b}, \mathsf{a})), (\{2, 3\}, (\mathsf{b}, \mathsf{b}))\}.$$

Remark 1 In some formalisms [6, 22], the objective (2) is to be minimized. For our purposes, these settings are equivalent and the results for minimization problems are analogous as one can invert the sign of all weights and maximize instead. Next, some papers consider only non-negative weights and the empty (nullary) scope $\emptyset \in C$ whose weight $f_\emptyset$ constitutes a bound on the WCSP optimal value [6, 22]. However, we will later need both positive and negative weights in a WCSP, so we require $\emptyset \notin C$ to simplify notations (also, with both positive and negative weights, $f_\emptyset$ would not yield a bound on the optimal value).

We will use another notation for the WCSP objective, which is common in machine learning, see, e.g., [3, Section 3]. We define an indicator map $\phi \colon D^V \to \{0, 1\}^T$ by

$$\phi_t(x) = [\![x[S] = k]\!] \quad \text{for each } t = (S, k) \in T \tag{3}$$

where $[\![\cdot]\!]$ denotes the *Iverson bracket*, which equals 1 if the logical expression in the bracket is true and 0 if it is false. The WCSP objective (2) can now be written as the dot product

$$\sum_{S \in C} f_S(x[S]) = \sum_{t \in T} f_t \phi_t(x) = \langle f, \phi(x) \rangle. \tag{4}$$

[3] The WCSP is also known under different names, e.g., as the finite-valued CSP [29, 30], discrete energy minimization [31], or maximum a posteriori (MAP) inference in graphical models [5]. It is also the main task in cost function networks [32].

This makes explicit that the WCSP objective is linear in the weight vector f. The WCSP optimal value is

$$\max_{x \in D^V} \langle f, \phi(x) \rangle = \max_{\mu \in M} \langle f, \mu \rangle \tag{5}$$

where

$$M = \phi(D^V) = \{\phi(x) \mid x \in D^V\} \subseteq \{0, 1\}^T. \tag{6}$$

Note that M is defined only by the structure (D, V, C).

3 Bounding the WCSP optimal value

We define the function $B\colon \mathbb{R}^T \to \mathbb{R}$ by

$$B(f) = \sum_{S \in C} \max_{k \in D^S} f_S(k) = \sum_{S \in C} \max_{t \in T_S} f_t. \tag{7}$$

This is a convex piecewise-affine function. For $f \in \mathbb{R}^T$, we call a tuple $t = (S, k) \in T$ *active*[4] if

$$f_t = \max_{t' \in T_S} f_{t'}. \tag{8}$$

The set of all tuples that are active for f is denoted[5] by $A^*(f) \subseteq T$. Note, $A^*(f) \subseteq T$ can be interpreted as a CSP.

Theorem 1 [2] *For every WCSP $f \in \mathbb{R}^T$ and every assignment $x \in D^V$ we have:*

(a) $B(f) \geq \langle f, \phi(x) \rangle$,
(b) $B(f) = \langle f, \phi(x) \rangle$ if and only if $x \in SOL(A^*(f))$.

Proof Statement (a) can be checked by comparing expressions (2) and (7) term by term.

Statement (b) says that $B(f) = \langle f, \phi(x) \rangle$ if and only if $(S, x[S]) \in A^*(f)$ for all $S \in C$. This is again straightforward from (2) and (7). □

Theorem 1 says that $B(f)$ is an *upper bound* on the WCSP optimal value. Moreover, it shows that $B(f) = \langle f, \phi(x) \rangle$ implies that x is a maximizer of the WCSP objective (2).

Example 2 Let $V = \{1, 2\}$, $D = \{\mathsf{a}, \mathsf{b}\}$, and $C = \{\{1\}, \{2\}, \{1, 2\}\}$. For this structure, the set of tuples is

$$\begin{aligned} T = \{&(\{1\}, \mathsf{a}), (\{1\}, \mathsf{b}), (\{2\}, \mathsf{a}), (\{2\}, \mathsf{b}), \\ &(\{1, 2\}, (\mathsf{a}, \mathsf{a})), (\{1, 2\}, (\mathsf{a}, \mathsf{b})), (\{1, 2\}, (\mathsf{b}, \mathsf{a})), (\{1, 2\}, (\mathsf{b}, \mathsf{b}))\}. \end{aligned} \tag{9}$$

For assignment $x=(\mathsf{a}, \mathsf{b}) \in D^V$ (i.e., $x_1 = \mathsf{a}$, $x_2=\mathsf{b}$), we have $\phi(x)=(1, 0, 0, 1, 0, 1, 0, 0) \in \{0, 1\}^T$ where the order of the tuples is given by (9).

[4] Our term 'active tuple' comes from the term 'active inequality'. Indeed, (7) can be calculated as the minimum of $\sum_{S \in C} z_S$ subject to $z_S \geq f_t \ \forall t \in T_S$, where $z_S \in \mathbb{R}$ are auxiliary variables. At optimum, we have $z_S = \max_{t \in T_S} f_t$ and an inequality $z_S \geq f_t$ is active if and only if tuple t is active.

[5] The set $A^*(f)$ corresponds to the notion of Bool(f) in [6]. The characteristic vector of the set $A^*(f)$ was denoted $\bar{f}$ in [2, 9], $\lceil f \rceil$ in [11], and mi$[f]$ in [5].

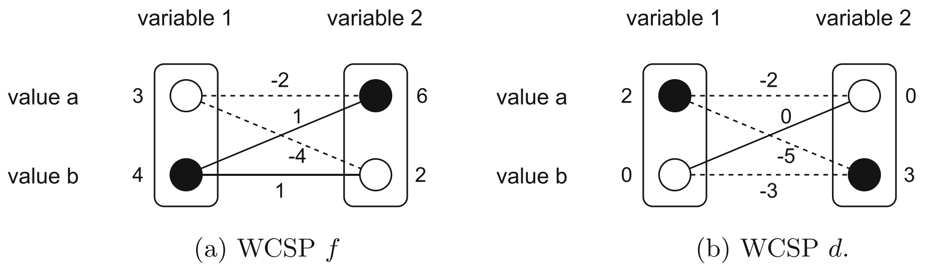


Fig. 1 Visualisations of two WCSPs f and d with structure as in Example 2. Variables (elements of V) are depicted as rounded rectangles, tuples (elements of T) as circles and line segments, and weights f_t (and d_t) are written next to the circles and line segments. Black circles and full lines indicate active tuples, whereas white nodes and dashed lines indicate non-active tuples

An example of a WCSP f with this structure is shown in Fig. 1a. The set of tuples active for f is

$$A^*(f) = \{(\{1\}, \mathsf{b}), (\{2\}, \mathsf{a}), (\{1,2\}, (\mathsf{b}, \mathsf{a})), (\{1,2\}, (\mathsf{b}, \mathsf{b}))\} \tag{10}$$

and the weight vector reads $f = (3, 4, 6, 2, -2, -4, 1, 1) \in \mathbb{R}^T$ (where the ordering is again given by (9)). Thus, the objective value of WCSP f for $x = (\mathsf{a}, \mathsf{b})$ is $\langle f, \phi(x)\rangle = 3+2-4 = 1$. The upper bound equals $B(f) = 4 + 6 + 1 = 11$ and is tight because the CSP $A^*(f)$ is satisfiable (recall Theorem 1). In particular, $\langle f, \phi(\mathsf{b}, \mathsf{a})\rangle = 11$.

3.1 Minimal upper bound over reparametrizations

We say that a WCSP $f \in \mathbb{R}^T$ is *reparametrization* of a WCSP $g \in \mathbb{R}^T$ (also known as an equivalence-preserving transformation of g) [1–3, 5–7, 10, 11, 18] if

$$\langle f, \phi(x)\rangle = \langle g, \phi(x)\rangle \quad \forall x \in D^V. \tag{11}$$

That is, $f - g \in M^\perp$ where

$$M^\perp = \{d \in \mathbb{R}^T \mid \langle d, \mu\rangle = 0\ \forall \mu \in M\} = \{d \in \mathbb{R}^T \mid \langle d, \phi(x)\rangle = 0\ \forall x \in D^V\} \tag{12}$$

is the orthogonal space [33, Chapter 1] of the set (6). Here, 'd' stands for 'direction' but note that any $d \in M^\perp$, as a vector from $\mathbb{R}^T$, can be also seen as a standalone WCSP. The set $M^\perp$ is a subspace of $\mathbb{R}^T$, consisting of all WCSPs that have zero objective value for all assignments. Although $M^\perp$ is defined by an exponential number of equalities in (12), it has a simple, polynomial-sized description (for binary WCSP see [2, §B], for WCSPs of any arity see [11, §3.2]). An example of WCSP $d \in M^\perp$ is in Fig. 1b. The set of all reparametrizations of f is the affine subspace[6] $f + M^\perp = \{f + d \mid d \in M^\perp\}$. Clearly, the binary relation 'is a reparametrization of' (on the set of WCSPs with a fixed structure) is reflexive, transitive and symmetric, hence an equivalence.

Given a WCSP $g \in \mathbb{R}^T$, it is a natural idea to minimize the upper bound on its optimal value by reparametrizations:

$$\min\{B(f) \mid f \text{ is a reparametrization of } g\} \quad = \quad \min_{f \in g + M^\perp} B(f). \tag{13}$$

[6] Note, the symbol '+' in the expression $f + M^\perp$ denotes the sum of a vector and a set of vectors.

By introducing auxiliary variables (as in Footnote 4), this problem can be transformed to a linear program, which is the *dual LP relaxation* of the WCSP g [1, 2, 5, 11]. Every f feasible for (13) satisfies

$$B(f) \geq \langle f, \phi(x) \rangle = \langle g, \phi(x) \rangle \quad \forall x \in D^V, \tag{14}$$

i.e., $B(f)$ is an upper bound on the optimal value $\max_x \langle g, \phi(x) \rangle$ of WCSP g. If inequality (14) holds with equality for some x, then f is optimal for (13) and the LP relaxation is tight. Necessary and sufficient conditions for optimality can be obtained from complementary slackness, see [2, 5, 11].

Problem (13) has been widely studied [1, 2, 4–6, 18, 23] and many approaches for its (approximate) large-scale optimization have been proposed, typically based on block-coordinate descent [2, 7–10, 21, 25, 27] or constraint propagation [2, 6, 16, 22, 34]. If f is optimal for (13), then the CSP $A^*(f)$ has a non-empty pairwise-consistency (PWC) closure (for binary WCSPs, PWC reduces to arc consistency) [11]. We conjecture that PWC is in general the strongest level of local consistency of $A^*(f)$ that can be achieved by reparametrizations without enlarging the WCSP structure (i.e., without introducing new weight functions).

We remark that some approaches [6, 18] achieve only (generalized) arc consistency rather than PWC because they optimize over a subset of all possible reparametrizations corresponding to a subspace of $M^\perp$. In this case, WCSPs f optimal for (13) have been called *optimally soft arc consistent (OSAC)*.

3.2 Minimal upper bound over super-reparametrizations

We say that a WCSP $f \in \mathbb{R}^T$ is a *super-reparametrization*[7] of a WCSP $g \in \mathbb{R}^T$ if

$$\langle f, \phi(x) \rangle \geq \langle g, \phi(x) \rangle \quad \forall x \in D^V. \tag{15}$$

That is, $f - g \in M^*$ where

$$M^* = \{d \in \mathbb{R}^T \mid \langle d, \mu \rangle \geq 0 \; \forall \mu \in M\} = \{d \in \mathbb{R}^T \mid \langle d, \phi(x) \rangle \geq 0 \; \forall x \in D^V\} \tag{16}$$

is the dual cone [33, Chapter 1] to the set (6). It is a polyhedral convex cone, consisting of the WCSPs that have nonnegative objective value for all assignments. This cone contains a line because $M^\perp \subseteq M^*$ and the subspace $M^\perp$ is non-trivial (assuming $|V| > 1$). Precisely, we have $M^* \cap (-M^*) = M^\perp$ where $-M^* = \{-d \mid d \in M^*\}$. The set of all super-reparametrizations of f is the translated cone $f + M^* = \{f + d \mid d \in M^*\}$. For a given $d \in \mathbb{R}^T$, deciding whether $d \notin M^*$ is NP-complete, as shown later in Corollary 2.

The binary relation 'is a super-reparametrization of' (on the set of WCSPs with a fixed structure) induced by the convex cone M^* is reflexive and transitive, hence a preorder. It is not antisymmetric: $f - g \in M^*$ and $g - f \in M^*$ does not imply $f = g$ but merely $f - g \in M^\perp$, i.e., that f is a reparametrization of g. This is because the cone M^* may contain a line, see [35, §2] and [36, §2.4].

Remark 2 The optimal value (5) of a WCSP f can be also written as

$$\max_{\mu \in M} \langle f, \mu \rangle = \max_{\mu \in \operatorname{conv} M} \langle f, \mu \rangle \tag{17}$$

[7] Super-reparametrizations were called *virtual potentials* in [25] and *sup-reparametrizations* in [27].

where conv denotes the convex hull operator [36]. The equality in (17) follows from the well-known fact that a linear function on a polytope attains its maximum in at least one vertex of the polytope [5, Corollary 3.44]. The set $\operatorname{conv} M \subseteq [0, 1]^T$ is known as the *marginal polytope* and has the central role in approaches to WCSP based on linear programming (see [3, 5] and references therein). It is easy to show that

$$M^* = (\operatorname{conv} M)^* = (\operatorname{cone} M)^* \tag{18}$$

where cone denotes the conic hull operator [36] and * the dual cone operator. Thus, (16) can also be seen as the dual cone to the marginal polytope which, to the best of our knowledge, has not been mentioned before.

Following [25], we consider the problem

$$\min \{B(f) \mid f \text{ is a super-reparametrization of } g\} \;=\; \min_{f \in g + M^*} B(f). \tag{19}$$

Again, this can be reformulated as a linear program. Every f feasible for (19) (i.e., every super-reparametrization of g) satisfies

$$B(f) \geq \langle f, \phi(x)\rangle \geq \langle g, \phi(x)\rangle \qquad \forall x \in D^V. \tag{20}$$

The next theorem characterizes optimal solutions:

Theorem 2 *Let f be feasible for (19). The following are equivalent:*

(a) f is optimal for (19).
(b) $B(f) = \max_{x \in D^V} \langle f, \phi(x)\rangle = \max_{x \in D^V} \langle g, \phi(x)\rangle$
(c) CSP $A^*(f)$ has a solution x satisfying $\langle f, \phi(x)\rangle = \langle g, \phi(x)\rangle$.

Proof (a)⇔(b): Denote $m = \max_x \langle g, \phi(x)\rangle$, which by (20) implies $B(f) \geq m$. To see that this bound is attained, define f by $f_t = m/|C|$ for all $t \in T$. It can be checked from (2) and (7) that $B(f) = \langle f, \phi(x)\rangle = m$ for all x, so f is feasible and optimal.

(b)⇒(c): Since every feasible f satisfies (20), (b) implies $B(f) = \langle f, \phi(x)\rangle = \langle g, \phi(x)\rangle$ for some x. By Theorem 1(b), this implies (c).

(c)⇒(b): By Theorem 1(b) together with (20), (c) implies $B(f) = \langle f, \phi(x)\rangle = \langle g, \phi(x)\rangle$ for some x. Statement (b) now follows from Theorem 1(a). □

Theorem 2 in particular says that the optimal value of (19) is equal to the optimal value of WCSP g (this has been observed already in [25, Theorem 1]). As stated in [25], this is not surprising because the complexity of the WCSP is hidden in the exponential set of constraints of (19). Let us remark that for $f \in g + M^*$, deciding whether f is optimal for (19) is NP-complete, as shown later in Corollary 3.

Theorem 2 has a simple corollary:

Theorem 3 *Let $g \in \mathbb{R}^T$. CSP $A^*(g)$ is satisfiable if and only if $B(g) \leq B(f)$ for every $f \in g + M^*$.*

Proof By Theorem 2, $A^*(g)$ is satisfiable if and only if (19) attains its optimum at the point $f = g$, i.e., $B(g) \leq B(f)$ for every $f \in g + M^*$. □

4 Iterative method to improve the bound by super-reparametrizations

In this section we present an iterative method to suboptimally solve (19). Starting from a feasible solution to (19), every iteration finds a new feasible solution with a lower objective, which by (20) corresponds to decreasing the upper bound on the optimal value of the initial WCSP.

4.1 Outline of the method

Consider a WCSP f feasible for (19), i.e., $f \in g + M^*$. By Theorem 2, a necessary (but not sufficient) condition for f to be optimal for (19) is that CSP $A^*(f)$ is satisfiable. By Theorem 3, $A^*(f)$ is satisfiable if and only if $B(f) \leq B(f')$ for all $f' \in f + M^*$. In summary, we have the following implications and equivalences:

$$\begin{tikzcd} f \text{ is optimal for (19)} \arrow[r, Rightarrow] \arrow[d, Leftrightarrow] & \text{CSP } A^*(f) \text{ is satisfiable} \arrow[d, Leftrightarrow] \\ B(f) \leq B(f') \;\; \forall f' \in g + M^* \arrow[r, Rightarrow] & B(f) \leq B(f') \;\; \forall f' \in f + M^* \end{tikzcd} \tag{21}$$

The left-hand equivalence is just the definition of the optimum of (19), the right-hand equivalence is Theorem 3, and the top implication follows from Theorem 2. The bottom implication independently follows from transitivity of super-reparametrizations, which says that $f' \in f + M^*$ implies $f' \in g + M^*$ (assuming $f \in g + M^*$).

