Bounds on Weighted CSPs Using Constraint Propagation and Super-Reparametrizations

Tomáš Dlask[1] ✉ 🄪
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Tomáš Werner ✉ 🄪
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Simon de Givry ✉ 🄪
Université Fédérale de Toulouse, ANITI, INRAE, UR 875, France

─── **Abstract** ───

We propose a framework for computing upper bounds on the optimal value of the (maximization version of) Weighted CSP (WCSP) using super-reparametrizations, which are changes of the weights that keep or increase the WCSP objective for every assignment. We show that it is in principle possible to employ arbitrary (under certain technical conditions) constraint propagation rules to improve the bound. For arc consistency in particular, the method reduces to the known Virtual AC (VAC) algorithm. Newly, 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.

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 small) subset of the variables. A popular approach to tackle this NP-hard combinatorial optimization problem is via its linear programming (LP) relaxation [21, 32, 30, 29, 20, 1]. The dual of this LP relaxation minimizes an upper bound on the WCSP optimal value over reparametrizations (also known as equivalence-preserving transformations) of the original WCSP instance. For large instances this is done only approximately, by methods based on block-coordinate descent [16, 14, 27, 28, 32, 17] or constraint propagation [5, 18, 32, 19]. Fixed points of these methods are characterized by a local consistency of the CSP formed by the active tuples (to be defined later) of the transformed WCSP [32, 16, 14, 5, 27].

---

[1] Corresponding author

This approach is limited in that it cannot enforce an arbitrary level of local consistency, unless new weight functions are introduced. Namely, it can achieve at most pairwise consistency [33, 34], which for binary WCSPs reduces to arc consistency (reparametrized WCSP corresponding to global optima of the dual LP relaxation have been called *optimally soft arc consistent* in [6, 5]).

In this paper, we study a different LP formulation of the WCSP, which was proposed in [17] but never pursued later. It differs from the above mentioned basic LP relaxation and does not belong to the known hierarchy of LP relaxations obtained by introducing new weight functions of higher arities [24, 33, 19, 1] (which leads to a fine-grained version of the Sherali-Adams hierarchy [22] for the WCSP). Our LP formulation again minimizes the upper bound on the WCSP optimal value, but this time over *super-reparametrizations* of the initial WCSP instance. Its remarkable feature is that it allows using almost arbitrary (up to some technical assumptions) constraint propagation techniques to improve the bound, without introducing new weight functions. On the other hand, it may neither preserve the value of the individual assignments nor the set of optimal assignments, but it nevertheless provides a valid, and possibly tighter, bound on the WCSP optimal value.

2   Notation

For clarity of presentation, we will consider only binary WCSPs with finite weights. However, it would be straightforward to generalize the approach described in the paper to WCSPs of any arity and with some weights infinite (i.e., including hard constraints).

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 $E \subseteq \binom{V}{2}$ be a set of variable pairs, so that (V, E) is an undirected graph. The *weighted constraint satisfaction problem (WCSP)* seeks to maximize the function

$$F(x \,|\, f) = \sum_{i \in V} f_i(x_i) + \sum_{\{i,j\} \in E} f_{ij}(x_i, x_j) \tag{1}$$

over all assignments $x \in D^V$. Here, $f_i \colon D \to \mathbb{R}$ and $f_{ij} \colon D^2 \to \mathbb{R}$ (where we assume that $f_{ij}(k, l) = f_{ji}(l, k)$) are *weight functions*, whose values together form a vector $f \in \mathbb{R}^T$ where

$$T = \underbrace{\{\, (i,k) \mid i \in V, \ k \in D \,\}}_{V \times D} \cup \{\, \{(i,k),(j,l)\} \mid \{i,j\} \in E, \ k,l \in D \,\} \tag{2}$$

is a set of *tuples*. For $t \in T$, we denote $f_t = f_i(k)$ if $t = (i,k) \in V \times D$, and $f_t = f_{ij}(k,l) = f_{ji}(l,k)$ if $t = \{(i,k),(j,l)\} \in T - (V \times D)$. The WCSP instance is defined by (D, V, E, f). However, as the structure (D, V, E) will be the same throughout the paper, we will refer to WCSP instances only as f (thus, we identify WCSP instances with vectors $f \in \mathbb{R}^T$).

We say that an assignment $x \in D^V$ *uses* a tuple $t = (i,k)$ if $x_i = k$, and x *uses* a tuple $t = \{(i,k),(j,l)\}$ if $x_i = k$ and $x_j = l$. In the *constraint satisfaction problem (CSP)*, we are given a set $A \subseteq T$ of *allowed tuples* (while the tuples $T - A$ are called *forbidden*) and look for an assignment x (a *solution* to the CSP) that uses only the allowed tuples, i.e., $(i, x_i) \in A$ for all $i \in V$ and $\{(i, x_i), (j, x_j)\} \in A$ for all $\{i, j\} \in E$. The CSP is *satisfiable* if it has a solution. The CSP instance is defined by (D, V, E, A) but, as (D, V, E) will be always the same, we will refer to it only as A (i.e., we identify CSP instances with subsets of T).