Suppose for the moment that we have an oracle that, for a given $f \in \mathbb{R}^T$, decides if $A^*(f)$ is satisfiable and if it is not, finds some $f' \in f + M^*$ such that $B(f') < B(f)$ (which exists by Theorem 3). By transitivity of super-reparametrizations, such f' is feasible for (19). This suggests an iterative scheme to improve feasible solutions to (19). We initialize $f^0 := g$ and then for $k = 0, 1, 2, \ldots$ repeat the following iteration:

```
If CSP $A^*(f^k)$ is satisfiable, stop.
Otherwise, find $f^{k+1} \in f^k + M^*$ such that $B(f^{k+1}) < B(f^k)$.
```

Note that transitivity of super-reparametrizations implies $f^k \in f^0 + M^*$ for every k, so every f^k is feasible for (19) as expected. An example of a single iteration is shown in Fig. 2a and b.

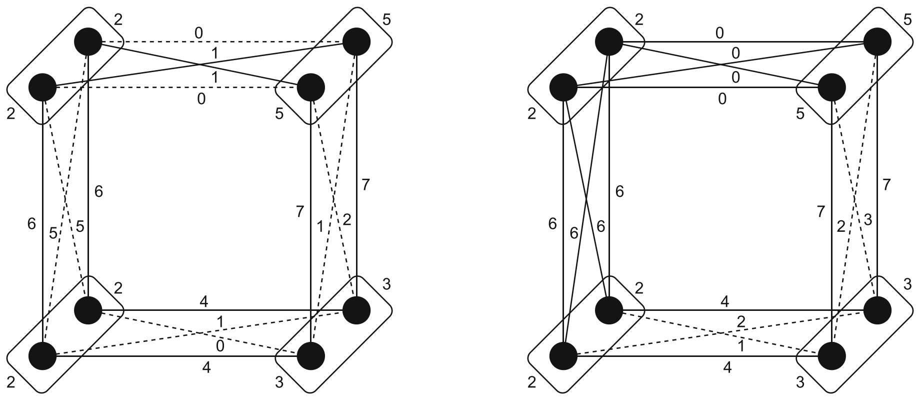
This iterative method belongs to the class of local search methods to solve (19): having a current feasible estimate f^k, we search for the next estimate f^{k+1} with a strictly better objective within a neighborhood $f^k + M^*$ of f^k. We can define *local optima* of (19) with respect to this method to be super-reparametrizations f of g such that $A^*(f)$ is satisfiable.

4.1.1 Properties of the method

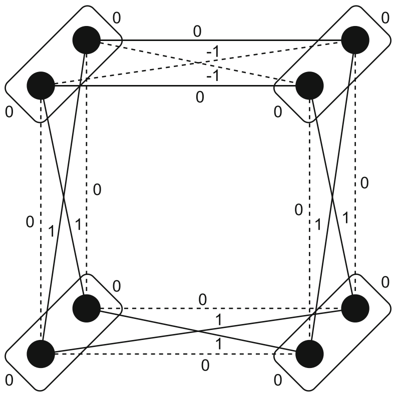
By transitivity of super-reparametrizations, for every k we have

$$f^{k+1} + M^* \subseteq f^k + M^* \tag{22}$$

which holds with equality if and only if $f^{k+1} \in f^k + M^{\perp}$ (i.e., f^{k+1} is a reparametrization of f^k). This shows that the search space of the method may shrink with increasing k, in other words, a larger and larger part of the feasible set $f^0 + M^*$ of (19) is cut off and becomes



(a) WCSP f^0, $A^*(f^0)$ unsatisfiable. (b) WCSP f^1, $B(f^1) < B(f^0)$.



(c) Certificate d of unsatisfiability of $A^*(f^0)$, $f^1 = f^0 + d$.

Fig. 2 Example of one iteration on a binary WCSP whose (hyper)graph is a cycle of length 4

forever inaccessible. If, for some k, all (global) optima of (19) happen to lie in the cut-off part, the method has lost any chance to find a global optimum. This is illustrated in Fig. 3.

This has the following consequence. Every f^k satisfies

$$B(f^k) \quad \geq \min_{f \in f^k + M^*} B(f) = \max_{x \in D^V} \langle f^k, \phi(x) \rangle. \tag{23}$$

In every iteration, the left-hand side of inequality (23) decreases and the right-hand side increases or stays the same due to (22). If both sides meet for some k, the CSP $A^*(f^k)$ becomes satisfiable by Theorem 1(b) and the method stops. Monotonic increase of the right-hand side can be seen as 'greediness' of the method: if we could choose f^{k+1} from the initial feasible set $f^0 + M^*$ rather than from its subset $f^k + M^*$, the right-hand side could also decrease. Any increase of the right-hand side is undesirable because the bounds $B(f^k)$ in future iterations will never be able to get below it. This is illustrated in Figs. 4 and 5. Unlike in (13), note that not every optimal assignment for WCSP f is optimal for WCSP g. We will return to this in Section 5.

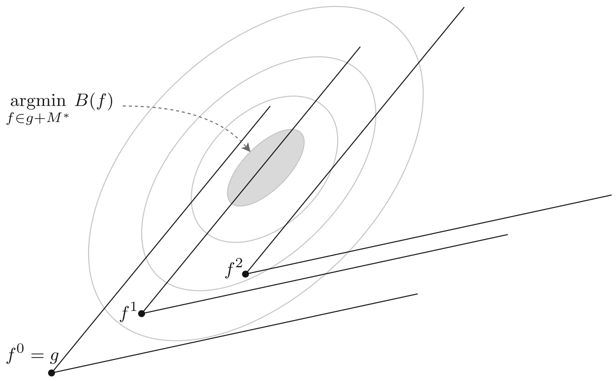


Fig. 3 The shrinking of the search space of the iterative method. The figure illustrates the translated cones $f^i + M^*$ and several contours of the objective $B(f)$. After the second iteration, all global minima of the original problem (marked in grey) become inaccessible as the right hand side of (23) increases

If $A^*(f^k)$ is unsatisfiable, there are usually many vectors $f^{k+1} \in f^k + M^*$ satisfying $B(f^{k+1}) < B(f^k)$. We should choose among them the one that does not cause 'too much' shrinking of the search space and/or increase of the right-hand side of (23). Inclusion (22) holds with equality if and only if $f^{k+1} \in f^k + M^\perp$, so whenever possible we should choose f^{k+1} to be a reparametrization (rather than just a super-reparametrization) of f^k. Unfortunately, we know of no other useful theoretical results to help us choose f^{k+1}, so we are left with heuristics. One natural heuristic is to choose f^{k+1} such that the vector $f^{k+1} - f^k$ is sparse (i.e., has only a small number of non-zero components) and its positive components are small. Unfortunately, this can sometimes be too restrictive because, e.g., vectors from $M^\perp$ can be dense and their components have unbounded magnitudes.

4.1.2 Employing constraint propagation

So far we have assumed we can always decide if CSP $A^*(f)$ is satisfiable. This is unrealistic because the CSP is NP-complete. Yet the approach remains applicable even if we

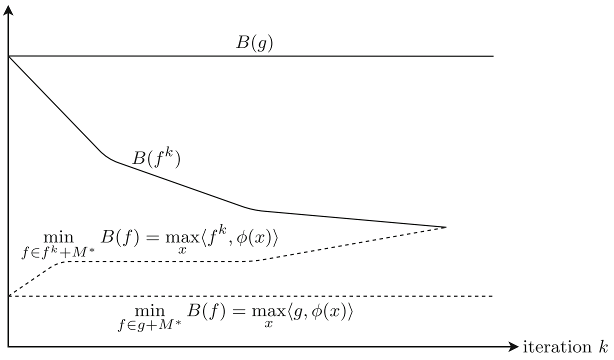


Fig. 4 Illustration to the iterative scheme: $B(g)$ and $B(f^k)$ are shown by the full lines, $\max_x \langle g, \phi(x)\rangle$ and $\max_x \langle f^k, \phi(x)\rangle$ are represented by the dashed lines

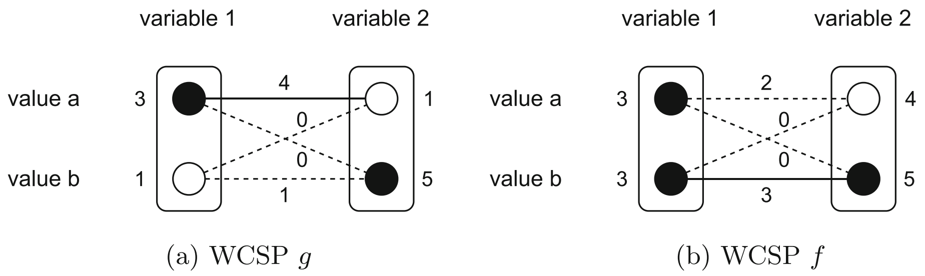


Fig. 5 WCSP f is a super-reparametrization of WCSP g and this pair of WCSPs satisfies $B(f) = 11 < B(g) = 12$ and $\max_{x \in D^V} \langle f, \phi(x) \rangle = 11 > \max_{x \in D^V} \langle g, \phi(x) \rangle = 8$. Assignment $x = (\mathsf{b}, \mathsf{b})$ is not optimal for g despite that $B(f) = \langle f, \phi(x) \rangle$. The fact that f is a super-reparametrization of g can be verified by computing the objective value for each assignment, e.g., for assignment $x = (\mathsf{a}, \mathsf{a})$ we have $\langle g, \phi(x) \rangle = 3 + 4 + 1 = 8 \leq \langle f, \phi(x) \rangle = 3 + 2 + 4 = 9$

detect unsatisfiability of $A^*(f)$ only sometimes, e.g., using constraint propagation. Then our iteration changes to:

```
Try to prove that CSP $A^*(f^k)$ is unsatisfiable.
If we succeed, find $f^{k+1} \in f^k + M^*$ such that $B(f^{k+1}) < B(f^k)$.
If we fail, stop.
```

In this case, stopping points of the method will be even weaker local minima of (19), but they nevertheless might be still non-trivial and useful.

In the sequel we develop this approach in detail. In particular we show, if $A^*(f^k)$ is unsatisfiable, how to find a vector $f^{k+1} \in f^k + M^*$ satisfying $B(f^{k+1}) < B(f^k)$. We will do it in two steps. First (in Section 4.2), given the CSP $A^*(f^k)$ we find a direction $d \in M^*$ using constraint propagation. This direction is a *certificate of unsatisfiability* of the CSP $A^*(f^k)$ and, at the same time, an *improving direction* for (19). Second (in Section 4.3), given d and f^k, we find a *step size* $\alpha > 0$ such that $f^{k+1} = f^k + \alpha d$ and $B(f^k) > B(f^{k+1})$. An example of such a certificate of unsatisfiability is shown in Fig. 2c.

4.1.3 Relation to existing approaches

The Augmenting DAG algorithm [2, 16] and the VAC algorithm [6] are (up to the precise way of computing certificates d and step sizes α) an example of the described approach, which uses arc consistency to attempt to prove unsatisfiability of $A^*(f^k)$. In this favorable case, there exist certificates $d \in M^\perp$, so we are, in fact, applying local search to (13) rather than (19). For stronger local consistencies, such certificates, in general, do not exist (i.e., inevitably $\langle d, \phi(x) \rangle > 0$ for some x).

The algorithm proposed in [25] can be also seen as an example of our approach. It interleaves iterations using arc consistency (in fact, the Augmenting DAG algorithm) and iterations using cycle consistency.

As an alternative to our approach, stronger local consistencies can be achieved by introducing new weight functions (of possibly higher arity) into the WCSP objective (2) and minimizing an upper bound by reparametrizations, as in [11, 21–23, 37]. In our particular case, after each update $f^{k+1} = f^k + \alpha d$ we could introduce a new weight function with scope

$$S' = \bigcup \{S \mid (S,k) \in T,\ d_S(k) \neq 0\} \tag{24}$$

and weights

$$f_{S'}(k) = -\alpha \sum_{\substack{S \in C \\ S \subseteq S'}} d_S(k[S]) \tag{25}$$

where $k \in D^{S'}$. Notice that such an added weight function would not increase the bound (7) since its weights are non-positive due to the fact that it needs to decrease the objective value for some assignments. In this view, our approach can be seen as enforcing stronger local consistencies but omitting these compensatory higher-order weight functions, thus saving memory.

Finally, the described approach can be seen as an example of the primal-dual approach [38] to optimize linear programs using constraint propagation. In detail, [38] proposed to construct the complementary slackness system for a given feasible solution and apply constraint propagation to detect if the system is satisfiable. If it is not satisfiable, this implies the existence of a certificate of unsatisfiability that can be used to improve the current solution. In our particular case, if (19) is formulated as a linear program, then the complementary slackness conditions (expressed in terms of the dual variables) are equivalent to the optimality conditions stated in Theorem 2 expressed as a set of linear equalities with an exponential number of non-negative variables. Applying constraint propagation on this system is in correspondence with constraint propagation on a CSP.

4.2 Certificates of unsatisfiability of CSP

Constraint propagation[8] is an iterative algorithm, which in each iteration (executed by a *propagator*) infers that some allowed tuples $R \subseteq A$ of a current CSP $A \subseteq T$ can be forbidden without changing its solution set, i.e., $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$, and forbids these tuples, i.e., sets $A := A - R$. The algorithm terminates when it is no longer able to forbid any tuples (in which case the propagator returns $R = \emptyset$) or when it becomes explicit that the current CSP is unsatisfiable. The former usually happens when the CSP achieves some *local consistency* level Φ. The latter happens if $A \cap T_S = \emptyset$ for some $S \in C$, which implies unsatisfiability of A because[9] every assignment has to use one tuple from each T_S.

In this section, we show how to augment constraint propagation so that if it proves a CSP unsatisfiable, it also provides its *certificate of unsatisfiability* $d \in M^*$. This certificate is needed as an improving direction for (19), as was mentioned in Section 4.1.2. First, in Section 4.2.1, we introduce a more general concept, *deactivating directions*. One iteration of constraint propagation constructs an R-deactivating direction for the current CSP A, which certifies that $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$. Then, in Section 4.2.2, we show how to *compose* the deactivating directions obtained from individual iterations of constraint propagation to

[8] We speak only about constraint propagation but the approach outlined in this section is applicable to any method that proves unsatisfiability of a CSP by iteratively forbidding subsets of tuples. In theory, as a stronger alternative one could also use any CSP solver that is augmented to provide a certificate of unsatisfiability (which is always possible, as we will discuss later in this section).

[9] If $|S| = 1$, this event is often called a 'domain wipe-out'.

a single deactivating direction for the initial CSP. If the initial CSP has been proved unsatisfiable by the propagation, this composed deactivating direction is then its certificate of unsatisfiability.

4.2.1 Deactivating directions

Definition 1 Let $A \subseteq T$ and $R \subseteq A$, $R \neq \emptyset$. An *R-deactivating direction for CSP A* is a vector $d \in M^*$ satisfying

(a) $d_t < 0$ for all $t \in R$,
(b) $d_t = 0$ for all $t \in A - R$.

For fixed A and R, all R-deactivating directions for A form a convex cone. Here, we show one way of constructing a deactivating direction:

Theorem 4 *Let $R \subseteq A \subseteq T$ be such that $SOL(A) = SOL(A - R)$ and $R \neq \emptyset$. Denote*[10]

$$\delta = |\{S \in C \mid T_S \cap R \neq \emptyset\}|. \tag{26}$$

Then vector $d \in \mathbb{R}^T$ with components

$$d_t = \begin{cases} -1 & \textit{if } t \in R \\ \delta & \textit{if } t \in T - A \\ 0 & \textit{otherwise (i.e., } t \in A - R\textit{)} \end{cases} \tag{27}$$

is an R-deactivating direction for A.

Proof Conditions (a) and (b) of Definition 1 are clearly satisfied, so it only remains to show that $d \in M^*$. We have

$$\langle d, \phi(x)\rangle = \sum_{t \in T} d_t \phi_t(x) = \sum_{t \in R} -\phi_t(x) + \sum_{t \in T-A} \delta\phi_t(x) = -n_1(x) + \delta n_2(x) \tag{28}$$

where $n_1(x) = |\{S \in C \mid (S, x[S]) \in R\}|$ and $n_2(x) = |\{S \in C \mid (S, x[S]) \in T - A\}|$.

For contradiction, let $x \in D^V$ satisfy $\langle d, \phi(x)\rangle < 0$. This implies $n_1(x) > 0$ and $n_2(x) = 0$, where the latter is because $n_1(x) \leq \delta$ by the definition of δ. That is, we have $(S^*, x[S^*]) \in R$ for some $S^* \in C$ and $(S, x[S]) \in A$ for all $S \in C$. But the latter means $x \in \mathrm{SOL}(A)$ and the former implies $x \notin \mathrm{SOL}(A - R)$, a contradiction. □

Theorem 5 *Let $A \subseteq T$ and $R \subseteq A$. If there exists an R-deactivating direction for A, then $SOL(A) = SOL(A - R)$.*

Proof Observe that $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$ is equivalent to $\mathrm{SOL}(A) \subseteq \mathrm{SOL}(A - R)$ because forbidding tuples may only remove solutions, i.e., SOL is an isotone map (see Section 5.1).

Let d be an R-deactivating direction for A and let $x \in \mathrm{SOL}(A) - \mathrm{SOL}(A - R)$, so $(S, x[S]) \in R$ for some $S \in C$. By (4), we have $\langle d, \phi(x)\rangle < 0$ because $d_S(x[S]) = 0$ for all $(S, x[S]) \in A - R$ by condition (b) in Definition 1 and $d_S(x[S]) < 0$ for all $(S, x[S]) \in R$ by condition (a). This contradicts $d \in M^*$. □

[10] The quantity $\delta > 0$ is the number of scopes S such that T_S contains at least one tuple from R. In other words, for every assignment $x \in D^V$, $(S, x[S]) \in R$ holds for at most δ scopes. We remark that the value of δ could be in some cases decreased, thus decreasing also the objective values $\langle d, \phi(x)\rangle$. However, deciding whether (27) is not an R-deactivating direction for A for a given value δ and A is an NP-complete problem, see Theorem 16 .

Combining Theorems 4 and 5 yields that for any $R \subseteq A$ with $R \neq \emptyset$, an R-deactivating direction for A exists if and only if $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$. Thus, any R-deactivating direction for A is a *certificate* of the fact that $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$.

Unfortunately, vectors d calculated naively by (27) can have many non-zero components, which is undesirable as explained in Section 4.1.1. However, it is clear from Definition 1 that if $A \subseteq A' \subseteq T$ and d is an R-deactivating direction for A', then d is an R-deactivating direction also for A. Moreover, (27) shows that larger sets A give rise to sparser vectors d. This offers us a possibility to obtain a sparser R-deactivating direction for A if we can provide a superset $A' \supseteq A$ of the allowed tuples satisfying $\mathrm{SOL}(A') = \mathrm{SOL}(A' - R)$.

Given $A \subseteq T$ and $R \subseteq A$, finding a maximal (w.r.t. the partial ordering by inclusion) superset $A' \supseteq A$ such that $\mathrm{SOL}(A') = \mathrm{SOL}(A' - R)$ is closely related to finding a minimal unsatisfiable core[11] of an unsatisfiable CSP. While finding a maximal such subset is very likely intractable,[12] for obtaining a 'sparse enough' vector d it suffices to find a 'large enough' such superset A'. Such a superset is often cheaply available as a side result of executing the propagator. Namely, we take $A' = T - P$ where P is the set of forbidden tuples that were visited during the run of the propagator. Clearly, tuples not visited by the propagator could not be needed to infer $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$. Note that P need not be the same for each CSP instance, even for a fixed level of local consistency: for example, if the arc consistency closure of A is empty, then A is unsatisfiable but a domain wipe-out may occur sooner or later depending on A, which affects which tuples needed to be visited.

Let us emphasize that an R-deactivating direction for A need not be always obtained using formula (27), any other method can be used as long as d satisfies Definition 1. We will now give examples of deactivating directions corresponding to some popular constraint propagation rules. In these examples, we assume that our CSP contains all unary constraints (i.e., $\{i\} \in C$ for each $i \in V$), so that rather than deleting domain values we can forbid tuples of the unary constraints.

Example 3 Let us consider (generalized) *arc consistency* (AC). A CSP A is (G)AC if for all $S \in C$, $i \in S$ and $k \in D$ we have the equivalence[13]

$$(\{i\}, k) \in A \quad \Longleftrightarrow \quad (\exists l \in D^S \colon (S, l) \in A,\ l_i = k). \tag{29}$$

If, for some $S \in C$, $i \in S$ and $k \in D$, the left-hand statement in (29) is true and the right-hand statement is false, the AC propagator infers $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$ where $R = \{(\{i\}, k)\}$. To infer this, it suffices to know that the tuples $P = \{(S, l) \mid l \in D^S,\ l_i = k\}$ are all forbidden. An R-deactivating direction d for A can be chosen as in (27) where $\delta = |\{S' \in C \mid T_{S'} \cap R \neq \emptyset\}| = 1$ and A is replaced by $T - P$. Note that then we have $d \in M^\perp$.

If the left-hand statement in (29) is false and the right-hand statement is true, the AC propagator infers $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$ where $R = \{(S, l) \mid l \in D^S,\ l_i = k\} \cap A$. To

[11] Given an unsatisfiable CSP $A \subseteq T$, finding a maximal set $A' \supseteq A$ such that A' is still unsatisfiable corresponds to finding a minimally unsatisfiable set of tuples [39]. This is a finer-grained (tuple-based rather than constraint-based) version of finding a minimal unsatisfiable core of a CSP [40]. Note that we are looking here for a *maximal* superset A' in contrast to a *minimal* unsatisfiable core because we define CSP instances by *allowed* tuples while cores are CSP instances defined by *forbidden* tuples.

[12] The problem of finding a minimal unsatisfiable core has been designated in [40] to be 'highly intractable' based on results from [41].

[13] Note, for convenience we use a slightly unusual definition of arc consistency, allowing to restrict not only domains but also constraint relations. This definition was also considered in [5, §6] or [11].

infer this, it suffices to know that the tuple $P = \{(\{i\}, k)\}$ is forbidden. In this particular case, rather than using (27) (with A replaced by $T - P$), it is better to choose d as

$$d_t = \begin{cases} -1 & \text{if } t \in \{(S, l) \mid l \in D^S,\ l_i = k\} \\ 1 & \text{if } t \in P \\ 0 & \text{otherwise} \end{cases} . \tag{30}$$

Vector (30) satisfies $d \in M^\perp$, in contrast to vector (27) which satisfies only $d \in M^*$. Thus, the update $f^{k+1} = f^k + \alpha d$ is a mere reparametrization, which is desirable as explained in Section 4.1.1.

We note that reparametrizations considered in the previous paragraphs correspond to soft arc consistency operations *extend* and *project* in [6].

Example 4 We now consider *cycle consistency* as defined in [25].[14] As this local consistency was defined only for binary CSPs, we assume that $|S| \le 2$ for each $S \in C$ and denote $E = \{S \in C \mid |S| = 2\}$, so that (V, E) is an undirected graph. Let $\mathcal{L}$ be a (polynomially sized) set of cycles in the graph (V, E). A CSP A is cycle consistent w.r.t. $\mathcal{L}$ if for each tuple $(\{i\}, k) \in A$ (where $i \in V$ and $k \in D$) and each cycle $L \in \mathcal{L}$ that passes through node $i \in V$, there exists an assignment x with $x_i = k$ that uses only allowed tuples in cycle L. It can be shown that the cycle repair procedure in [25] constructs a deactivating direction whenever an inconsistent cycle is found. Moreover, the constructed direction in this case coincides with (27) where A is replaced by $T - P$ for a suitable set P that contains a subset of the forbidden tuples within the cycle.

Example 5 Recall that a CSP A is *singleton arc consistent (SAC)* if for every tuple $t = (\{i\}, k) \in A$ (where $i \in V$ and $k \in D$), the CSP[15] $A|_{x_i=k} = A - (T_{\{i\}} - \{(\{i\}, k)\})$ has a non-empty arc-consistency closure. Good (i.e., sparse) deactivating directions for SAC can be obtained as follows. For some $(\{i\}, k) \in A$, we enforce arc consistency of CSP $A|_{x_i=k}$, during which we store the causes for forbidding each tuple. If $A|_{x_i=k}$ is found to have empty AC closure, we backtrack and identify only those tuples which were necessary to prove the empty AC closure. These tuples form the set P. The deactivating direction is then constructed as in (27) where $R = \{(\{i\}, k)\}$ and A is replaced by $T - P$. Note that SAC does not have bounded support as many other local consistencies [26] do, so the size of P can be significantly different for different CSP instances. We show a detailed example of constructing a deactivating direction using SAC in Appendix.

4.2.2 Composing deactivating directions

Consider now a propagator which, for a current CSP $A \subseteq T$, returns a set $R \subseteq A$ such that $\mathrm{SOL}(A) = \mathrm{SOL}(A - R)$ and an R-deactivating direction for A. This propagator is applied iteratively, each time forbidding a different set of tuples, until the current CSP achieves the desired local consistency level Φ or it becomes explicit that the CSP is unsatisfiable. This is outlined in Algorithm 1, which stores the generated sets R_i of tuples being forbidden and the corresponding R_i-deactivating directions d^i. By line 5 of the algorithm, we have

[14] This is different from *cyclic consistency* as defined in [34]. E.g., reparametrizations are sufficient to enforce cyclic consistency, whereas super-reparametrizations are needed for cycle consistency. Cycle consistency is not common in the constraint programming community. It can be shown that if the graph (V, E) is complete, then path inverse consistency [42, 43] corresponds to cycle consistency w.r.t. all cycles of length 3.

[15] This can be also stated as $A|_{x_i=k} = A - \{(\{i\}, k') \mid k' \in D - \{k\}\}$. In other words, the solutions of the CSP $A|_{x_i=k}$ are the solutions x to CSP A satisfying $x_i = k$. This notation is used, e.g., in [44].

$A_i = A - \bigcup_{j=0}^{i-1} R_j$ for every $i \in \{0, \ldots, n+1\}$. Therefore, by Theorem 5, we have $\mathrm{SOL}(A) = \mathrm{SOL}(A_1) = \mathrm{SOL}(A_2) = \cdots = \mathrm{SOL}(A_{n+1})$, which implies that if A_{n+1} is unsatisfiable then so is A.

Algorithm 1 The procedure `propagate` applies constraint propagation to CSP $A \subseteq T$ and returns the sequence $(R_i)_{i=0}^{n}$ of tuple sets that were forbidden and the corresponding deactivating directions $(d^i)_{i=0}^{n}$. If all tuples in some scope $S \in C$ become forbidden during propagation, `propagate` returns also S, otherwise it returns $S = \emptyset$.

```
1: procedure $(S, (R_i)_{i=0}^{n}, (d^i)_{i=0}^{n}) = \texttt{propagate}(A)$
2: Initialize $n := 0$, $A_0 := A$.
3: while $A_n$ is not $\Phi$-consistent do
4:     Find a set $R_n \subseteq A_n$ and an $R_n$-deactivating direction $d^n$ for $A_n$.
5:     $A_{n+1} := A_n - R_n$
6:     if $\exists S \in C\colon A_{n+1} \cap T_S = \emptyset$ then
7:         return $(S, (R_i)_{i=0}^{n}, (d^i)_{i=0}^{n})$
8:     $n := n + 1$
9: return $(\emptyset, (R_i)_{i=0}^{n-1}, (d^i)_{i=0}^{n-1}))$
```

In this section, we show how to compose the generated sequence of R_i-deactivating directions d^i for A_i into a single $\left(\bigcup_{i=0}^{n} R_i\right)$-deactivating direction for A. This can be done using the following composition rule:

Theorem 6 *Let $A \subseteq T$ and $R, R' \subseteq A$ where $R \cap R' = \emptyset$. Let d be an R-deactivating direction for A. Let d' be an R'-deactivating direction for $A - R$. Let*

$$\delta = \begin{cases} 0 & \text{if } d'_t \le -1 \text{ for all } t \in R, \\ \max\{(-1 - d'_t)/d_t \mid t \in R,\ d'_t > -1\} & \text{otherwise.} \end{cases} \tag{31}$$

Then $d'' = d' + \delta d$ is an $(R \cup R')$-deactivating direction for A.

Proof First, if $d'_t \le -1$ for all $t \in R$, then $d'' = d'$ satisfies the required condition immediately. Otherwise, $\delta > 0$ since $d_t < 0$ for all $t \in R$ by definition and $-1 - d'_t < 0$ due to $d'_t > -1$ in the definition of δ. We will show that d'' satisfies the conditions in Definition 1.

For $t \in R$ with $d'_t \le -1$, $d''_t = d'_t + \delta d_t < d'_t \le -1$ because $\delta d_t < 0$. If $t \in R$ and $d'_t > -1$, then $\delta \ge (-1 - d'_t)/d_t$, so $d''_t = d'_t + \delta d_t \le -1$. Summarizing, we have $d''_t < 0$ for all $t \in R$.

For $t \in R'$, $d'_t < 0$ and $d_t = 0$ holds by definition due to $R' \subseteq A - R$, thus $d''_t = d'_t + \delta d_t = d'_t < 0$ which together with the previous paragraph yields condition (a).

Due to $A - R \supseteq (A - R) - R' = A - (R \cup R')$, for any $t \in A - (R \cup R')$ we have $d_t = 0$ and $d'_t = 0$, which implies $d''_t = d' + \delta d = 0$, thus verifying condition (b).

Finally, we have $d'' \in M^*$ because $d, d' \in M^*$ and $\delta \ge 0$. □

Theorem 6 allows us to combine R_i-deactivating direction d^i for $A_i = A_{i-1} - R_{i-1}$ with R_{i-1}-deactivating direction d^{i-1} for A_{i-1} into a single $(R_{i-1} \cup R_i)$-deactivating direction for A_{i-1}. Iteratively, we can thus gradually build a $\left(\bigcup_{i=0}^{n} R_i\right)$-deactivating direction for A, which certifies unsatisfiability of A whenever Algorithm 1 detects on line 6 that A_{n+1} (and thus also A) is unsatisfiable.

However, it is not always necessary to construct a full $\left(\bigcup_{i=0}^{n} R_i\right)$-deactivating direction because not every iteration of constraint propagation may have been necessary to prove unsatisfiability of A. Instead, we can use the scope $S \in C$ satisfying $A_{n+1} \cap T_S = \emptyset$

(where $A_{n+1} = A - \bigcup_{i=0}^n R_i$, as mentioned above) returned by Algorithm 1 on line 7 and construct an R^*-deactivating direction d^* for a (usually smaller) set $R^* \subseteq \bigcup_{i=0}^n R_i$ such that $(A - R^*) \cap T_S = \emptyset$. Such a direction d^* still certifies unsatisfiability of A and can be sparser and/or may have lower objective values $\langle d^*, \phi(x)\rangle$ than a $\left(\bigcup_{i=0}^n R_i\right)$-deactivating direction, which is desirable as explained in Section 4.1.1.

This is outlined in Algorithm 2, which composes only a subsequence of directions d^i based on a given set of indices $I \subseteq \{0, \dots, n\}$ and constructs an R^*-deactivating direction with $R^* \supseteq \bigcup_{i \in I} R_i$. Although Algorithm 2 is applicable for any set I, in our case I is obtained by taking a scope $S \in C$ such that $A_{n+1} \cap T_S = \emptyset$ and then setting

$$I = \{i \in \{0, \dots, n\} \mid R_i \cap T_S \neq \emptyset\} \tag{32}$$

so that $(A - R^*) \cap T_S = \emptyset$ due to the following fact:

Proposition 1 *Let $S \in C$ be such that $(A - \bigcup_{i=0}^n R_i) \cap T_S = \emptyset$. Let I be given by (32). Then $(A - \bigcup_{i \in I} R_i) \cap T_S = \emptyset$.*

Proof For any sets $A, R, T' \subseteq T$ we have $(A - R) \cap T' = (T' - R) \cap A$. In particular, $(A - \bigcup_{i=0}^n R_i) \cap T_S = (T_S - \bigcup_{i=0}^n R_i) \cap A$. But $T_S - \bigcup_{i=0}^n R_i = T_S - \bigcup_{i \in I} R_i$ because for each $i \notin I$ we have $R_i \cap T_S = \emptyset$ which is equivalent to $T_S - R_i = T_S$. □

Algorithm 2 The procedure `compose` takes the sequences $(R_i)_{i=0}^n$ and $(d^i)_{i=0}^n$ (generated by the procedure `propagate`) and a non-empty index set $I \subseteq \{0, \dots, n\}$ and composes them to an R^*-deactivating direction d^* for A.

```
1: procedure $(R^*, d^*) = \texttt{compose}((R_i)_{i=0}^n, (d^i)_{i=0}^n, I)$
2: Initialize $i := \max I$, $d^* := d^i$, $R^* := R_i$.
3: while $i > 0$ do
4:     $i := i - 1$
5:     if $i \in I$ or $\exists t \in R_i : d^*_t \neq 0$ then
6:         $d^* := d^* + \delta d^i$ (where $\delta$ is (31) with $d', d, R$ replaced by $d^*, d^i, R_i$)
7:         $R^* := R^* \cup R_i$
8: return $(R^*, d^*)$
```

Correctness of Algorithm 2 is given by the following theorem:

Proposition 2 *Algorithm 2 returns an R^*-deactivating direction d^* for A where $\bigcup_{i \in I} R_i \subseteq R^* \subseteq \bigcup_{i=0}^n R_i$.*

Proof The fact that $R^* \supseteq \bigcup_{i \in I} R_i$ is obvious due to $R_{\max I} \subseteq R^*$ by initialization on line 2 and $R_i \subseteq R^*$ for any $i \in I$ such that $i < \max I$ because in such case the update on line 7 is performed. Similarly, $R^* \subseteq \bigcup_{i=0}^n R_i$ holds by initialization of R^* on line 2 and updates on line 7.

It remains to show that d^* is R^*-deactivating, which we will do by induction. We claim that vector d^* is always R^*-deactivating direction for A_i on line 3 and R^*-deactivating direction for A_{i+1} on line 5.

Initially, we have $d^* = d^i$, so d^* is R_i-deactivating (i.e., R^*-deactivating since $R^* = R_i$ before the loop is entered) for A_i. Also, when vector d^* is first queried on line 5, i decreased by 1 due to the update on line 4, so d^* is R^*-deactivating for A_{i+1}. The required property thus holds when the condition on line 5 is first queried with $i = \max I - 1$.

We proceed with the inductive step. If the condition on line 5 is not satisfied, then necessarily $d^*_t = 0$ for all $t \in R_i$. So, if d^* is R^*-deactivating for A_{i+1}, then it is also R^*-deactivating for $A_i = A_{i+1} \cup R_i$, as seen from Definition 1.

If the condition on line 5 is satisfied, d^* is R^*-deactivating for A_{i+1} before the update on lines 6-7. Since $A_{i+1} = A_i - R_i$ and d^i is R_i-deactivating for A_i, Theorem 6 can be applied to d^i and d^* to obtain an $(R^* \cup R_i)$-deactivating direction for A_i. After updating R^* on line 7, it becomes R^*-deactivating for A_i.