---

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

For a tuple $t \in T$, we denote

$$
U(t) = \begin{cases} \{\, (i, k') \mid k' \in D \,\} & \text{if } t = (i, k) \\ \{\, \{(i, k'), (j, l')\} \mid k', l' \in D \,\} & \text{if } t = \{(i, k), (j, l)\} \end{cases}
$$

so that, e.g., for all $i \in V$ and $k, k' \in D$ we have $U((i, k)) = U((i, k'))$. By

$$
\mathcal{U} = \{\, U(t) \mid t \in T \,\} = \{\, \{(i, k) \mid k \in D\} \mid i \in V \,\} \cup \{\, \{\{(i, k), (j, l)\} \mid k, l \in D\} \mid \{i, j\} \in E \,\}
$$

we denote the natural partition of T into $|V| + |E|$ subsets. Clearly, any assignment uses exactly one tuple from each set $U \in \mathcal{U}$.

For a CSP $A \subseteq T$ and $t \in T$, we denote $A|_t = A - (U(t) - \{t\})$. That is, x is a solution to CSP $A|_t$ if and only if x is a solution to CSP A and x uses tuple t. E.g., in $A|_{(i,k)}$ we search for solutions x to CSP A satisfying $x_i = k$ (this is often denoted also as $A|_{x_i=k}$).

3 Bounding the WCSP Optimal Value

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

$$
B(f) = \sum_{i \in V} \max_{k \in D} f_i(k) + \sum_{\{i,j\} \in E} \max_{k,l \in D} f_{ij}(k, l) = \sum_{U \in \mathcal{U}} \max_{t \in U} f_t. \tag{3}
$$

For $f \in \mathbb{R}^T$, we call a tuple $t \in T$ *active* if $f_t = \max_{t' \in U(t)} f_{t'}$. Thus, a tuple $t = (i, k) \in T$ is active if $f_i(k) = \max_{k' \in D} f_i(k')$, and a tuple $t = \{(i, k), (j, l)\} \in T$ is active if $f_{ij}(k, l) = \max_{k', l' \in D} f_{ij}(k', l')$. The set of all tuples that are active for f is denoted[3] by $A^*(f) \subseteq T$.

▶ **Theorem 1** ([32]). *For every WCSP $f \in \mathbb{R}^T$ and every assignment $x \in D^V$ we have:*
(a) $B(f) \geq F(x \mid f)$,
(b) $B(f) = F(x \mid f)$ *if and only if x is a solution to CSP $A^*(f)$.*

Proof. (a) can be checked by comparing expressions (1) and (3) term by term.

(b) says that $B(f) = F(x \mid f)$ if and only if assignment x uses only the active tuples of f. This is again straightforward from (1) and (3). ◀

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

3.1 Minimizing the Upper Bound by Reparametrizations

If WCSPs $f, g \in \mathbb{R}^T$ satisfy $F(x \mid f) = F(x \mid g)$ for all $x \in D^V$, we say that f is a *reparametrization* of g (or *equivalent* to g or an equivalence-preserving transformation of g) [16, 30, 21, 32, 33, 20, 6, 5, 28]. As the function $F(x \mid f)$ is linear in f for every x, equality $F(x \mid f) = F(x \mid g)$ can be written as $F(x \mid f - g) = 0$. Linearity of $F(x \mid \cdot)$ also implies that the set $\{\, h \in \mathbb{R}^T \mid F(x \mid h) = 0 \; \forall x \in D^V \,\}$ is a subspace[4] of $\mathbb{R}^T$.

---

[3] The set of active tuples $A^*(f)$ corresponds to the notion of $\mathrm{Bool}(f)$ in [5]. The characteristic vector of the set $A^*(f)$ was denoted $\bar{f}$ in [27, 32], $\lceil f \rceil$ in [33], and $\mathrm{mi}[f]$ in [20].

[4] For binary WCSPs with a connected graph (V, E), this subspace can be parametrized as $h_i(k) = \sum_{j \mid \{i,j\} \in E} \varphi_{ij}(k)$ and $h_{ij}(k, l) = -\varphi_{ij}(k) - \varphi_{ji}(l)$ where $\varphi_{ij}, \varphi_{ji} \colon D \to \mathbb{R}$, $\{i, j\} \in E$, are arbitrary unary weight functions [32, Theorem 3]. For WCSPs of any arity, see [33, §3.2].

Given a WCSP $g \in \mathbb{R}^T$, this suggests to minimize the upper bound on its optimal value $\max_x F(x\,|\,g)$ by reparametrizations:

$$\min_{f \in \mathbb{R}^T} B(f) \qquad \text{subject to} \qquad F(x\,|\,f) = F(x\,|\,g) \ \ \forall x \in D^V. \qquad (4)$$

Although this problem has an exponential number of constraints, its feasible set is an affine subspace of $\mathbb{R}^T$ and thus the number of constraints can be reduced to polynomial. Using the well-known trick, the problem can be transformed to a linear program[5], which is the *dual LP relaxation* of the WCSP g [21, 32, 20]. WCSPs f optimal for (4) have been called *optimally soft arc consistent (OSAC)* in [6, 5]. If any optimal solution f to (4) satisfies $B(f) = F(x\,|\,f)$ for some x, the LP relaxation is tight. Otherwise, $B(f)$ is only a strict upper bound on the optimal value of WCSP g.