When eventually $i = 0$, d^* is R^*-deactivating for $A_0 = A$ by line 2 in Algorithm 1. $\square$

Remark 3 This is similar to what the VAC [6] or Augmenting DAG algorithm [2, 16] do for arc consistency. To attempt to disprove satisfiability of CSP $A^*(f)$, these algorithms enforce AC of $A^*(f)$, during which the causes for forbidding tuples are stored. If the empty AC closure of $A^*(f)$ is detected (which corresponds to $T_S \cap A_{n+1} = \emptyset$ for some $S \in C$), these algorithms do not iterate through all previously forbidden tuples but only trace back the causes for forbidding the elements of the wiped-out domain (here, the elements of T_S).

4.3 Line search

In Section 4.2 we showed how to construct an R-deactivating direction d for a CSP A, which certifies unsatisfiability of A whenever $(A - R) \cap T_S = \emptyset$ for some $S \in C$. Given a WCSP $f \in \mathbb{R}^T$ with $A^*(f) = A$, to obtain $f' \in f + M^*$ with $B(f') < B(f)$ (as in Theorem 3), we need to find a step size $\alpha > 0$ so that $f' = f + \alpha d$, as discussed in Section 4.1.2. That means, we need to find $\alpha > 0$ such that $B(f + \alpha d) < B(f)$. This task is known in numerical optimization as *line search*.

Finding the best step size (i.e., exact line search) would require finding a global minimum of the univariate convex piecewise-affine function $\alpha \mapsto B(f + \alpha d)$. As this would be too expensive for large WCSP instances, we find only a suboptimal step size (approximate line search) in the following theorem.[16]

Theorem 7 *Let $f \in \mathbb{R}^T$. Let d be an R-deactivating direction for $A^*(f)$. Denote*[17]

$$\beta = \min \left\{ \frac{\max_{t \in T_{S'}} f_t - f_{t'}}{d_{t'}} \,\middle|\, S' \in C,\ t' \in T_{S'},\ d_{t'} > 0 \right\},$$

$$\gamma = \min \left\{ \frac{f_t - f_{t'}}{d_{t'} - d_t} \,\middle|\, S \in C,\ (A^*(f) - R) \cap T_S = \emptyset, t \in T_S \cap R,\ t' \in T_S - R,\ d_{t'} > d_t \right\}.$$

Then $\beta, \gamma > 0$ and for every $S \in C$ and $\alpha \in \mathbb{R}$, WCSP $f' = f + \alpha d$ satisfies:

(a) If $(A^*(f) - R) \cap T_S \neq \emptyset$ and $0 \le \alpha \le \beta$, then $\max_{t \in T_S} f'_t = \max_{t \in T_S} f_t$.
(b) If $(A^*(f) - R) \cap T_S \neq \emptyset$ and $0 < \alpha < \beta$, then $A^*(f') \cap T_S = (A^*(f) - R) \cap T_S$.
(c) If $(A^*(f) - R) \cap T_S = \emptyset$ and $0 < \alpha \le \min\{\beta, \gamma\}$, then $\max_{t \in T_S} f'_t < \max_{t \in T_S} f_t$.

Proof We have $\beta > 0$ because $d_{t'} > 0$ implies t' is an inactive tuple, so $\max_{t \in T_S} f_t > f_{t'}$. We have $\gamma > 0$ because in $f_t - f_{t'}$ tuple t is always active and t' is inactive, hence $f_t > f_{t'}$.

[16] In detail, the step size $\min\{\beta, \gamma\}$ computed in Theorem 7 corresponds to the first break (i.e., non-differentiable) point of the univariate function with a lower objective. This is analogous to the *first-hit strategy* in [45, §3.1.4].

[17] β is always defined: by Definition 1 we have $\langle d, \phi(x) \rangle \ge 0$ for all x, hence $\exists t\colon d_t < 0 \Rightarrow \exists t'\colon d_{t'} > 0$. γ is defined and needed only in (c), where we assume that $(A^*(f) - R) \cap T_S = \emptyset$ for some $S \in C$. If the set in the definition of γ is empty, then we define $\gamma = +\infty$ and thus $\min\{\beta, \gamma\} = \beta$.

To prove (a), let $t^* \in (A^*(f) - R) \cap T_S$. Hence, by Definition 1, $d_{t^*} = 0$ and the value $\max_{t \in T_S} f'_t$ does not decrease for any α since $f'_{t^*} = f_{t^*} + \alpha d_{t^*} = f_{t^*}$. To show the maximum does not increase, consider a tuple $t' \in T_S$ such that $d_{t'} > 0$ (due to $\alpha \geq 0$, tuples with $d_{t'} \leq 0$ cannot increase the maximum). It follows that $\alpha \leq \beta \leq \frac{\max_{t \in T_S} f_t - f_{t'}}{d_{t'}}$, so $f'_{t'} = f_{t'} + d_{t'}\alpha \leq \max_{t \in T_S} f_t$.

To prove (b), let $(A^*(f) - R) \cap T_S \neq \emptyset$. As in (a), we have $\max_{t \in T_S} f_t = \max_{t \in T_S} f'_t$. If $t \in (A^*(f) - R) \cap T_S$, then $d_t = 0$ and such tuples remain active by $f'_t = f_t$. Tuples $t \in R \cap T_S$ become inactive since $f'_t = f_t + d_t\alpha < f_t = \max_{t' \in T_S} f_{t'}$ by $d_t < 0$ and $\alpha > 0$. Tuples $t \notin A^*(f)$ either satisfy $d_t \leq 0$ and cannot become active or satisfy $d_t > 0$ and by $\alpha < \beta \leq \frac{\max_{t' \in T_S} f_{t'} - f_t}{d_t}$, $f'_t = f_t + d_t\alpha < \max_{t' \in T_S} f_{t'}$, so $t \notin A^*(f')$.

To prove (c), let $(A^*(f) - R) \cap T_S = \emptyset$. For all $t \in T_S \cap R$, we have $f'_t = f_t + \alpha d_t < f_t$ by $d_t < 0$ and $\alpha > 0$, i.e., $\max_{t \in T_S \cap R} f'_t < \max_{t \in T_S \cap R} f_t$. We proceed to show that $f'_t \leq \max_{t' \in T_S \cap R} f'_{t'}$ for every $t' \in T_S - R$. Let $t^* \in T_S \cap R$ satisfy $f'_{t^*} = \max_{t \in T_S \cap R} f'_t$. If $d_{t'} > d_{t^*}$, $\alpha \leq \gamma \leq \frac{f_{t^*} - f_{t'}}{d_{t'} - d_{t^*}}$ implies $f'_{t^*} = f_{t^*} + \alpha d_{t^*} \geq f_{t'} + \alpha d_{t'} = f'_{t'}$. If $d_{t'} \leq d_{t^*}$, then also $\alpha d_{t'} \leq \alpha d_{t^*}$ and $f'_{t'} = f_{t'} + \alpha d_{t'} \leq f_{t^*} + \alpha d_{t^*} = f'_{t^*}$ holds for any $\alpha \geq 0$ since $f_{t'} < f_{t^*}$. As a result, $\max_{t' \in T_S - R} f'_{t'} \leq \max_{t \in T_S \cap R} f'_t < \max_{t \in T_S \cap R} f_t = \max_{t \in T_S} f_t$. □

If d is an R-deactivating direction for CSP $A^*(f)$ and for all $S \in C$ we have $(A^*(f) - R) \cap T_S \neq \emptyset$ then, by Theorem 7(a,b), there is $\alpha > 0$ such that $f' = f + \alpha d$ satisfies $B(f') = B(f)$ and $A^*(f') = A^*(f) - R$. This justifies why such direction d is called R-deactivating: a suitable update of f along this direction makes tuples R inactive for f.

Remark 4 This might suggest that to improve the current bound $B(f)$, we need not use Algorithm 2 to construct an R^*-deactivating direction d^* with $(A^*(f) - R^*) \cap T_S = \emptyset$ for some $S \in C$, but instead, perform steps using the intermediate R_i-deactivating directions d^i to create a sequence $f^{i+1} = f^i + \alpha_i d^i$ satisfying $B(f^0) = B(f^1) = \cdots = B(f^n) > B(f^{n+1})$. Unfortunately, it is hard to make this work reliably as there are many choices for the intermediate step sizes $0 < \alpha_i < \beta_i$. We empirically found Algorithm 3 to be preferable.

If d is an R-deactivating direction for $A^*(f)$ and for some $S \in C$ we have $(A^*(f) - R) \cap T_S = \emptyset$, then, by Theorem 7(a,c), there is $\alpha > 0$ such that $f' = f + \alpha d$ satisfies $B(f') < B(f)$. The following corollary of Theorem 7 finally justifies why the certificate d of unsatisfiability of CSP $A^*(f)$ is an improving direction for (19):

Corollary 1 *CSP $A \subseteq T$ is unsatisfiable if and only if there is $d \in M^*$ such that for every $f \in \mathbb{R}^T$ with $A = A^*(f)$ there exists $\alpha > 0$ such that $B(f + \alpha d) < B(f)$*

Proof First, if for some $S \in C$ we have that $A \cap T_S = \emptyset$, A is unsatisfiable and no $f \in \mathbb{R}^T$ satisfies $A = A^*(f)$, so the second condition is trivially satisfied by choosing any $d \in M^*$.

Otherwise, let d be any A-deactivating direction (which exists by Theorem 4). It follows from Theorem 7 that for any $f \in \mathbb{R}^T$ with $A^*(f) = A$, we can compute a suitable step size $\alpha > 0$ such that $B(f + \alpha d) < B(f)$. The remaining part follows from Theorem 3. □

4.4 Final algorithm

Having certificates of unsatisfiability from Section 4.2 and step sizes from Section 4.3, we can now formulate in detail the iterative method outlined in Section 4.1.2, see Algorithm 3. First, constraint propagation is applied to CSP $A^*(f)$ by Algorithm 1 until either $A^*(f)$ is proved unsatisfiable or no more propagation is possible. In the latter case, the algorithm halts and returns $B(f)$ as the best achieved upper bound on the optimal value of WCSP g. Otherwise, if $A^*(f)$ is proved unsatisfiable due to $A_{n+1} \cap T_S = \emptyset$ for some $S \in C$, define I as in (32) so that $(A^*(f) - \bigcup_{i \in I} R_i) \cap T_S = \emptyset$, and compute an R^*-deactivating direction d^* where $R^* \supseteq \bigcup_{i \in I} R_i$ using Proposition 2. Since $(A^*(f) - R^*) \cap T_S = \emptyset$, we can update WCSP f using Theorem 7. Consequently, the bound $B(f)$ strictly improves after each update on line 7.

Algorithm 3 The final algorithm to iteratively improve feasible solutions to (19).

```
   input: WCSP $g \in \mathbb{R}^T$
1: Initialize $f := g$.
2: repeat
3:     $(S, (R_i)_{i=0}^n, (d^i)_{i=0}^n) :=$ propagate($A^*(f)$)
4:     if $S \neq \emptyset$ then
5:         Define $I$ as in (32).
6:         $(R^*, d^*) :=$ compose($(R_i)_{i=0}^n, (d^i)_{i=0}^n, I$)
7:         Update $f := f + \min\{\beta, \gamma\} d^*$ following Theorem 7.
8: until $S = \emptyset$
9: return $B(f)$
```

Remark 5 In the maximization version of WCSP, hard constraints can be modelled by allowing minus-infinite weights, i.e., we then have $g \in (\mathbb{R} \cup \{-\infty\})^T$. We argue that Algorithm 3 can be easily extended to such a setting. Without loss of generality, one can assume that

$$\forall S \in C \; \exists t \in T_S \colon \; g_t \in \mathbb{R}, \tag{33}$$

i.e., there is at least one finite weight in each scope for the input WCSP g (as otherwise the WCSP is infeasible).

With this assumption, the definition of the active-tuple CSP $A^*(\cdot)$ remains unchanged and the tuples with minus-infinite weights are never active. Next, see that propagation in the active-tuple CSP and construction of the improving direction depend only on $A^*(f)$, so these subroutines need not be modified and, consequently, the improving direction d^* still contains only finite weights, i.e., $d^* \in \mathbb{R}^T$.

The only difference can arise when computing the step size $\alpha = \min\{\beta, \gamma\}$ by Theorem 7. If $f \in \mathbb{R}^T$, then α is always finite. In contrast, if $f \in (\mathbb{R} \cup \{-\infty\})^T$, then it may happen that $\beta = \gamma = \infty$, so $\alpha = \min\{\beta, \gamma\} = \infty$ where we assume the usual arithmetic with infinities, so, e.g., $a - (-\infty) = \infty$ for $a \in \mathbb{R}$. As discussed earlier, the weights of the active tuples are always finite, which avoids indeterminate expressions when computing β and γ. Note, arithmetic with infinities is different from the addition with ceiling operator [6, 46] (unless the ceiling is infinite).

Next, we comment on the finite and infinite case:

- If the computed step size is finite, then $\alpha d^* \in \mathbb{R}^T$ and the update on line 7 can be performed following the aforementioned arithmetic with infinities. In this case, condition (33) holds for the updated f since the set of tuples with minus-infinite weight (i.e., the set $\{t \in T \mid f_t = -\infty\}$) is kept unchanged by the update and we can continue with the next iteration.
- On the other hand, if the step size is infinite, then the bound $B(f + \alpha d^*)$ can be made arbitrarily low by setting α large enough. Stated formally, this means $\forall b \in \mathbb{R} \, \exists \alpha > 0 \colon B(f + \alpha d^*) \leq b$, which proves infeasibility of the WCSP instance, so the algorithm should return $-\infty$ and terminate.

All in all, if hard constraints are allowed, the only required change in Algorithm 3 is that, if $\beta = \gamma = \infty$ on line 7, then the algorithm should terminate and return $-\infty$ (which is an upper bound on the infeasible initial WCSP).

In Algorithm 3 we additionally used a heuristic analogous to *capacity scaling* in network flow algorithms [47, §7.3]. On line 3 of Algorithm 3, we replace the active tuples $A^*(f)$ with 'almost' active tuples

$$A^*_\theta(f) = \left\{ t = (S,k) \in T \;\middle|\; f_t \geq \max_{t' \in T_S} f_{t'} - \theta \right\} \tag{34}$$

for some threshold $\theta > 0$.[18] This forces the algorithm to disprove satisfiability using tuples that are far from being active, thus hopefully leading to larger step sizes and faster decrease of the bound. Initially, θ is set to a high value and whenever we are unable to disprove satisfiability of $A^*_\theta(f)$, the current θ is decreased as $\theta := \theta/10$. The process continues until θ becomes very small.

Although our theoretical results are more general, our implementation is limited only to binary WCSPs, i.e., instances where the maximum arity of the weighted constraints is at most 2. We implemented two versions of Algorithm 3 (including capacity scaling[19]), differing in the local consistency used to attempt to disprove satisfiability of CSP $A^*(f)$:

- *Virtual singleton arc consistency via super-reparametrizations (VSAC-SR)* uses singleton arc consistency. Precisely, we alternate between AC and SAC propagators: whenever a single tuple $(i,k) \in V \times D$ is removed by SAC, we step back to enforcing AC until no more AC propagations are possible, and repeat.
- *Virtual cycle consistency via super-reparametrizations (VCC-SR)* is the same as VSAC-SR except that SAC is replaced by CC. Though our implementation is different than [25] (we compose deactivating directions rather than alternate between the cycle-repair procedure and the Augmenting DAG algorithm), it has the same fixed points.

The procedures for generating deactivating directions for AC, SAC and CC were implemented as described in Examples 3, 5, and 4. We used AC3 algorithm to enforce AC. In SAC and CC it is useful to step back to AC whenever possible because deactivating directions of AC correspond to reparametrizations rather than super-reparametrizations, which is desirable as explained in Section 4.1.1.

Remark 6 In analogy to [6, 22], let us call a WCSP instance f *virtual Φ-consistent* (e.g., virtual AC or virtual RPC) if $A^*(f)$ has a non-empty Φ-consistency closure. Then, a *virtual Φ-consistency algorithm* naturally refers to an algorithm to transform a given WCSP instance to a virtual Φ-consistent WCSP instance. In the VAC algorithm, this transformation is equivalence-preserving, i.e., a reparametrization. But in our case, it is a super-reparametrization, which is why we call our algorithms VSAC-SR and VCC-SR.

[18] This is similar to the notion of $\mathrm{Bool}_\theta(f)$ in [6, §11.1], tolerance δ in [38, §4.2], and $\mathrm{mi}_\epsilon[f]$ in [5, §6.2.4].

[19] In detail, we initialized $\theta = \max_{k_i,k_j} g_{\{i,j\}}(k_i,k_j) - \min_{k_i,k_j} g_{\{i,j\}}(k_i,k_j) + \max_k g_{i'}(k) - \min_k g_{i'}(k)$ where $\{i,j\} \in C$ and $i' \in V$ is the edge and variable with the lowest index (based on indexing in the input instance), respectively. The terminating condition was $\theta \leq 10^{-6}$. Let us note that if capacity scaling is used with $\theta > 0$ and the construction of the improving direction is deterministic (which is the case in our implementation), then the method is guaranteed to terminate after a finite number of iterations. This follows from our more general results that we state in [48, §2.2.1]. In order to improve the efficiency of our method, we also decreased θ whenever the bound did not improve by more than 10^{-15} in 20 consecutive iterations.

Since we restricted ourselves to binary WCSPs, let $E = \{S \in C \mid |S| = 2\}$ so that (V, E) is an undirected graph. The cycles in VCC-SR were chosen as follows: if $2|E|/|V| \leq 5$ (i.e., the average degree of the nodes in (V, E) is at most 5), then all cycles of length 3 and 4 present in the graph (V, E) are used. If $2|E|/|V| \leq 10$, then all cycles of length 3 present in the graph are used. If $2|E|/|V| > 10$ or the above method did not result in any cycles, we use all fundamental cycles w.r.t. a spanning tree of the graph (V, E).[20] No additional edges are added to the graph. Note, [25] experimented with grid graphs (where cycles of length 4 and 6 of the grid were used) and complete graphs (where cycles of length 3 were used).

Since both VSAC-SR and VCC-SR start by enforcing VAC (i.e., making $A^*(f)$ arc consistent by reparametrizations), before running these methods we used toulbar2 to reparametrize the input WCSP instance to a VAC state (because a specialized algorithm is faster than the more general Algorithm 3). We employed specialized data structures for storing the sequences $(R_i)_{i=0}^n$ and $(d^i)_{i=0}^n$ from Algorithm 1, which utilize the property that the sets $(R_i)_{i=0}^n$ are disjoint and make easier sequential querying of (sparse) vectors $(d^i)_{i=0}^n$ in Algorithm 2. Note that the sequence $(A_i)_{i=0}^{n+1}$ need not be stored and is only needed for theoretical analysis. Moreover, sparse representations were used when composing deactivating directions in Algorithm 2. To avoid working with 'structured' tuples (1), we employed a bijection between T and $\{1, \ldots, |T|\}$ to work with numerical indices instead.

Besides the above improvements, we did not fine-tune our implementation for efficiency. Thus, the set $A^*(f)$ was always calculated by iterating through all tuples (which could be made faster if sparsity of the improving direction was taken into account). The hyper-parameters of our algorithm (e.g., the decrease schedule of θ or constants mentioned in Footnote 19) were not learned nor systematically optimized. SAC was checked on all active tuples without warm-starting or using any faster SAC algorithm than SAC1 [44, 50]. Perhaps most importantly, we did not implement inter-iteration warm-starting as in [17, 45], i.e., after updating the weights on line 6 of Algorithm 3, some deactivating directions in the sequence that were not used to compose the improving direction may be preserved for the next iteration instead of being computed from scratch. Except for computing deactivating directions, the code was the same for VSAC-SR and VCC-SR. We implemented everything in Java.

4.5 Experiments

We compared the bounds calculated by VSAC-SR and VCC-SR with the bounds provided by EDAC [14], VAC [6], pseudo-triangles (option `-t=8000` in toulbar2, adds up to 8 GB of ternary weight functions), PIC, EDPIC, maxRPC, and EDmaxRPC [22], which are implemented in toulbar2 [51]. Our motivation for choosing these local consistencies is as follows: EDAC is the typically chosen local consistency that is maintained during branch-and-bound search. VAC is highly related to our approach and can be used in pre-processing (as it is faster than OSAC which is usually too memory- and time-consuming for practical purposes). Finally, we consider a class of recently proposed triangle-based consistencies [22] that enforce stronger forms of local consistency.

We did the comparison on the Cost Function Library benchmark [52]. Due to limited computation resources, we used only the smallest 16500 instances (out of 18132). Of these, we omitted instances containing weight functions of arity 3 or higher. Moreover, to avoid easy instances, we omitted instances that were solved by VAC without search (i.e., toulbar2 with

[20] Let (V, E') be a spanning tree of (V, E). A *fundamental cycle* w.r.t. the spanning tree is the unique cycle in the graph $(V, E' \cup \{e\})$ where $e \in E - E'$. By choosing different edges $e \in E - E'$, we obtain the set of all $|E - E'|$ fundamental cycles w.r.t. the spanning tree [49, Chapter 9].

options `-A -bt=0` found an optimal solution). We also omitted the *validation* instances that are used for testing and debugging. Overall, 5371 instances were left for our comparison.

For each instance and each method, we only calculated the upper bound and did not do any search. For each instance and method, we computed the normalized bound $\frac{B_w - B_m}{B_w - B_b}$ where B_m is the bound computed by the method for the instance and B_w and B_b is the worst and best bound for the instance among all the methods, respectively. Thus, the best bound[21] transforms to 1 and the worst bound to 0, i.e., greater is better.

For 26 instances, at least one method was not able to finish in the prespecified 1-hour CPU-time limit. These timed-out methods were omitted from the calculation of the normalized bounds for these instances. From the point of view of the method, the instance was not incorporated into the average of the normalized bounds of this particular method. We note that implementations of VSAC-SR and VCC-SR provide a bound when terminated at any time, whereas the implementations of the other methods provide a bound only when they are left to finish. Time-out happened 5, 2, 3, 6, and 24 times for pseudo-triangles, PIC, EDPIC, maxRPC, and EDmaxRPC, respectively. This did not affect the results much as there were 5371 instances in total.

The results in Table 1 show that no method is best for all instance groups, instead, each method is suitable for a different group. However, VSAC-SR performed best for most groups and otherwise was often competitive to the other strong consistency methods. VSAC-SR seems particularly good at spinglass_maxcut [53], planning [54] and qplib [55] instances. Taking the overall unweighted average of group averages (giving the same importance to each group), VSAC-SR achieved the greatest average value. We also evaluated the ratio to worst bound, B_m/B_w, for instances with $B_w \neq 0$; the results were qualitatively the same: VSAC-SR again achieved the best overall average of 3.93 (or 4.15 if only groups with ≥ 5 instances are considered) compared to second-best pseudo-triangles with 2.71 (or 2.84).

The runtimes (on a laptop with i7-4710MQ processor at 2.5 GHz and 16GB RAM) are reported in Table 2. Again, the results are group-dependent and one can observe that the methods explore different trade-offs between bound quality and runtime. However, the strong consistencies are comparable in terms of runtime on average, except for pseudo-triangles, which is a faster method that however needs significantly more memory.

The code that was used to obtain these results is available at https://cmp.felk.cvut.cz/~dlaskto2/code/VSAC-SR.zip.

5 Additional properties of super-reparametrizations

In this section, we present a more detailed study of properties of WCSPs that are preserved by (possibly optimal) super-reparametrizations. To that end, we first revisit in Section 5.1 the notion of a minimal CSP for a set of assignments. The key result of Section 5 is presented in Section 5.2, where we study the relation of the set of optimal assignments of some WCSP to the set of optimal assignments of its super-reparametrization optimal for (19), showing that they need not coincide in general. In Section 5.3, we give some properties of general (i.e., not necessarily optimal for (19)) super-reparametrizations.

[21] To avoid numerical precision issues, bounds B_m within $B_b \pm 10^{-4} B_b$ or $B_b \pm 0.01$ are also normalized to 1. If $B_w = B_b$, then the normalized bounds for all methods are equal to 1 on this instance.

Table 1 Results on instances from Cost Function Library: Average normalized bounds (for each instance group, the best average normalized bound is in bold)

Instance Group	Instances	EDAC	VAC	VSAC-SR	VCC-SR	Pseudo-tr.	PIC	EDPIC	maxRPC	EDmaxRPC
/biqmaclib/	157	0.02	0.11	0.90	0.22	**0.92**	0.83	0.81	0.79	0.81
/crafted/academics/	8	0.88	0.88	0.97	0.95	0.88	0.88	0.88	0.88	**1.00**
/crafted/auction/paths/	420	0.00	0.09	0.91	0.35	**0.99**	0.45	0.68	0.64	0.57
/crafted/auction/regions/	411	0.00	0.05	**0.99**	0.10	0.98	0.08	0.18	0.23	0.13
/crafted/auction/scheduling/	419	0.00	0.02	**1.00**	0.09	0.80	0.41	0.38	0.41	0.24
/crafted/coloring/	33	0.94	0.94	0.99	0.97	0.98	**1.00**	**1.00**	**1.00**	0.99
/crafted/feedback/	6	0.00	0.00	0.54	0.58	0.71	0.49	0.53	0.51	**0.72**
/crafted/kbtree/	1800	0.25	0.29	0.60	0.67	0.80	0.73	0.81	0.76	**0.89**
/crafted/maxclique/dimacs_maxclique/	49	0.06	0.24	**0.98**	0.39	0.87	0.39	0.50	0.51	0.55
/crafted/maxcut/spinglass_maxcut/unweighted/	5	0.00	0.00	**1.00**	0.42	0.15	0.15	0.15	0.15	0.15
/crafted/maxcut/spinglass_maxcut/weighted/	5	0.00	0.00	**1.00**	0.38	0.17	0.17	0.17	0.17	0.17
/crafted/modularity/	6	0.17	0.19	0.38	0.25	**0.99**	0.96	0.94	0.96	0.97
/crafted/planning/	65	0.00	0.54	**0.94**	0.72	0.32	0.07	0.09	0.07	0.17
/crafted/sumcoloring/	43	0.04	0.15	0.47	0.50	**0.81**	0.53	0.63	0.64	0.61
/crafted/warehouses/	49	0.35	0.99	**1.00**	0.99	0.35	0.42	0.42	0.42	0.42
/qaplib/	5	0.40	0.40	0.40	0.41	**0.99**	0.97	0.97	0.98	0.97
/qplib/	23	0.00	0.10	**0.96**	0.38	0.27	0.25	0.25	0.24	0.25
/random/maxcsp/completeloose/	50	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
/random/maxcsp/completetight/	50	0.00	0.12	0.57	0.72	0.88	0.94	**0.99**	0.69	0.76
/random/maxcsp/denseloose/	50	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
/random/maxcsp/densetight/	50	0.02	0.14	0.52	**1.00**	0.68	0.48	0.49	0.52	0.60
/random/maxcsp/sparseloose/	90	0.96	0.96	**1.00**	0.96	0.96	0.96	0.96	0.96	0.96

Table 1 continued

Instance Group	Instances	EDAC	VAC	VSAC-SR	VCC-SR	Pseudo-tr.	PIC	EDPIC	maxRPC	EDmaxRPC
/random/maxcsp/sparsetight/	50	0.01	0.12	0.54	**1.00**	0.64	0.40	0.40	0.43	0.51
/random/maxcut/random_maxcut/	400	0.00	0.00	0.77	0.13	0.95	0.98	0.98	0.97	**0.99**
/random/mincut/	500	0.09	**1.00**	**1.00**	**1.00**	0.10	0.10	0.10	0.10	0.10
/random/randomksat/	493	0.01	0.02	0.75	0.22	**0.95**	0.91	0.89	0.86	0.87
/random/wqueens/	6	0.00	0.52	**0.96**	0.94	0.48	0.12	0.29	0.13	0.72
/real/celar/	23	0.00	0.05	0.08	0.16	**0.97**	0.66	0.66	0.78	0.95
/real/maxclique/protein_maxclique/	1	0.00	0.00	**1.00**	0.03	0.93	0.04	0.04	0.08	0.04
/real/spot5/	1	0.00	0.08	**1.00**	0.49	**1.00**	0.74	0.66	0.41	0.74
/real/tagsnp/tagsnp_r0.5/	23	0.04	0.86	**0.95**	0.86	0.31	0.31	0.33	0.29	0.46
/real/tagsnp/tagsnp_r0.8/	80	0.13	0.66	**0.91**	0.68	0.29	0.39	0.38	0.33	0.47
Average over all groups	5371	0.20	0.36	**0.82**	0.58	0.72	0.56	0.58	0.56	0.62
Average over groups with $\geq$ 5 instances	5369	0.21	0.38	**0.80**	0.60	0.71	0.57	0.59	0.58	0.63

Table 2 Results on instances from Cost Function Library: Average CPU time in seconds (for each instance group, the shortest average CPU time is in bold)

Instance Group	Instances	EDAC	VAC	VSAC-SR	VCC-SR	Pseudo-tr.	PIC	EDPIC	maxRPC	EDmaxRPC
/biqmaclib/	157	**0.11**	0.12	180.07	34.60	83.25	1240.00	1241.29	1242.16	1271.86
/crafted/academics/	8	**0.11**	**0.11**	28.61	1.04	29.08	121.44	120.86	108.08	104.47
/crafted/auction/paths/	420	**0.04**	**0.04**	1.96	0.83	1.92	0.19	0.23	0.48	0.64
/crafted/auction/regions/	411	**0.20**	0.32	32.14	9.45	673.42	49.85	51.37	102.61	110.48
/crafted/auction/scheduling/	419	**0.10**	0.12	16.22	2.03	49.85	26.90	26.89	32.06	32.30
/crafted/coloring/	33	**0.09**	0.10	4.99	1.40	0.20	545.50	545.50	545.51	545.50
/crafted/feedback/	6	**0.70**	**0.70**	3588.39	3600.11	11.64	1860.89	1874.08	1875.93	1873.07
/crafted/kbtree/	1800	**0.02**	**0.02**	3.13	11.25	0.10	0.04	0.05	0.06	0.07
/crafted/maxclique/dimacs_maxclique/	49	**0.71**	1.32	279.08	126.90	955.60	1345.67	1342.14	1429.73	1428.12
/crafted/maxcut/spinglass_maxcut/unweighted/	5	0.02	0.02	0.82	0.44	0.02	**0.01**	**0.01**	**0.01**	**0.01**
/crafted/maxcut/spinglass_maxcut/weighted/	5	0.02	0.02	1.09	0.53	0.02	**0.01**	**0.01**	**0.01**	**0.01**
/crafted/modularity/	6	**0.19**	0.29	1023.48	127.39	66.25	706.30	783.02	741.91	1442.57
/crafted/planning/	65	**0.16**	0.29	638.85	60.62	7.41	0.93	0.96	2.33	4.73
/crafted/sumcoloring/	43	**1.29**	1.94	727.49	963.61	255.72	1508.37	1508.36	1509.34	1512.68
/crafted/warehouses/	49	4.10	9.48	735.80	735.83	**4.09**	29.48	29.54	28.80	29.82
/qaplib/	5	**0.08**	0.09	119.05	278.53	7.38	1448.63	1444.95	1450.09	1449.22
/qplib/	23	**0.13**	0.14	255.85	43.11	195.32	626.25	626.24	626.27	626.36
/random/maxcsp/completeloose/	50	**0.06**	**0.06**	1.31	0.16	0.48	0.09	0.10	0.19	0.18
/random/maxcsp/completetight/	50	**0.02**	0.03	6.35	12.68	0.47	0.21	0.25	0.31	0.33
/random/maxcsp/denseloose/	50	**0.02**	**0.02**	166.78	0.06	0.11	0.03	0.03	0.03	0.03
/random/maxcsp/densetight/	50	**0.02**	**0.02**	4.20	17.38	0.10	0.06	0.07	0.07	0.08
/random/maxcsp/sparseloose/	90	**0.03**	**0.03**	611.38	0.05	0.06	0.04	0.04	0.04	0.04
/random/maxcsp/sparsetight/	50	**0.02**	**0.02**	11.00	9.74	0.06	0.04	0.05	0.05	0.05

Table 2 continued

Instance Group	Instances	EDAC	VAC	VSAC-SR	VCC-SR	Pseudo-tr.	PIC	EDPIC	maxRPC	EDmaxRPC
/random/maxcut/random_maxcut/	400	**0.01**	**0.01**	0.73	0.15	0.04	0.03	0.03	0.05	0.07
/random/mincut/	500	1.09	2.43	14.40	86.22	1.12	0.88	**0.87**	**0.87**	**0.87**
/random/randomksat/	493	**0.02**	**0.02**	3.42	0.17	0.13	0.07	0.10	0.16	0.31
/random/wqueens/	6	**1.33**	1.49	992.85	502.42	644.87	1800.15	1800.20	1800.18	1800.60
/real/celar/	23	**0.27**	0.28	1798.51	2972.69	66.56	300.76	219.91	495.26	1066.87
/real/maxclique/protein_maxclique/	1	**0.26**	0.44	25.24	6.77	1196.62	114.62	114.99	215.30	220.81
/real/spot5/	1	**0.01**	**0.01**	0.62	0.08	0.11	0.03	0.03	0.04	0.04
/real/tagsnp/tagsnp_r0.5/	23	**4.83**	378.77	3338.53	2897.83	239.38	3155.96	3148.66	3172.58	3295.19
/real/tagsnp/tagsnp_r0.8/	80	**1.52**	22.82	1239.73	858.83	90.05	195.12	206.76	359.55	409.88
Average over all groups	5371	**0.55**	13.17	495.38	417.59	143.17	471.21	471.49	491.88	538.35
Average over groups with ≥ 5 instances	5369	**0.58**	14.04	527.54	445.20	112.82	498.80	499.08	517.49	566.88

5.1 Minimal CSP

Let us ask when for a given set $X \subseteq D^V$ of assignments (i.e., a $|V|$-ary relation over D) does there exist $A \subseteq T$ such that $X = \mathrm{SOL}(A)$, i.e., when is X representable as the solution set of a CSP with a given structure (D, V, C). For that, denote

$$A_{\min}(X) = \bigcap \mathcal{A}^{\uparrow}(X) \quad \text{where} \quad \mathcal{A}^{\uparrow}(X) = \{A \subseteq T \mid X \subseteq \mathrm{SOL}(A)\}\,. \tag{35}$$

Thus, $\mathcal{A}^{\uparrow}(X)$ is the set of all CSPs whose solution set includes X and $A_{\min}(X)$ is the intersection of these CSPs. We call $A_{\min}(X)$ the *minimal CSP* for X. For CSPs with only binary relations, this concept was studied in [56] and [57, §2.3.2].