3.2   Minimizing the Upper Bound by Super-Reparametrizations

If WCSPs $f, g \in \mathbb{R}^T$ satisfy $F(x\,|\,f) \geq F(x\,|\,g)$ (that is, $F(x\,|\,f-g) \geq 0$) for all $x \in D^V$, we say that f is a *super-reparametrization*[6] of g. The set $\{\, h \in \mathbb{R}^T \mid F(x\,|\,h) \geq 0 \ \forall x \in D^V \,\}$ is a polyhedral convex cone. Following [17], we consider the problem

$$\min_{f \in \mathbb{R}^T} B(f) \qquad \text{subject to} \qquad F(x\,|\,f) \geq F(x\,|\,g) \ \ \forall x \in D^V. \qquad (5)$$

▶ **Theorem 2** ([17]). *The optimal value of problem* (5) *is* $\max_{x \in D^V} F(x\,|\,g)$.

Proof. By Theorem 1(a), every feasible f satisfies

$$B(f) \geq F(x\,|\,f) \geq F(x\,|\,g) \qquad \forall x \in D^V. \qquad (6)$$

Denoting $F^* = \max_x F(x\,|\,g)$, this implies $B(f) \geq F^*$. To see that this bound is attained, consider f defined by $f_t = F^*/(|V| + |E|)$ for all $t \in T$. It can be checked from (1) and (3) that $B(f) = F(x\,|\,f) = F^*$ for all x, so f is feasible and optimal.   ◀

Theorem 2 says that any feasible solution f to (5) yields an upper bound $B(f)$ on the optimal value of WCSP g, which is attained if f is optimal for (5). Thus, finding a global optimum of (5) in fact means solving the WCSP g. This is not surprising, as the complexity of the WCSP is hidden in the exponential set of constraints of (5).

▶ **Theorem 3.** *Let* $g \in \mathbb{R}^T$. *Let* $f \in \mathbb{R}^T$ *be feasible for* (5). *Then* f *is optimal for* (5) *if and only if CSP* $A^*(f)$ *has a solution* x *satisfying* $F(x\,|\,f) = F(x\,|\,g)$.

Proof. By (6) and Theorem 2, a feasible f is optimal if and only if $B(f) = F(x\,|\,f) = F(x\,|\,g)$ for some x. The claim now follows from Theorem 1.   ◀

Next, we state a useful corollary of Theorem 3.

▶ **Theorem 4.** *Let* $g \in \mathbb{R}^T$. *CSP* $A^*(g)$ *is unsatisfiable if and only if there exists* $h \in \mathbb{R}^T$ *such that* $B(g + h) < B(g)$ *and* $F(x\,|\,h) \geq 0$ *for all* $x \in D^V$.

---

[5] Namely, by introducing auxiliary variables $z_U \in \mathbb{R}$, problem (4) is equivalent to minimizing $\sum_{U \in \mathcal{U}} z_U$ subject to $z_U \geq f_t \ \forall U \in \mathcal{U}, \ t \in U$ and $F(x\,|\,f) = F(x\,|\,g) \ \forall x \in D^V$, which is a linear program.
[6] They are called *sup-reparametrizations* in [23] and *virtual potentials* in [17].

Proof. Theorem 3 in particular says that problem (5) attains its optimum at $f = g$ if and only if $A^*(g)$ is satisfiable. That is, $A^*(g)$ is unsatisfiable if and only if there exists $f \in \mathbb{R}^T$ such that $B(f) < B(g)$ and $F(x \mid f) - F(x \mid g) = F(x \mid f - g) \geq 0$ for all x. Substituting $h = f - g$ yields the desired claim. ◀

We will refer to vector h in Theorem 4 as a *certificate of unsatisfiability of CSP $A^*(g)$ for g*. Note that "for g" is important because h depends not only on the set $A^*(g)$ but also on the vector g itself. We will discuss this in more detail later on in §3.2.1.

3.2.1 Iterative Scheme

Given a feasible solution f to (5), we call a vector $h \in \mathbb{R}^T$ an *improving vector* if vector $f' = f + h$ is feasible for (5) and $B(f') < B(f)$. Clearly, an improving vector exists if and only if f is not optimal for (5). Theorem 3 says that for f to be optimal, it is necessary (but not sufficient) that CSP $A^*(f)$ is satisfiable. Therefore, if $A^*(f)$ is unsatisfiable, there exists an improving vector. Any certificate h of unsatisfiability of $A^*(f)$ for f (i.e., h satisfies $B(f + h) < B(f)$ and $F(x \mid h) \geq 0$ for all x) is such an improving vector: indeed, $f' = f + h$ is feasible for (5) because $F(x \mid f') = F(x \mid f) + F(x \mid h) \geq F(x \mid f) \geq F(x \mid g)$ for all x.

This suggests an iterative scheme to progressively improve feasible solutions to (5): initialize $f := g$ and then repeat this iteration (see Figure 1 in the appendix for an example):
1. If CSP $A^*(f)$ is satisfiable, stop. Otherwise, find a certificate h of unsatisfiability of $A^*(f)$ for f.
2. Update $f := f + h$.