Proposition 3 *The map SOL preserves intersections*[22]*, i.e., for any $A_1, A_2 \subseteq T$ we have $SOL(A_1 \cap A_2) = SOL(A_1) \cap SOL(A_2)$.*

Proof For any $x \in D^V$ we have

$$\begin{aligned} x \in \mathrm{SOL}(A_1) \cap \mathrm{SOL}(A_2) &\iff x \in \mathrm{SOL}(A_1),\ x \in \mathrm{SOL}(A_2) \\ &\iff \forall S \in C\colon (S, x[S]) \in A_1,\ (S, x[S]) \in A_2 \\ &\iff \forall S \in C\colon (S, x[S]) \in A_1 \cap A_2 \\ &\iff x \in \mathrm{SOL}(A_1 \cap A_2). \end{aligned}$$

□

Proposition 4 *For any $X \subseteq D^V$, the set $\mathcal{A}^{\uparrow}(X)$ is closed under intersections, i.e., for any $A_1, A_2 \subseteq T$ we have $A_1, A_2 \in \mathcal{A}^{\uparrow}(X) \implies A_1 \cap A_2 \in \mathcal{A}^{\uparrow}(X)$.*

Proof If $X \subseteq \mathrm{SOL}(A_1)$ and $X \subseteq \mathrm{SOL}(A_2)$, then $X \subseteq \mathrm{SOL}(A_1) \cap \mathrm{SOL}(A_2) = \mathrm{SOL}(A_1 \cap A_2)$, where the equality holds by Proposition 3. □

Proposition 4 implies $A_{\min}(X) \in \mathcal{A}^{\uparrow}(X)$, i.e., $X \subseteq \mathrm{SOL}(A_{\min}(X))$. This shows that $A_{\min}(X)$ is the smallest CSP whose solution set includes X. It follows that $X = \mathrm{SOL}(A_{\min}(X))$ if and only if $X = \mathrm{SOL}(A)$ for some $A \subseteq T$.

The minimal CSP for X can be equivalently defined in terms of tuples:

Proposition 5 [56, 57] *We have $A_{\min}(X) = \{(S, k) \in T \mid \exists x \in X\colon x[S] = k\}$.*

Proof Denote $A' = \{(S, k) \in T \mid \exists x \in X\colon x[S] = k\}$. By definition of A' we have $\mathrm{SOL}(A') \supseteq X$, so $A' \in \mathcal{A}^{\uparrow}(X)$ and $A_{\min}(X) = \bigcap \mathcal{A}^{\uparrow}(X) \subseteq A'$.

It remains to show that $A_{\min}(X) \supseteq A'$. For contradiction, suppose there is a tuple $(S^*, k^*) \in A' - A_{\min}(X)$. By definition of A', there exists $x \in X$ such that $x[S^*] = k^*$. However, since $(S^*, x[S^*]) = (S^*, k^*) \notin A_{\min}(X)$, we have $x \notin \mathrm{SOL}(A_{\min}(X))$. By $x \in X$, this contradicts $X \subseteq \mathrm{SOL}(A_{\min}(X))$. □

Recall that a CSP A is *positively consistent* [58, 59] (called 'minimal' in [56, 57]) if and only if for each $(S, k) \in A$ there exists $x \in \mathrm{SOL}(A)$ such that $x[S] = k$, i.e., each allowed tuple is used by at least one solution, i.e., no tuple can be forbidden without losing some solutions. Proposition 5 shows that $A_{\min}(X)$ is positively consistent for every $X \subseteq D^V$.

[22] This implies that the map SOL is isotone, i.e., $A_1 \subseteq A_2 \subseteq T$ implies $\mathrm{SOL}(A_1) \subseteq \mathrm{SOL}(A_2)$. Although isotony is obvious (clearly, enlarging the set of allowed tuples of a CSP preserves or enlarges its solution set), note that isotony does not imply preserved intersections. A weaker result than our Proposition 3 is [56, Theorem 3.2]: in our notation, it says that $\mathrm{SOL}(A_1) = \mathrm{SOL}(A_2)$ implies $\mathrm{SOL}(A_1 \cap A_2) = \mathrm{SOL}(A_1)$.

Theorem 8 *For any $X \subseteq D^V$ and $A \subseteq T$, we have*

$$A_{\min}(X) \subseteq A \quad \iff \quad X \subseteq SOL(A). \tag{36}$$

Proof If $A_{\min}(X) \subseteq A$, then by isotony of SOL we have $\mathrm{SOL}(A_{\min}(X)) \subseteq \mathrm{SOL}(A)$. Since $X \subseteq \mathrm{SOL}(A_{\min}(X))$, we have $X \subseteq \mathrm{SOL}(A)$.

If $X \subseteq \mathrm{SOL}(A)$, i.e., $A \in \mathcal{A}^{\uparrow}(X)$, then $A_{\min}(X) = \bigcap \mathcal{A}^{\uparrow}(X) \subseteq A$. □

Comparing (36) with (35) shows that the set $\mathcal{A}^{\uparrow}(X)$ is just an interval (w.r.t. the partial ordering by inclusion):

$$\mathcal{A}^{\uparrow}(X) = \{A \mid A_{\min}(X) \subseteq A \subseteq T\} = [A_{\min}(X), T]. \tag{37}$$

Theorem 8 further reveals that the maps $A_{\min}$ and SOL form a *Galois connection* [60] between sets $2^{(D^V)}$ and 2^T, partially ordered by inclusion (to our knowledge, we are the first to notice this). Associated with the Galois connection are the closure operator $\mathrm{SOL} \circ A_{\min}$ and the dual closure operator[23] $A_{\min} \circ \mathrm{SOL}$. We have already seen their meaning:

- For any CSP $A \subseteq T$, the CSP $A_{\min}(\mathrm{SOL}(A))$ is the *positive consistency closure*[24] of A, i.e., the smallest CSP with the same solution set as A. A CSP A is positively consistent if and only if $A_{\min}(\mathrm{SOL}(A)) = A$.
- For any set of assignments $X \subseteq D^V$, $\mathrm{SOL}(A_{\min}(X))$ is the smallest (possibly non-strict) superset of X which is the solution set of some CSP $A \subseteq T$. We have $X = \mathrm{SOL}(A_{\min}(X))$ if and only if $X = \mathrm{SOL}(A)$ for some $A \subseteq T$.

Remark 7 Following [60, §7.27], it is easy to see in this case that the maps $A_{\min}$ and SOL are mutually inverse bijections (even order-isomorphisms) if we restrict ourselves only to positively consistent CSPs, i.e., $\{A \subseteq T \mid \mathrm{SOL}(A_{\min}(A)) = A\}$ and sets of assignments representable as solution sets of some CSP $A \subseteq T$, i.e., $\{\mathrm{SOL}(A) \mid A \subseteq T\}$.

5.2 Optimal assignments from optimal super-reparametrizations

Theorem 2 says that the optimal value of (19) coincides with the optimal value $\max_x \langle g, \phi(x)\rangle$ of WCSP g. We now focus on the optimal assignments (rather than optimal value) of WCSP g. For brevity, we will denote the set of all optimal assignments of WCSP g as

$$\mathrm{OPT}(g) = \operatorname*{arg\,max}_{x \in D^V} \langle g, \phi(x)\rangle \subseteq D^V. \tag{38}$$

Theorem 9 *If f is optimal for* (19)*, then*[25] *$OPT(g) \subseteq OPT(f) = SOL(A^*(f))$.*

[23] Recall that an operator is a closure (dual closure) if it is isotone, idempotent and increasing (decreasing), see e.g. [61, §1.4].

[24] The term *consistency closure*, as used in constraint programming [43], is a dual closure in our notation because taking a consistency closure of a CSP *deletes* some of its allowed tuples (it would be a closure if the CSP was defined by a set of *forbidden* tuples).

[25] Statement (c) in Theorem 2 is equivalent to $\mathrm{SOL}(A^*(f)) \cap W(f-g) \neq \emptyset$ where $W(d) = \{x \in D^V \mid \langle d, \phi(x)\rangle = 0\}$, i.e., satisfiability of CSP $A^*(f)$ with an additional global constraint $x \in W(f-g)$. As a corollary of Theorem 9, we have that $\mathrm{SOL}(A^*(f)) \cap W(f-g) \neq \emptyset \implies \mathrm{SOL}(A^*(f)) \cap W(f-g) = \mathrm{OPT}(g)$ for any super-reparametrization f of g.

Proof To show $\mathrm{OPT}(g) \subseteq \mathrm{OPT}(f)$, let $x^* \in \mathrm{OPT}(g)$. By Theorem 2, $\langle g, \phi(x^*)\rangle = B(f)$. Analogously to the proof of Theorem 2: since $B(f) \geq \langle f, \phi(x^*)\rangle \geq \langle g, \phi(x^*)\rangle$, we have that $B(f) = \langle f, \phi(x^*)\rangle = \langle g, \phi(x^*)\rangle$, thus x^* is optimal for WCSP f.

The equality $\mathrm{OPT}(f) = \mathrm{SOL}(A^*(f))$ follows from $B(f) = \max_{x\in D^V}\langle f, \phi(x)\rangle = \max_{x\in D^V}\langle g, \phi(x)\rangle$ and Theorem 1. □

Our main goal in Section 5 is to characterize when the inclusion in Theorem 9 holds with equality, which is given by Theorem 11 below.

Proposition 6 *For every $g \in \mathbb{R}^T$ and $A \subseteq T$ such that $OPT(g) \subseteq SOL(A)$, there exists $f \in \mathbb{R}^T$ optimal for* (19) *such that $A = A^*(f)$.*

Proof Define the vector f as

$$f_t = \begin{cases} F_1/|C| \text{ if } t \in A \\ F_2/|C| \text{ if } t \notin A \end{cases} \quad \forall t \in T \tag{39}$$

where

$$F_1 = \max_{x\in D^V}\langle g, \phi(x)\rangle, \quad F_2 = \max\{\langle g, \phi(x)\rangle \mid x \in D^V, \langle g, \phi(x)\rangle < F_1\} \tag{40}$$

are the best and the second-best objective value of WCSP g. Note, if $\mathrm{OPT}(g) = D^V$, then F_2 is undefined but it does not matter because it is never used in (39).

Since $\emptyset \neq \mathrm{OPT}(g) \subseteq \mathrm{SOL}(A)$, CSP A is satisfiable. Therefore for each $S \in C$ we have $A \cap T_S \neq \emptyset$, hence

$$\max_{t\in T_S} f_t = F_1/|C|. \tag{41}$$

Equality $A = A^*(f)$ now follows from (39).

To show that f is feasible for (19), we distinguish two cases:

- If $x \in \mathrm{OPT}(g)$, i.e., $\langle g, \phi(x)\rangle = F_1$, then $x \in \mathrm{SOL}(A) = \mathrm{SOL}(A^*(f))$. Therefore for all $S \in C$ we have $(S, x[S]) \in A^*(f)$, hence $f_S(x[S]) = F_1/|C|$ by (41). Substituting into (4) yields $\langle f, \phi(x)\rangle = F_1$. Hence $\langle f, \phi(x)\rangle = F_1 = \langle g, \phi(x)\rangle$.
- If $x \notin \mathrm{OPT}(g)$, we have $f_t \geq F_2/|C|$ for all $t \in T$, hence $\langle f, \phi(x)\rangle \geq F_2$ by (4). By (40) we also have $\langle g, \phi(x)\rangle \leq F_2$. Hence $\langle f, \phi(x)\rangle \geq F_2 \geq \langle g, \phi(x)\rangle$.

To show that f is optimal for (19), we use (41) to obtain $B(f) = \sum_{S\in C} F_1/|C| = F_1 = \max_x\langle g, \phi(x)\rangle$ and apply Theorem 2. □

Theorem 10 *For every $g \in \mathbb{R}^T$, we have*

$$\mathcal{A}^{\uparrow}(OPT(g)) = \left\{A^*(f) \mid f \textit{ is optimal for } (19)\right\}. \tag{42}$$

Proof The inclusion $\supseteq$ says that for every optimal f we have $\mathrm{OPT}(g) \subseteq \mathrm{SOL}(A^*(f))$, which was proved in Theorem 9. The inclusion $\subseteq$ was proved in Proposition 6. □

Now we combine the results of Sections 5.1 and 5.2 to obtain the main result of Section 5. First observe that, by (37), the set (42) is just the interval $[A_{\min}(\mathrm{OPT}(g)), T]$.

Theorem 11 *For every $g \in \mathbb{R}^T$, the following statements are equivalent:*

(a) $OPT(g) = SOL(A)$ for some $A \subseteq T$,
(b) $OPT(g) = OPT(f)$ for some f optimal for (19).

If both statements are true, then statement (a) holds, e.g., for $A = A_{\min}(OPT(g))$ and statement (b) holds, e.g., if $A^*(f) = A_{\min}(OPT(g))$.

Proof Let $g \in \mathbb{R}^T$. By Theorem 10, there exists f optimal for (19) satisfying $A^*(f) = A_{\min}(\mathrm{OPT}(g))$. By Theorem 9, this f satisfies $\mathrm{OPT}(f) = \mathrm{SOL}(A^*(f))$.

By the results of Section 5.1, statement (a) is equivalent to $\mathrm{OPT}(g) = \mathrm{SOL}(A_{\min}(\mathrm{OPT}(g)))$. Therefore, if (a) holds, then (b) holds for the above f. In the other direction, if (b) holds for the above f, then (a) holds. □

Theorem 11 shows that the inclusion in Theorem 9 holds with equality for some optimal f if and only if the set $\mathrm{OPT}(g)$ of optimal assignments of WCSP g is representable as a solution set of some CSP with the same structure. If no such CSP exists, then $\mathrm{OPT}(g) \subsetneq \mathrm{OPT}(f)$ for all optimal f. An example of WCSP g for which no such CSP exists is in Fig. 6.

It is natural to ask which WCSPs possess this property. Though we are currently unable to provide a full characterization of such WCSPs, we identify two such classes:

Theorem 12 [1, 2] *If the LP relaxation (13) of a WCSP $g \in \mathbb{R}^T$ is tight, then $OPT(g) = SOL(A)$ for some $A \subseteq T$.*

Proof If the LP relaxation (13) is tight, then there exists a vector $f \in \mathbb{R}^T$ such that $B(f) = \max_{x \in D^V} \langle g, \phi(x) \rangle$ and f is a reparametrization of g, i.e., $\langle f, \phi(x) \rangle = \langle g, \phi(x) \rangle$ for all $x \in D^V$, thus, f is also optimal for (19). It follows that the sets of optimal assignments for f and g coincide. By Theorem 9, $A^*(f)$ is the required CSP. □

Theorem 13 *If a WCSP $g \in \mathbb{R}^T$ has a unique optimal assignment (i.e., $|OPT(g)| = 1$), then $OPT(g) = SOL(A)$ for some $A \subseteq T$.*

Proof The case with $|D| = 1$ is trivial, so let $|D| \geq 2$ and $\mathrm{OPT}(g) = \{x\}$.

We claim that $A = \{(S, x[S]) \mid S \in C\}$ is the required CSP, i.e., $\mathrm{SOL}(A) = \{x\}$. For contradiction, suppose that $x' \in \mathrm{SOL}(A)$ and $x' \neq x$. By $x' \in \mathrm{SOL}(A)$, we necessarily have that $x'[S] = x[S]$ for all $S \in C$. By definition (4), this implies $\langle g, \phi(x') \rangle = \langle g, \phi(x) \rangle$. Thus, $\{x', x\} \subseteq \mathrm{OPT}(g)$, which is contradictory with $|\mathrm{OPT}(g)| = 1$. □

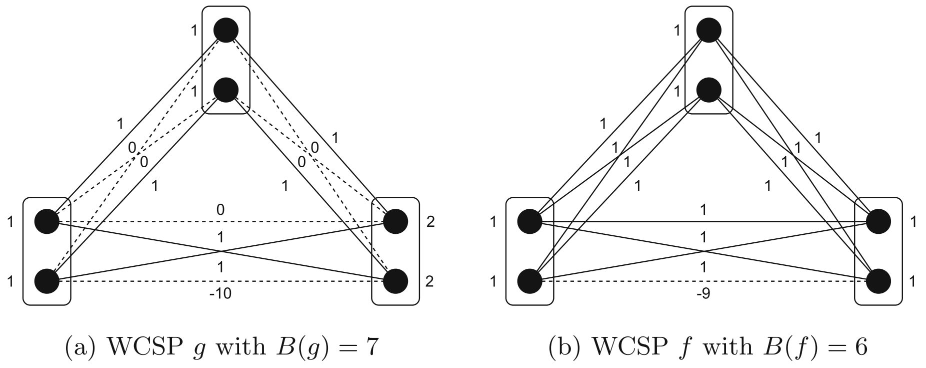


Fig. 6 WCSP f is an optimal super-reparametrization of WCSP g. It is easy to verify that $A^*(f) = A_{\min}(\mathrm{OPT}(g))$ but $\mathrm{OPT}(f) = \mathrm{SOL}(A^*(f)) \supsetneq \mathrm{OPT}(g)$

5.3 Properties of general super-reparametrizations

Finally, we present one property of general super-reparametrizations f of a fixed WCSP $g \in \mathbb{R}^T$, i.e., f is only feasible (but possibly not optimal) for (19).

Theorem 14 *For every $g \in \mathbb{R}^T$ we have*

$$\{A^*(f) \mid f \in \mathbb{R}^T \text{ is a super-reparametrization of } g\} = \{A^*(f) \mid f \in \mathbb{R}^T\}. \tag{43}$$

Proof The inclusion $\subseteq$ is trivial. To prove $\supseteq$, let $f' \in \mathbb{R}^T$ be arbitrary. Define $f \in \mathbb{R}^T$ as $f_t = B(g)/|C| + [\![t \in A^*(f')]\!]$, $t \in T$. Clearly, f is a super-reparametrization of g due to $\langle f, \phi(x)\rangle \geq B(g) \geq \langle g, \phi(x)\rangle$ for any $x \in D^V$. In addition, by definition of f, $\max_{t \in T_S} f_t = B(g)/|C| + 1$ for any $S \in C$, hence $A^*(f') = A^*(f)$. □

Theorem 14 shows that the left-hand set in (43) does not depend on g at all. Therefore, if we approximately optimize (19), i.e., we find a (possibly non-optimal) super-reparametrization f of g, then there is in general no relation between sets $\mathrm{OPT}(f)$ and $\mathrm{OPT}(g)$. However, as shown in Section 3.2, an arbitrary super-reparametrization still maintains the valuable property that it provides an upper bound $B(f)$ on the optimal value $\max_x \langle g, \phi(x)\rangle$ of WCSP g.