Recall that satisfiability of $A^*(f)$ is not sufficient for optimality of f, as we are neglecting the (difficult) condition $F(x \mid f) = F(x \mid g)$ in Theorem 3. Consequently, in Step 1 we are able to generate only improving vectors h satisfying $F(x \mid h) \geq 0$ for all x, while general improving vectors (as defined above) need to satisfy only $F(x \mid f + h) \geq F(x \mid g)$ for all x. The former condition implies the latter but not *vice versa*. Therefore, fixed points of the algorithm are *local minima* of problem (5), in the sense that a fixed point cannot be improved by moving in any direction h satisfying the former condition (but possibly can be improved by moving in a direction h satisfying the latter condition).

During the algorithm, this manifests itself as follows. At any time, f satisfies (6), hence also $B(f) \geq \max_x F(x \mid f) \geq \max_x F(x \mid g)$. In every iteration, $B(f)$ decreases and the number $\max_x F(x \mid f)$ increases or does not change (due to $F(x \mid h) \geq 0$ for all x). When these two numbers meet, $A^*(f)$ becomes satisfiable by Theorem 1 and the algorithm stops. Monotonic increase of $\max_x F(x \mid f)$ can be seen as "greediness" of the algorithm: if we could generate general improving vectors, $\max_x F(x \mid f)$ could also decrease. Any increase of $\max_x F(x \mid f)$ is undesirable because the bound $B(f)$ will never be able to get below it.

Due to this greediness, the achievable (i.e., after possible convergence) gap $B(f) - \max_x F(x \mid g)$ critically depends on the "quality" of the certificates h. For a given f, there can be many certificates h of unsatisfiability of $A^*(f)$ for f. Good certificates are those for which the difference $\max_x F(x \mid f + h) - \max_x F(x \mid f) \geq 0$ is small (ideally zero). Intuitively, this means $F(x \mid h)$ should be zero for most of the assignments x and small for the remaining assignments. In turn, one heuristic for this is to keep vector h sparse[7].

---

[7] Sparsity of h is not the whole answer, though, because, e.g., vectors h satisfying $F(x \mid h) = 0$ for all x can be dense, according to Footnote 4.

So far, we supposed that in Step 1 of the algorithm we were always able to decide if CSP $A^*(f)$ is satisfiable. This is unrealistic because the CSP is NP-complete. But the approach remains applicable even if we detect unsatisfiability of $A^*(f)$ (and provide a certificate h) only sometimes, e.g., using constraint propagation. Then the iterative scheme becomes:

1. Attempt to prove that CSP $A^*(f)$ is unsatisfiable. If we succeed, find a certificate h of unsatisfiability of $A^*(f)$ for f. If we fail, stop.
2. Update $f := f + h$.

In this case, the fixed points of the algorithm will be even weaker local minima of problem (5), but they nevertheless might be still non-trivial and useful.

In the rest of the paper we develop this approach in detail. More precisely, we will compute an improving vector h in two steps: first (in §4) we compute an *improving direction* $d \in \mathbb{R}^T$ from $A^*(f)$ using constraint propagation, and then (in §5) we compute a suitable *step length* $\alpha > 0$ such that $h = \alpha d$. This is because from the CSP $A^*(f)$ alone it is possible to obtain only improving directions, while the step α depends also on f.[8]

Relation to Existing Approaches

The Augmenting DAG algorithm [18, 32] and the VAC algorithm [5] are (up to the precise way of computing the certificates h) an example of the described approach, which uses arc consistency to prove unsatisfiability of $A^*(f)$. In this favorable case, there exist certificates h that satisfy $F(x|h) = 0$ for all x, so we are in fact solving (4) rather than (5). For stronger local consistencies such certificates in general do not exist (i.e., $F(x|h) > 0$ for some x).

The algorithm proposed in [17] 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 (1) and minimizing an upper bound by reparametrizations, as in [24, 1, 33, 34, 19]. In our particular case, after updating $f := f + h$ we could introduce[9] a weight function with scope formed by the variables of all tuples $t \in T$ with $h_t \neq 0$. 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 [12] to optimize linear programs using constraint propagation.

4 Deactivating Directions

Here we describe a special kind of directions, *deactivating directions* (this name will be justified in §5). Under additional conditions, these directions certify unsatisfiability of a CSP.

▶ **Definition 5.** *Let $A \subseteq T$ and $S \subseteq A$, $S \neq \emptyset$. An S-deactivating direction for CSP A is a vector $d \in \mathbb{R}^T$ satisfying*
(a) $d_t < 0$ *for all $t \in S$,*
(b) $d_t = 0$ *for all $t \in A - S$,*
(c) $F(x|d) \geq 0$ *for all $x \in D^V$.* ⌟

Note that for fixed A and S, all S-deactivating directions for A form a convex cone.

---

[8] It follows from Theorem 4, 6, and 15 that a CSP $A \subseteq T$ is unsatisfiable if and only if there is a direction $d \in \mathbb{R}^T$ such that (i) $F(x|d) \geq 0$ for all x and (ii) for every $f \in \mathbb{R}^T$ such that $A = A^*(f)$ there exists $\alpha > 0$ such that $B(f + \alpha d) < B(f)$.

[9] Notice that such an added weight function would not increase the bound (3) since its weights are non-positive due to the fact that it needs to decrease the value for some assignments.

▶ **Theorem 6.** *Let $A \subseteq T$ and $S \subseteq A$, $S \neq \emptyset$. An S-deactivating direction $d \in \mathbb{R}^T$ for A exists if and only if CSP $A|_t$ has no solution for any $t \in S$.*

Proof. For one direction, we proceed by contradiction. Let d be an S-deactivating direction for A and let x^* be a solution to A that uses at least one tuple from S. By (1), we have $F(x^*|d) < 0$ because $d_t = 0$ for all $t \in A - S$ by condition (b) in Definition 5 and $d_{t^*} < 0$ for all $t^* \in S$ by condition (a). This contradicts condition (c).

For the other direction, if CSP $A|_t$ has no solution for any $t \in S$, sets S and $P = T - A$ satisfy Property 7, so an S-deactivating direction exists by Theorem 8 (given below). ◀

Suppose $d \in \mathbb{R}^T$ is an S-deactivating direction for $A \subseteq T$. If for some $U \in \mathcal{U}$ it holds that $(A - S) \cap U = \emptyset$ (equivalently[10], $A \cap U \subseteq S$) then, by Theorem 6, CSP A is unsatisfiable because (as noted in §2) every assignment uses exactly one tuple from every set from $\mathcal{U}$. In that case, d certifies unsatisfiability of CSP A.[11]

4.1 Obtaining Deactivating Directions Using Constraint Propagation

Although the CSP is NP-complete, satisfiability of some CSPs can be disproved in polynomial time by constraint propagation. This is an iterative method, which in each iteration infers that certain tuples of a given CSP instance are not used by any solution and forbids these tuples. In contrast to usual usage of constraint propagation, we require that in every iteration it also provides an S-deactivating direction for the set of tuples S it forbids. By Theorem 6, such a direction always exists.

We will argue that finding an S-deactivating direction for a CSP A is not harder than infering that $A|_t$ has no solution for any $t \in S$. Formally, we consider an algorithm (such as a constraint propagation method) satisfying the following property:

▶ **Property 7.** *The algorithm takes a set $A \subseteq T$ on input and returns sets $S \subseteq A$ and $P \subseteq T - A$ such that CSP $(T - P)|_t$ is unsatisfiable for every $t \in S$.* ⌟

The condition in Property 7 is equivalent to requiring that for any CSP A' where all tuples from P are forbidden (i.e., $A' \subseteq T - P$), $A'|_t$ is unsatisfiable for all $t \in S$.[12] Note that this implies that CSP $A|_t$ is unsatisfiable for all $t \in S$ due to $A \subseteq T - P$.

Returning $S = \emptyset$ indicates that the algorithm is not able to forbid any tuple. In addition, the algorithm provides a "proof" set $P \subseteq T - A$ which can be interpreted as a set of tuples which are needed to verify that $A|_t$ is unsatisfiable for each $t \in S$. It is natural that P is a subset of forbidden tuples since such tuples suffice to disprove satisfiability of a CSP.

▶ **Theorem 8.** *Let $A \subseteq T$. If an algorithm satisfying Property 7 takes A on input and returns sets S, P with $S \neq \emptyset$, then $d \in \mathbb{R}^T$ defined as*[13]

$$d_t = \begin{cases} -1 & \text{if } t \in S \\ |\{U \in \mathcal{U} \mid U \cap S \neq \emptyset\}| & \text{if } t \in P \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

is S-deactivating for A.

---

[10] Indeed, for any $A, S, U \subseteq T$ we have $(A - S) \cap U = \emptyset \iff A \cap U \subseteq S$. We will use this equivalence repeatedly in the sequel.

[11] Let us emphasise that this is different from the certificate of unsatisfiability of CSP $A^*(f)$ for f (in the sense of Theorem 4) because deactivating directions do not contain the step length. Following Footnote 8, the step length can be computed for any WCSP f with $A^*(f) = A$ using Theorem 15.

[12] This holds due to $A \subseteq T - P$ and the fact that if A' is unsatisfiable, then any $A \subseteq A'$ is unsatisfiable.

[13] $|\{U \in \mathcal{U} \mid U \cap S \neq \emptyset\}|$ is the number of scopes in $\mathcal{U}$ which contain at least one tuple from S. In other words, every assignment uses at most $|\{U \in \mathcal{U} \mid U \cap S \neq \emptyset\}|$ tuples from S.

Proof. Conditions (a) and (b) of Definition 5 are clearly satisfied and it only remains to show that $F(x|d) \geq 0 \; \forall x \in D^V$. We proceed by contradiction: let $x^* \in D^V$ such that $F(x^*|d) < 0$. Denote $m = |\{U \in \mathcal{U} \mid U \cap S \neq \emptyset\}|$. Necessarily, x^* uses at least one tuple $t^* \in S$ (otherwise, $F(x^*|d)$ would not be negative). Additionally, x^* does not use any tuple from P. The reason is that every x uses exactly one tuple from each set $U \in \mathcal{U}$, so it can use at most m tuples t with $d_t = -1$. If at least one tuple from P was used, we would have $F(x|d) \geq 0$ by (7).