6 Hardness remarks

Unsurprisingly, a number of decision and optimization problems related to (19) is computationally hard since the optimization problem (19) is hard itself. We overview a number of such problems here.

Theorem 15 *The following problem is NP-complete: Given $f, g \in \mathbb{Q}^T$, decide whether f is not a super-reparametrization of g (i.e., whether f is not feasible for* (19)*).*

Proof Membership in NP can be shown easily by the notion of a non-deterministic algorithm [62, §10]. First, one can choose any $x \in D^V$ and then in polynomial time decide whether $\langle f, \phi(x)\rangle < \langle g, \phi(x)\rangle$.

To show NP-hardness, we perform a reduction from CSP satisfiability which is known to be NP-complete. Let $A \subseteq T$ be a CSP. We would like to decide whether $\mathrm{SOL}(A) \neq \emptyset$.

Let us define $g \in \{0, 1\}^T$ by

$$g_S(k) = [\![(S, k) \in A]\!] \quad \forall (S, k) \in T. \tag{44}$$

Thus, for any $x \in D^V$, $\langle g, \phi(x)\rangle$ equals to the number of constraints in CSP A that are satisfied by the assignment x. So, $\langle g, \phi(x)\rangle \in \{0, 1, \ldots, |C|\}$ and $\langle g, \phi(x)\rangle = |C|$ if and only if $x \in \mathrm{SOL}(A)$. Consequently, $\max_x \langle g, \phi(x)\rangle \leq |C| - 1$ if and only if $\mathrm{SOL}(A) = \emptyset$.

We define $f \in \mathbb{Q}^T$ by $f_t = (|C| - 1)/|C|$, $t \in T$. In analogy to Theorem 2, $\langle f, \phi(x)\rangle = |C| - 1$ for all assignments $x \in D^V$. Hence, $\mathrm{SOL}(A) = \emptyset$ if and only if f is a super-reparametrization of g. □

Corollary 2 *The following problem is NP-complete: Given $d \in \mathbb{Q}^T$, decide whether $d \notin M^*$.*

Proof Membership in NP is analogous to Theorem 15. The question of whether f is not a super-reparametrization of g from Theorem 15 reduces to whether $d = f - g \notin M^*$. □

Corollary 3 *The following problem is NP-complete: Given $f, g \in \{0, 1\}^T$ where f is a super-reparametrization of g, decide whether f is optimal for* (19).

Proof Membership in NP follows from Theorem 2: as in Theorem 15, one can choose $x \in D^V$ and then in polynomial time decide whether $\langle f, \phi(x)\rangle = \langle g, \phi(x)\rangle$ and $x \in \mathrm{SOL}(A^*(f))$.

The hardness part is completely analogous to the proof of Theorem 15 except that we define $f \in \{0,1\}^T$ by $f_t = 1, t \in T$, so $B(f) = |C|$. Clearly, f is element-wise greater or equal to g, so it is a super-reparametrization. Moreover, $|C| = \max_x \langle g, \phi(x)\rangle$ if and only if $\mathrm{SOL}(A) \neq \emptyset$, so f is optimal for (19) if and only if $\mathrm{SOL}(A) \neq \emptyset$. □

Recall that in formula (27), the number δ had the concrete value given by Theorem 4. However, sometimes the value of δ can be decreased while (27) still remains to be an R-deactivating direction for A. Finding a small such δ is desirable because then (27) results in smaller objective values $\langle d, \phi(x)\rangle$, as explained in Section 4.1.1. Unfortunately, finding the least value of δ is likely intractable:

Theorem 16 *The following problem is NP-complete: Given $\delta \in \mathbb{Q}$ and $R \subseteq A \subseteq T$ satisfying $SOL(A) = SOL(A - R)$, decide whether vector d given by* (27) *is not an R-deactivating direction for A.*

Proof Membership in NP is analogous to Theorem 15. Since conditions (a) and (b) from Definition 1 are satisfied, the question boils down to deciding whether $d \in M^*$. This cannot be reduced to the case in Corollary 2 because d in (27) has a special form.

To show hardness, we proceed by reduction from the 3-coloring problem [49, 63]: given a graph $G^* = (V^*, E^*)$, decide whether it is 3-colorable. Let $G = (V, C)$ be the graph sum (also known as the disjoint union of graphs) of G^* and K_4 [49, §8.1.2]. K_4 is the complete graph with 4 vertices. Informally, G is the graph obtained from G^* by adding 4 new vertices and including an edge between each pair of these new vertices.

Let CSP A have the structure (D, V, C) where $|D| = 3$ and

$$A = \{(\{i,j\}, (k_i, k_j)) \mid \{i,j\} \in C, k \in D^{\{i,j\}}, k_i \neq k_j\}. \tag{45}$$

Hence, any $x \in D^V$ can be interpreted as an assignment of colors to the nodes of G and $x \in \mathrm{SOL}(A)$ if and only if x is a 3-coloring of G. Since G contains K_4 as its subgraph, it is not 3-colorable and A is unsatisfiable. Hence, setting $R = A$ satisfies $\mathrm{SOL}(A - R) = \mathrm{SOL}(\emptyset) = \emptyset = \mathrm{SOL}(A)$.

For the purpose of our reduction, let us define $\delta = (|C| - 2)/2 > 0$. We will show that for such a setting, d is not an R-deactivating direction for A if and only if G^* is 3-colorable.

Plugging the above-defined sets A and R into the definition of d in (27) yields

$$\langle d, \phi(x)\rangle = \sum_{\substack{\{i,j\}\in C \\ x_i \neq x_j}} (-1) + \sum_{\substack{\{i,j\}\in C \\ x_i = x_j}} \delta = \delta(|C| - \mathrm{COL}(x)) - \mathrm{COL}(x) \tag{46}$$

where $\mathrm{COL}(x) = |\{\{i,j\} \in C \mid x_i \neq x_j\}|$ is the number of edges in G whose adjacent vertices have different colors in assignment $x \in D^V$.

If G^* is 3-colorable, then there is $x \in D^V$ such that $\mathrm{COL}(x) = |C| - 1$. In other words, only for a single edge in $C - E^*$ (i.e., edge of graph K_4), the adjacent vertices are assigned the same color, so $\langle d, \phi(x)\rangle = -|C|/2 < 0$ by (46) and definition of δ. Hence, d is not an R-deactivating direction for A.

For the other case, if G^* is not 3-colorable, then for any $x \in D^V$, $\mathrm{COL}(x) \leq |C| - 2$. The reason is that for at least one edge in K_4 and at least one edge in G^*, the adjacent vertices will be assigned the same color in any assignment. By substituting the value of δ and a simple manipulation of (46), one obtains

$$\langle d, \phi(x)\rangle = \delta|C| - (\delta + 1)\mathrm{COL}(x) = |C|\,(|C| - 2 - \mathrm{COL}(x))\,/2 \geq 0 \tag{47}$$

where the term in brackets is non-negative due to $\mathrm{COL}(x) \le |C| - 2$ for any $x \in D^V$. So, d is an R-deactivating direction for A. □

In connection to Section 5.1, a number of decision problems concerning the minimal CSP have been also proved hard. For recent results, see [64, 65].

7 Summary and discussion

We have proposed a method to compute upper bounds on the (maximization version of) WCSP. The WCSP is formulated as a linear program with an exponential number of constraints, whose feasible solutions are super-reparametrizations of the input WCSP instance (i.e., WCSP instances with the same structure and greater or equal objective values). Whenever the CSP formed by the active (i.e., maximal in their weight functions) tuples of a feasible WCSP instance is unsatisfiable, there exists an improving direction (in fact, a certificate of unsatisfiability of this CSP) for the linear program. As this approach provides only a subset of all possible improving directions, it can be seen as a local search. We showed how these improving directions can be generated by constraint propagation (or, more generally, by other methods to prove unsatisfiability of a CSP). We showed that super-reparametrizations are closely related to the dual cone to the well-known marginal polytope.

Special cases of our approach are the VAC / Augmenting DAG algorithm [2, 6, 16], which uses arc consistency, and the algorithm in [25], which uses cycle consistency. We have implemented the approach for singleton arc consistency, resulting in VSAC-SR algorithm. When compared to existing soft local consistency methods on a public dataset, VSAC-SR provides comparable or better bounds for many instances. Although the runtimes are higher than those of the simpler techniques, such as EDAC or VAC, one can control different trade-offs between bound quality and runtime by stopping the method prematurely, e.g., when the step size becomes small or terminating already with a greater value of θ (see Footnote 19).

The approach in general requires storing all the weights of the super-reparametrized WCSP instance. This may be a drawback when the domains are large and/or the weight functions are not given explicitly as a table of values but rather by an algorithm (oracle).

We expect our improved bounds to be useful when solving practical WCSP instances. Applications may include, e.g., using the method in preprocessing, pruning the search space during branch-and-bound search, providing tighter optimality gaps for solutions proposed by heuristic approaches, or generating high-quality proposals for solutions, as in [25]. However, we have done no experiments with this, so it is open whether the tighter bounds would outweigh the higher complexity of the algorithm. Due to the many options in which the method can be used, we leave this for future research. In addition, our approach can be also useful to solve more WCSP instances even without search (similarly, as the VAC algorithm solves all supermodular WCSPs without search) or, given a suitable primal heuristic, to solve WCSP instances approximately.

The approach can be straightforwardly extended to WCSPs with different domain sizes[26] and some weights equal to minus infinity (i.e., some constraints being hard). Of course, further experiments would be needed to evaluate the quality of the bounds if infinite weights are allowed. The WCSP framework also usually assumes a pre-defined specific finite bound that is updated during branch-and-bound [32] – although the presented pseudocode does not support this, it is not difficult to extend it in this way.

[26] In fact, our implementation already supports different domain sizes. We did not present our theoretical results for this generalized setting only to simplify notation.

Finally, we presented a theoretical analysis of the concept of super-reparametrizations of WCSPs, describing the properties of optimal super-reparametrizations and characterizing the set of active-tuple CSPs induced by different optimal super-reparametrizations. For example, even an optimal super-reparametrization may change the set of optimal assignments, as shown in Section 5.2. Additionally, we have shown that general (i.e., possibly non-optimal) super-reparametrizations are only weakly related to the original WCSP instance.

Acknowledgements Tomáš Dlask and Tomáš Werner were supported by the Czech Science Foundation (grant 19-09967S) and the OP VVV project CZ.02.1.01/0.0/0.0/16_019/0000765. Tomáš Dlask was also supported by the Grant Agency of the Czech Technical University in Prague (grants SGS19/170/OHK3/3T/13 and SGS22/061/OHK3/1T/13). Simon de Givry was supported by the French *Agence nationale de la Recherche* (ANR-19-P3IA-0004 ANITI).

Funding Open access publishing supported by the National Technical Library in Prague.

Declarations

Conflicts of interest The authors have no competing interests to declare.

Appendix: Example: EDAC, VAC, and VSAC

In this section, we present an example that shows how EDAC, VAC, and VSAC can be gradually enforced in a WCSP. As in [6, 14], we restrict this section to binary WCSPs and assume that $\{i\} \in C$ for each $i \in V$. Recall that VAC and VSAC were formally defined already in Remark 6: a WCSP $f \in \mathbb{R}^T$ is virtual Φ-consistent if $A^*(f)$ has a non-empty Φ-consistency closure. Now, we proceed to define EDAC in our formalism. For this purpose, let $E = \{S \in C \mid |S| = 2\}$ be the set of binary constraint scopes (as in Section 4.4 and Example 4) and $N_i = \{j \in V \mid \{i, j\} \in E\}$ be the set of neighbors of node i in the undirected graph (V, E).

Definition 2 [6, 14] Let $A \subseteq T$, $\{i, j\} \in E$, and $k_i \in D$. The tuple $(\{i\}, k_i)$ is *simply supported* by j in A if $\exists k_j \in D$ such that $(\{i, j\}, (k_i, k_j)) \in A$. The tuple $(\{i\}, k_i)$ is *fully supported* by j in A if $\exists k_j \in D$ such that $(\{i, j\}, (k_i, k_j)) \in A$ and $(\{j\}, k_j) \in A$.

Definition 3 [6, 14] Let $g \in \mathbb{R}^T$ be a binary WCSP and $\preceq$ be a total order on its set of variables V. WCSP g is *Existential Directional Arc Consistent (EDAC)* w.r.t. $\preceq$ if the following conditions hold:

- $\forall i \in V\ \forall k \in D\ \forall j \in N_i \colon i \preceq j \implies (\{i\}, k)$ is fully supported by j in $A^*(g)$,
- $\forall i \in V\ \forall k \in D\ \forall j \in N_i \colon j \preceq i \implies (\{i\}, k)$ is simply supported by j in $A^*(g)$,
- $\forall i \in V\ \exists k \in D \colon (\{i\}, k) \in A^*(g)$ and $\forall j \in N_i \colon (\{i\}, k)$ is fully supported by j in $A^*(g)$.

Remark 8 The notions of Definition 3 correspond to the notions in [6, 14] but they are tailored to our formalism. The first difference is that we do not consider infinite weights (i.e., hard constraints), which simplifies some conditions in the definition. The second difference is that the 'baseline' for a weight $g_t, t \in T_S$ is neither $\perp$ nor 0, but rather $\max_{t' \in T_S} g_{t'}$. Consequently, we require $g_t = \max_{t' \in T_S} g_{t'}$ (i.e., $t \in A^*(g)$) in the definitions instead of $g_t = 0$.

Remark 9 Let us also comment on the individual conditions in Definition 3. The first condition is known as Directional Arc Consistency w.r.t. $\preceq$ [6, 14]. The first and second condition together are known as Full Directional Arc Consistency w.r.t. $\preceq$ [6, 14]. Finally, the third condition is Existential Arc Consistency [6, 14].

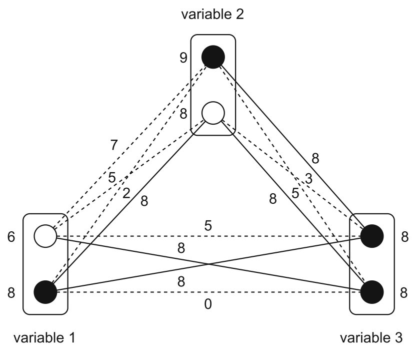
We now proceed to show our example where EDAC, VAC, and VSAC will be gradually enforced. The initial WCSP f^1 is depicted in Fig. 7a. The structure of this WCSP is (D, V, C) where $D = \{\mathsf{a}, \mathsf{b}\}$, $V = \{1, 2, 3\}$, and $C = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 3\}\}$. The names of the variables are indicated in the figures. To simplify the figures throughout this example, we do not state the names of the values in them – the upper value is a and the lower value is b. The optimal objective value of WCSP f^1 is 43, which is attained, e.g., by the assignment $x = (\mathsf{a}, \mathsf{a}, \mathsf{a})$.

WCSP f^1 is not EDAC (w.r.t. the natural ordering by $\leq$) because the tuple $(\{1\}, \mathsf{b})$ is not fully supported by variable 2. To make WCSP f^1 EDAC, it is sufficient to shift weight from the unary tuple $(\{2\}, \mathsf{a})$ to the binary weight function with scope $\{1, 2\}$, which results in WCSP f^2 depicted in Fig. 7b. WCSP f^2 is a reparametrization of f^1 due to $f^2 - f^1 \in M^\perp$ (depicted in Fig. 7c).

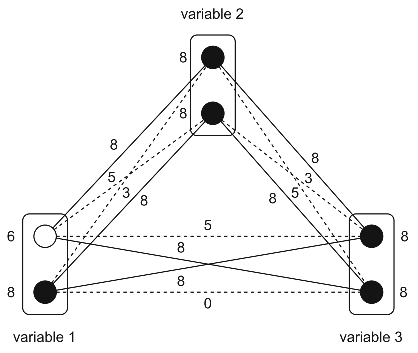
WCSP f^2 is EDAC w.r.t. $\leq$ but it is not VAC because the AC closure of $A^*(f^2)$ is empty. To see that the AC closure is empty, we can follow the propagations that are depicted in Fig. 7d. The arrows point from the cause of forbidding a tuple to the newly forbidden tuple. First, we can forbid the tuple $(\{1, 2\}, (\mathsf{a}, \mathsf{a}))$ because the tuple $(\{1\}, \mathsf{a})$ is forbidden. Second, we can forbid the tuple $(\{2\}, \mathsf{a})$ because both tuples $(\{1, 2\}, (\mathsf{a}, \mathsf{a}))$ and $(\{1, 2\}, (\mathsf{b}, \mathsf{a}))$ are forbidden. Next, we gradually forbid $(\{2, 3\}, (\mathsf{a}, \mathsf{a}))$, $(\{3\}, \mathsf{a})$, $(\{1, 3\}, (\mathsf{a}, \mathsf{b}))$, and $(\{3\}, \mathsf{b})$. This leads to domain wipe-out in variable 3. By shifting the weights against the direction of the arrows (as depicted in Fig. 7d), we make this WCSP VAC. This yields the WCSP f^3 in Fig. 7e which is a reparametrization of f^2. For clarity, we also show how the weights were transformed in Fig. 7f.