Since x^* uses only tuples from the set $T - P$, it is a solution to CSP $T - P$. But x^* uses at least one tuple $t^* \in S$, i.e., $(T - P)|_{t^*}$ is satisfiable, which contradicts Property 7.   ◄

Note that Property 7 can be easily satisfied by any algorithm which prunes the CSP search space by forbidding tuples which are not used in any solution. Such tuples form the set S and set P can always be trivially chosen as $P = T - A$. Unfortunately, deactivating direction (7) calculated using $P = T - A$ would have many positive components. Consequently, an update of weights along such deactivating direction may substantially increase the values $F(x|f)$, which is undesirable as explained in §3.2.1.

Ideally, P should be as small as possible because then the deactivating directions do not increase the weights much and thus allow subsequent improvement of the bound. Though finding the smallest set P satisfying Property 7 is probably intractable[14], in practice we can often easily find a small such set. E.g., P can simply be the set of forbidden tuples which the algorithm needed to visit to make its decision. Alternatively, it may only be necessary to check the support of some tuple to forbid it. Importantly, 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.

We will now give examples of deactivating directions corresponding to well-known consistency conditions.

▶ **Example 9.** Let us consider *arc consistency* (AC). A CSP A is arc consistent if the equivalence[15] $(i, k) \in A \iff \bigvee_{l \in D}(\{(i, k), (j, l)\} \in A)$ holds for all $\{i, j\} \in E$, $k \in D$.

Let $k \in D$ and $\{i, j\} \in E$. If $(i, k) \in A$ and $\{(i, k), (j, l)\} \notin A$ for all $l \in D$, the AC propagator infers that $A|_{(i,k)}$ is unsatisfiable and forbids the tuple (i, k), that is, returns $S = \{(i, k)\}$. An S-deactivating direction d for A can be in this case simply

$$
d_t = \begin{cases} -1 & \text{if } t = (i, k) \\ 1 & \text{if } t \in \{\{(i, k), (j, l)\} \mid l \in D\} \\ 0 & \text{otherwise} \end{cases} \tag{8}
$$

because to forbid the tuple (i, k), it sufficed to verify that $\{(i, k), (j, l)\} \notin A$ for all $l \in D$. Thus, $P = \{\{(i, k), (j, l)\} \mid l \in D\}$.

For the other case, let $k \in D$ and $\{i, j\} \in E$. If $(i, k) \notin A$, AC propagator forbids tuples $S = \{\{(i, k), (j, l)\} \mid l \in D\} \cap A$ based on $P = \{(i, k)\}$. In this particular case, it is a good idea to choose an S-deactivating direction d for A as

---

[14] Set P is related (but not equivalent) to an *unsatisfiable core* of CSP $A|_t$. Finding a minimal unsatisfiable core is a "highly intractable problem" [15].

[15] Note, for convenience we use a slightly unusual definition of arc consistency, allowing to restrict not only domains but also constraint relations.

$$d_t = \begin{cases} -1 & \text{if } t \in \{\{(i,k),(j,l)\}) \mid l \in D\} \\ 1 & \text{if } t = (i,k) \\ 0 & \text{otherwise} \end{cases} . \tag{9}$$

Notice that all the binary tuples are set to -1 in (9), instead of just the tuples S as in (7). Although (7) would also provide an S-deactivating direction, it is better to use (9) because both directions d defined by (8) and (9) satisfy $F(x \mid d) = 0$ for all x. Thus, WCSPs g and $g + \alpha d$ are equivalent[16] for any $\alpha \in \mathbb{R}$ which is desirable (see §3.2.1). ⌟

▶ **Example 10.** We now consider *cycle consistency* as defined in [17].[17] Let $\mathcal{C}$ be a (polynomially sized) set of cycles in the graph (V, E). A CSP A is cycle consistent if for each tuple $(i, k) \in A \cap (V \times D)$ and each cycle $C \in \mathcal{C}$ that passes through node $i \in V$, there exists an assignment x with $x_i = k$ that uses only allowed tuples in cycle C. It can be shown that the cycle repair procedure in [17] in fact constructs a deactivating direction whenever an inconsistent cycle is found. Moreover, the constructed direction in this case coincides with (7) for a suitable set P which contains a subset of the forbidden tuples within the cycle. ⌟

▶ **Example 11.** Recall that a CSP A is *singleton arc consistent (SAC)* if for every tuple $t = (i, k) \in A \cap (V \times D)$, the CSP $A|_t$ has a non-empty arc-consistency closure. Good (i.e., sparse) deactivating directions for SAC can be obtained as follows. For some $(i, k) \in A \cap (V \times D)$, we enforce arc consistency of CSP $A|_{(i,k)}$, during which we store the causes for forbidding each tuple. If $A|_{(i,k)}$ is found to have an 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 Theorem 8 with $S = \{(i, k)\}$. Note that SAC does not have bounded support as many other kinds of local consistencies [3], so the size of P can be significantly different for different CSP instances. ⌟

4.2 Composing Deactivating Directions

Recall that constraint propagation iteratively forbids some tuples of a given CSP $A \subseteq T$, until it is no longer able to forbid any tuple or it becomes explicit that the CSP is unsatisfiable. The latter happens if all tuples of some set $U \in \mathcal{U}$ become forbidden[18] (i.e., $U \cap A = \emptyset$), because (as noted in §2) every assignment uses exactly one tuple from every set from $\mathcal{U}$.

Formally, consider a propagation rule to enforce a local consistency condition Φ, such that if CSP A is not Φ-consistent then it finds a non-empty set $S \subseteq A$ of tuples to forbid and an S-deactivating direction[19] for A. This rule is applied to the given CSP iteratively, each time forbidding a different set of tuples. This is outlined in Algorithm 1, which stores the generated sets S_r of tuples being forbidden and the corresponding S_r-deactivating directions d^r. Note that, by line 4 of the algorithm, $A_r = A - \bigcup_{q=0}^{r-1} S_q$ for every $r = 0, \ldots, s+1$.

The generated sequence of S_r-deactivating directions d^r for A_r can be composed into a single $\left(\bigcup_{q=0}^{s} S_q\right)$-deactivating direction for A using the following composition rule (the proof is given in the appendix):

---

[16] Such reparametrizations correspond to soft arc consistency operations *extend* and *project* in [5].

[17] This is different from *cyclic consistency* as defined in [4]. E.g., reparametrizations are sufficient to enforce cyclic consistency, whereas super-reparametrizations are needed for cycle consistency.

[18] If $U = \{(i, k) \mid k \in D\}$ for some $i \in V$, this is often called "domain wipe-out".

[19] The deactivating direction can be constructed in any way, e.g. (but not necessarily) using Theorem 8.

■ **Algorithm 1** Propagation phase: given a CSP $A \subseteq T$, propagation is applied to A while deactivating directions are stored.

---

1 Initialize $s := 0$, $A_0 := A$
2 **while** A_s *is not* Φ-*consistent* **do**
3 Find set $S_s \subseteq A_s$ and an S_s-deactivating direction d^s for A_s.
4 $A_{s+1} := A_s - S_s$
5 **if** $\exists U \in \mathcal{U} : U \cap A_{s+1} = \emptyset$ **then**
6   **return** *unsatisfiable*, $(A_r)_{r=0}^{s+1}$, $(S_r)_{r=0}^{s}$, $(d^r)_{r=0}^{s}$
7 $s := s + 1$
8 **return** *possibly satisfiable*

■ **Algorithm 2** Composition phase: the sequences $(S_r)_{r=0}^{s}$ and $(d^r)_{r=0}^{s}$ generated by Algorithm 1 for given $R \subseteq \{0, ..., s\}$, $R \neq \emptyset$ are composed to an M-deactivating direction d' for A.

---

1 Initialize $r := \max R$, $d' := d^r$, $M := S_r$.
2 **while** $r > 0$ **do**
3 $r := r - 1$
4 **if** $r \in R$ *or* $\exists t \in S_r : d'_t \neq 0$ **then**
5 $d' := d' + \delta d^r$ (where δ is given by (10) where d, S are replaced by d^r, S_r)
6 $M := M \cup S_r$
7 **return** d', M

▶ **Proposition 12.** *Let $A \subseteq T$ and $S, S' \subseteq A$ where $S \cap S' = \emptyset$. Let d be an S-deactivating direction for A. Let d' be an S'-deactivating direction for $A - S$. Let*

$$\delta = \begin{cases} 0 & \text{if } d'_t \leq -1 \text{ for all } t \in S, \\ \max\{ (-1 - d'_t)/d_t \mid t \in S, \ d'_t > -1 \} & \text{otherwise.} \end{cases} \tag{10}$$

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

Proposition 12 allows us to combine S_r-deactivating direction d^r for $A_r = A_{r-1} - S_{r-1}$ with S_{r-1}-deactivating direction d^{r-1} for A_{r-1} into a single $(S_{r-1} \cup S_r)$-deactivating direction for A_{r-1}. By induction, we are able to gradually build a $\left(\bigcup_{q=0}^{s} S_q\right)$-deactivating direction for A, which certifies unsatisfiability of A whenever Algorithm 1 returns "unsatisfiable".

However, it is not always needed to construct a full $\left(\bigcup_{q=0}^{s} S_q\right)$-deactivating direction as not every step of the propagator is necessary to prove unsatisfiability. Instead, one can choose any $U \in \mathcal{U}$ such that $U \cap A_{s+1} = \emptyset$ (equivalent to $U \cap \left(A - \bigcup_{q=0}^{s} S_q\right) = \emptyset$, i.e, $U \cap A \subseteq \bigcup_{q=0}^{s} S_q$) and construct an M-deactivating direction for a (possibly smaller) set $M \subseteq \bigcup_{q=0}^{s} S_q$, so that $U \cap A \subseteq M$. Such direction still certifies unsatisfiability of A and can be sparser than a $\left(\bigcup_{q=0}^{s} S_q\right)$-deactivating direction, which is desirable as explained in §3.2.1.

This is outlined in Algorithm 2, which composes only a subsequence of directions given by the set $R \subseteq \{0, ..., s\}$ and constructs an M-deactivating direction where $M \supseteq \bigcup_{r \in R} S_r$. Although Algorithm 2 is applicable for any set R, in our case R is obtained by first choosing any $U \in \mathcal{U}$ such that $U \cap (A - \bigcup_{q=0}^{s} S_q) = \emptyset$ and then setting $R = \{r \in \{0, ..., s\} \mid S_r \cap U \neq \emptyset\}$, so that $U \cap (A - M) = \emptyset$ due to $U \cap A \subseteq \bigcup_{r \in R} S_r \subseteq M$. Correctness of Algorithm 2 is given by the following theorem, whose proof is given in the appendix.