WCSP f^3 is VAC and even OSAC, so the bound $B(f^3)$ cannot be improved by reparametrizations (without introducing a ternary weight function with scope $\{1, 2, 3\}$). WCSP f^3 is however not VSAC because it has empty SAC closure. Thus, $A^*(f^3)$ is unsatisfiable and we are able to construct a super-reparametrization of WCSP f^3 with a better bound (recall Theorem 3 and Section 4.1.2). We show next the details of the construction, following our results from Sections 4.2, 4.3, and 4.4.

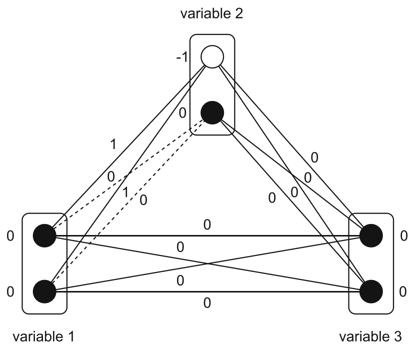
Figure 8a shows how arc consistency is enforced in the CSP $A^*(f^3)|_{x_1=\mathsf{a}}$ and the notation is analogous to Fig. 7d. In detail, we gradually forbid tuples $(\{3\}, \mathsf{a})$, $(\{2, 3\}, (\mathsf{a}, \mathsf{a}))$, $(\{2\}, \mathsf{a})$, and finally $(\{2\}, \mathsf{b})$, which leads to domain wipe-out in variable 2. Consequently, the AC closure of $A^*(f^3)|_{x_1=\mathsf{a}}$ is empty. To derive this, it suffices that the tuples $P = \{(\{1, 3\}, (\mathsf{a}, \mathsf{a})), (\{2, 3\}, (\mathsf{a}, \mathsf{b})), (\{1, 2\}, (\mathsf{a}, \mathsf{b}))\}$ are forbidden in $A^*(f^3)$. Applying Theorem 4 with $A = T - P \supseteq A^*(f^3)$ and $R = \{(\{1\}, \mathsf{a})\}$ results in $\{(\{1\}, \mathsf{a})\}$-deactivating direction d for $A^*(f^3)$ that is shown in Fig. 8b. Analogously, we can compute a $\{(\{1\}, \mathsf{b})\}$-deactivating direction d' for $A^*(f^3)$ (not shown). By summing these deactivating directions together (i.e., using Theorem 6 which in this case yields $\delta = 1$), we obtain a $\{(\{1\}, \mathsf{a}), (\{1\}, \mathsf{b})\}$-deactivating direction $d'' = d + d'$ for $A^*(f^3)$ that certifies unsatisfia-



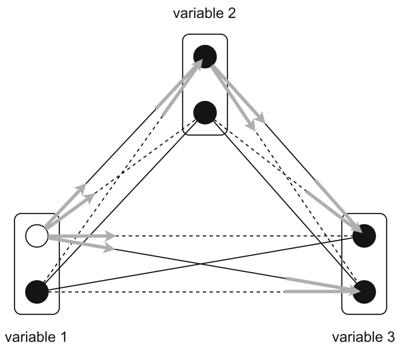
(a) WCSP f^1 with $B(f^1) = 49$. This WCSP is not EDAC w.r.t. $\leq$.



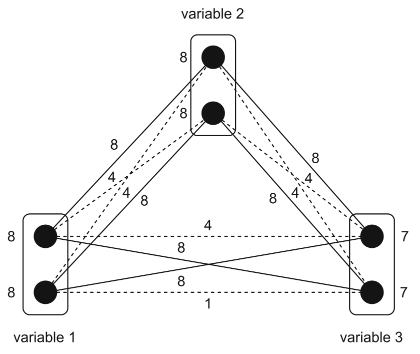
(b) WCSP f^2 with $B(f) = 48$. This WCSP is EDAC w.r.t. $\leq$ but not VAC.



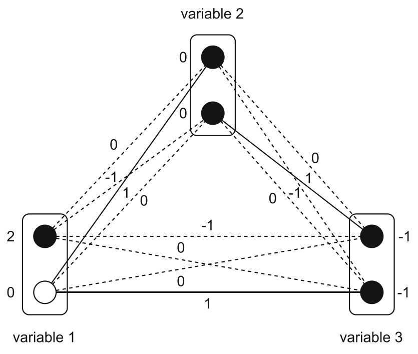
(c) WCSP $f^2 - f^1 \in M^{\perp}$.



(d) Propagation of arc consistency in CSP $A^*(f^2)$.

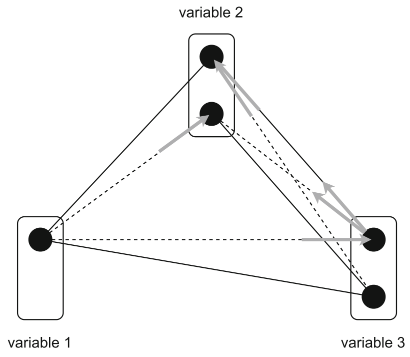


(e) WCSP f^3 with $B(f) = 47$. This WCSP is VAC but not VSAC.

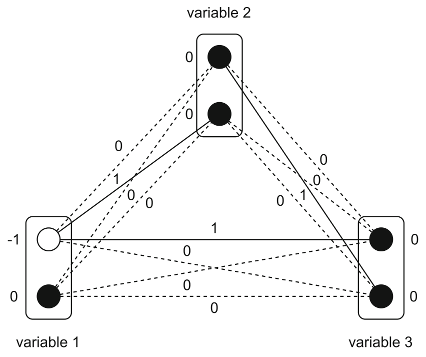


(f) WCSP $f^3 - f^2 \in M^{\perp}$.

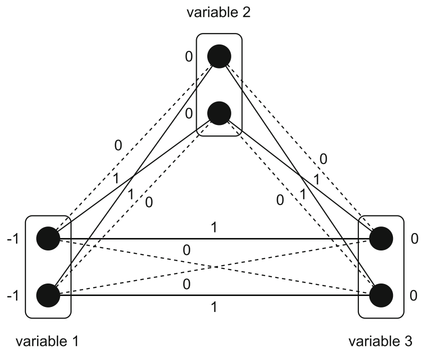
Fig. 7 Enforcing EDAC and VAC in a WCSP via reparametrizations



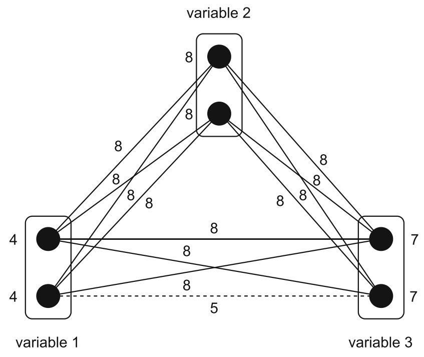
(a) Propagation of arc consistency in CSP $A^*(f^3)|_{x_1=\mathsf{a}}$.



(b) $\{(\{1\},\mathsf{a})\}$-deactivating direction d for CSP $A^*(f^3)$.



(c) $\{(\{1\},\mathsf{a}),(\{1\},\mathsf{b})\}$-deactivating direction d'' for CSP $A^*(f^3)$.



(d) WCSP $f^4 = f^3 + 4d''$ with $B(f^4) = 43$. This WCSP is VSAC.

Fig. 8 Enforcing VSAC via super-reparametrizations

bility of $A^*(f^3)$. By Theorem 7, we compute the step size $\alpha = \min\{\beta, \gamma\} = 4$ and obtain WCSP $f^4 = f^3 + \alpha d''$ which is shown in Fig. 8d and is VSAC.

References

1. Schlesinger, M. (1976). Sintaksicheskiy analiz dvumernykh zritelnikh signalov v usloviyakh pomekh (Syntactic analysis of two-dimensional visual signals in noisy conditions). *Kibernetika, 4*(113–130), 2.
2. Werner, T. (2007). A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 29*(7), 1165–1179.
3. Wainwright, M. J., & Jordan, M. I. (2008). Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning, 1*(1–2), 1–305.
4. Živný, S. (2012). *The Complexity of Valued Constraint Satisfaction Problems*. Cognitive Technologies: Springer.
5. Savchynskyy, B. (2019). Discrete graphical models - an optimization perspective. *Foundations and Trends in Computer Graphics and Vision, 11*(3–4), 160–429.
6. Cooper, M. C., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., & Werner, T. (2010). Soft arc consistency revisited. *Artificial Intelligence, 174*(7–8), 449–478.
7. Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*(10), 1568–1583.
8. Globerson, A., Jaakkola, T.S. (2008). Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in neural information processing systems* (pp. 553–560)
9. Tourani, S., Shekhovtsov, A., Rother, C., Savchynskyy, B. (2018). MPLP++: Fast, parallel dual block-coordinate ascent for dense graphical models. In *Proceedings of the European conference on computer vision* (pp. 251–267)
10. Tourani, S., Shekhovtsov, A., Rother, C., Savchynskyy, B. (2020) Taxonomy of dual block-coordinate ascent methods for discrete energy minimization. In *International conference on artificial intelligence and statistics* (pp. 2775–2785). PMLR
11. Werner, T. (2010). Revisiting the linear programming relaxation approach to gibbs energy minimization and weighted constraint satisfaction. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 32*(8), 1474–1488.
12. Kolmogorov, V. (2014). A new look at reweighted message passing. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 37*(5), 919–930.
13. Werner, T., Průša, D., Dlask, T. (2020). Relative interior rule in block-coordinate descent. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 7559–7567)
14. de Givry, S., Heras, F., Zytnicki, M., Larrosa, J. (2005). Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *IJCAI* (vol. 5, pp. 84–89)
15. Cooper, M.C., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M. (2008). Virtual arc consistency for weighted CSP. In *Proceedings of the 22nd AAAI conference on artificial intelligence* (pp. 253–258)
16. Koval, V. K., & Schlesinger, M. I. (1976). Dvumernoe programmirovanie v zadachakh analiza izobrazheniy (Two-dimensional Programming in Image Analysis Problems). *Automatics and Telemechanics, 8*, 149–168. In Russian.
17. Werner, T. (2005). A Linear Programming Approach to Max-sum Problem: A Review. Center for Machine Perception, Czech Technical University. CTU-CMP-2005-25
18. Cooper, M.C., de Givry, S., Schiex, T. (2007). Optimal soft arc consistency. In *Proceedings of the 20th international joint conference on artifical intelligence* (vol. 7, pp. 68–73)
19. Průša, D., & Werner, T. (2015). Universality of the Local Marginal Polytope. *IEEE Trans on Pattern Analysis and Machine Intelligence, 37*(4), 898–904.
20. Průša, D., & Werner, T. (2019). Solving LP relaxations of some NP-hard problems is as hard as solving any linear program. *SIAM J Optimization, 29*(3), 1745–1771.
21. Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T., Weiss, Y. Tightening LP Relaxations for MAP using Message Passing. Citeseer
22. Nguyen, H., Bessiere, C., de Givry, S., & Schiex, T. (2017). Triangle-based consistencies for cost function networks. *Constraints, 22*(2), 230–264.
23. Batra, D., Nowozin, S., Kohli, P. (2011). Tighter relaxations for MAP-MRF inference: A local primal-dual gap based separation algorithm. In *Proceedings of the Fourteenth international conference on artificial intelligence and statistics* (pp. 146–154)
24. Thapper, J., Živný, S. (2015). Sherali-Adams relaxations for valued CSPs. In *International colloquium on automata, languages, and programming* (pp. 1058–1069). Springer

25. Komodakis, N., Paragios, N. (2008) Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles. In *European conference on computer vision* (pp. 806–820). Springer
26. Bessiere, C., & Debruyne, R. (2008). Theoretical analysis of singleton arc consistency and its extensions. *Artificial Intelligence, 172*(1), 29–41.
27. Sontag, D., Jaakkola, T. (2009). Tree block coordinate descent for MAP in graphical models. In *Artificial intelligence and statistics* (pp. 544–551)
28. Dlask, T., Werner, T., de Givry, S. (2021). Bounds on weighted CSPs using constraint propagation and super-reparametrizations. In L.D. Michel, (Eds.) *27th international conference on principles and practice of constraint programming (CP 2021). vol. 210 of Leibniz International Proceedings in Informatics (LIPIcs)* (pp. 23:1–23:18). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik
29. Thapper, J., & Živný, S. (2016). The complexity of finite-valued CSPs. *Journal of the ACM (JACM), 63*(4), 1–33.
30. Kolmogorov, V., Thapper, J., & Živný, S. (2015). The power of linear programming for general-valued CSPs. *SIAM Journal on Computing, 44*(1), 1–36.
31. Kappes, J. H., Andres, B., Hamprecht, F. A., Schnörr, C., Nowozin, S., Batra, D., et al. (2015). A Comparative Study of Modern Inference Techniques for Structured Discrete Energy Minimization Problems. *Intl. J. of Computer Vision, 115*(2), 155–184.
32. Cooper, M.C., de Givry, S., Schiex, T. (2020) Valued Constraint Satisfaction Problems. In *A guided tour of artificial intelligence research* (pp. 185–207). Springer
33. Zalinescu, C. (2002). *Convex Analysis in General Vector Spaces*. World Scientific.
34. Cooper, M. C. (2004). Cyclic consistency: a local reduction operation for binary valued constraints. *Artificial Intelligence, 155*(1–2), 69–92.
35. Jahn, J., & Ha, T. X. D. (2011). New order relations in set optimization. *Journal of Optimization Theory and Applications, 148*(2), 209–236.
36. Boyd, S., Vandenberghe. L. (2004) Convex optimization. Cambridge university press
37. Werner, T. (2015). Marginal consistency: upper-bounding partition functions over commutative semirings. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 37*(7), 1455–1468.
38. Dlask, T., Werner, T. (2020). Bounding linear programs by constraint propagation: application to Max-SAT. In *International conference on principles and practice of constraint programming* (pp. 177–193). Springer
39. Grégoire, É., Mazure, B., Piette, C. (2007). MUST: Provide a finer-grained explanation of unsatisfiability. In *International conference on principles and practice of constraint programming* (pp. 317–331). Springer
40. Grégoire, E., Mazure, B., & Piette, C. (2008). On finding minimally unsatisfiable cores of CSPs. *International Journal on Artificial Intelligence Tools, 17*(04), 745–763.
41. Papadimitriou, C.H., Wolfe, D. (1985) The complexity of facets resolved. Cornell University
42. Freuder, E.C., Elfe, C.D. (1996) Neighborhood inverse consistency preprocessing. In *AAAI/IAAI* (Vol 1, pp. 202–208)
43. Bessiere, C. (2006). *Constraint propagation*. In Handbook of constraint programming: Elsevier.
44. Bessiere, C., Cardon, S., Debruyne, R., & Lecoutre, C. (2011). Efficient algorithms for singleton arc consistency. *Constraints, 16*(1), 25–53.
45. Dlask, T. (2018). *Minimizing Convex Piecewise-Affine Functions by Local Consistency Techniques [Master's thesis]*. Faculty of Electrical Engineering: Czech Technical University in Prague.
46. Larrosa, J., & Schiex, T. (2004). Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence, 159*(1–2), 1–26.
47. Ahuja, R.K., Magnanti, T.L., Orlin, J.B. (1993) Network flows: Theory, applications and algorithms
48. Dlask, T. (2022). Block-Coordinate Descent and Local Consistencies in Linear Programming [Dissertation, available online: https://dspace.cvut.cz/handle/10467/102874?locale-attribute=en]. Czech Technical University in Prague, Faculty of Electrical Engineering;
49. Rosen, K. H., & Michaels, J. G. (2000). Handbook of Discrete and Combinatorial Mathematics. *Boca Raton, CRC Press, 1232*, 2000.
50. Debruyne, R., Bessiere, C. (1997). Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97* (pp. 412–417)
51. Available online.: toulbar2. https://miat.inrae.fr/toulbar2. Accessed 12 Jan 2021
52. Available online.: Cost Function Library benchmark. https://forgemia.inra.fr/thomas.schiex/cost-function-library, commit 356bbb85. Accessed 12 Jan 2021
53. Available online.: Spin Glass Server. https://software.cs.uni-koeln.de/spinglass, recently moved to http://spinglass.uni-bonn.de/. Accessed 12 Jan 2023
54. Cooper, M. C., de Roquemaurel, M., & Régnier, P. (2011). A weighted CSP approach to cost-optimal planning. *AI Communications, 24*(1), 1–29.

55. Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., et al. (2019). QPLIB: a library of quadratic programming instances. *Mathematical Programming Computation, 11*(2), 237–265.
56. Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences, 7*, 95–132.
57. Dechter, R., Cohen, D., et al. (2003). *Constraint processing*. Morgan Kaufmann.
58. Astesana, J., Cosserat, L., Fargier, H. (2010) Constraint-based vehicle configuration: A case study. In *2010 22nd IEEE international conference on tools with artificial intelligence* (vol 1, pp 68–75)
59. Bessiere, C., Fargier, H., Lecoutre, C. (2013). Global inverse consistency for interactive constraint satisfaction. In Schulte, C. (Es), *Principles and practice of constraint programming* (pp. 159–174). Berlin, Springer
60. Davey, B. A., & Priestley, H. A. (2002). *Introduction to lattices and order*. Cambridge University Press.
61. Blyth, T. S. (2005). *Lattices and Ordered Algebraic Structures*. Springer, London: Universitext.
62. Alsuwaiyel, M. (1999). Algorithms: Design Techniques and Analysis. World Scientific
63. Karp, RM. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer
64. Gottlob, G. (2012). On minimal constraint networks. *Artificial Intelligence, 191*, 42–60.
65. Escamocher, G., & O'Sullivan, B. (2018). Pushing the frontier of minimality. *Theoretical Computer Science, 745*, 172–201.