▶ **Theorem 13.** *Algorithm 2 returns an M-deactivating direction d' for A with $M \supseteq \bigcup_{r \in R} S_r$.*

▶ Remark 14. This is similar to what the VAC [5] or Augmenting DAG algorithm [18, 32] 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 AC closure of $A^*(f)$ is found empty (which corresponds to $U \cap A_{s+1} = \emptyset$ for some $U \in \mathcal{U}$), 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 U).   ⌟

5 Line Search and the Final Algorithm

In §4 we showed how to construct an S-deactivating direction $d \in \mathbb{R}^T$ for a CSP A, which certifies unsatisfiability of A whenever $A \cap U \subseteq S$ (i.e., $(A - S) \cap U = \emptyset$) for some $U \in \mathcal{U}$. For a WCSP $f \in \mathbb{R}^T$, to obtain a certificate $h \in \mathbb{R}^T$ of unsatisfiability of CSP $A^*(f)$ for f in the sense of Theorem 4, we need a step length $\alpha > 0$ so that $h = \alpha d$, as mentioned in §3.2.1. The step length is obtained using the following (somewhat more general) result, whose proof is given in the appendix.

▶ **Theorem 15.** *Let $f \in \mathbb{R}^T$. Let d be an S-deactivating direction for $A^*(f)$. Denote*[20]

$$\beta = \min\{\,(\max_{t \in U(t')} f_t - f_{t'})/d_{t'} \mid t' \in T,\ d_{t'} > 0\,\},$$
$$\gamma = \min\{\,(f_t - f_{t'})/(d_{t'} - d_t) \mid U \in \mathcal{U},\ A^*(f) \cap U \subseteq S,\ t \in U \cap S,\ t' \in U - S,\ d_{t'} > d_t\,\}.$$

Then $\beta, \gamma > 0$ and for every $U \in \mathcal{U}$ and $\alpha \in \mathbb{R}$, WCSP $f' = f + \alpha d$ satisfies:
(a) If $A^*(f) \cap U \not\subseteq S$ and $0 \le \alpha \le \beta$, then $\max_{t \in U} f'_t = \max_{t \in U} f_t$.
(b) If $A^*(f) \cap U \subseteq S$ and $0 < \alpha \le \min\{\beta, \gamma\}$, then $\max_{t \in U} f'_t < \max_{t \in U} f_t$.
(c) If $A^*(f) \cap U \not\subseteq S$ and $0 < \alpha < \beta$, then $A^*(f') \cap U = (A^*(f) - S) \cap U$.

If d is an S-deactivating direction for CSP $A^*(f)$ and for all $U \in \mathcal{U}$ we have $A^*(f) \cap U \not\subseteq S$ then, by Theorem 15(a,c), there is $\alpha > 0$ such that $f' = f + \alpha d$ satisfies $B(f') = B(f)$ and $A^*(f') = A^*(f) - S$. This justifies why such direction d is called S-deactivating: a suitable update of f along this direction makes tuples S inactive for f.

▶ Remark 16. This might suggest that to improve the current bound $B(f)$, we need not use Algorithm 2 to construct an S'-deactivating direction d' such that $A^*(f) \cap U \subseteq S'$ for some $U \in \mathcal{U}$, but instead perform steps using the intermediate S_r-deactivating directions d^r to create a sequence $f^{r+1} = f^r + \alpha_r d^r$ satisfying $B(f^0) = B(f^1) = \cdots = B(f^s) > B(f^{s+1})$. Unfortunately, it is hard to make this work reliably as there are many choices for the intermediate step sizes $0 < \alpha_r < \beta_r$. We empirically found Algorithm 3 to be preferable.   ⌟

If d is an S-deactivating direction for $A^*(f)$ and for some $U \in \mathcal{U}$ we have $A^*(f) \cap U \subseteq S$, then, by Theorem 15(a,b), there is $\alpha > 0$ such that $f' = f + \alpha d$ satisfies $B(f') < B(f)$. Thus, $h = \alpha d$ is a certificate of unsatisfiability of $A^*(f)$ for f in the sense of Theorem 4.

---

[20] β is always defined: by Definition 5 we have $F(x \mid d) \ge 0$ for all x, hence $\exists t\colon d_t < 0 \Rightarrow \exists t'\colon d_{t'} > 0$.
   γ is defined and needed only in (b), where we assume that $U \cap A^*(f) \subseteq S$ for some $U \in \mathcal{U}$.

■ **Algorithm 3** The final algorithm to iteratively improve feasible solutions to (5) (i.e., an upper bound on the optimal value of WCSP g).

---

1 Initialize $f := g$.
2 **while** *Algorithm 1 returns "unsatisfiable" on $A^*(f)$* **do**
3   Generate sequences $(A_r)_{r=0}^{s+1}$, $(S_r)_{r=0}^{s}$, $(d^r)_{r=0}^{s}$ by Algorithm 1.
4   Let $U \in \mathcal{U} : U \cap A_{s+1} = \emptyset$ and set $R := \{r \in \{0, ..., s\} \mid S_r \cap U \neq \emptyset\}$.
5   Compute M-deactivating direction d' using Algorithm 2.
6   Update $f := f + \min\{\beta, \gamma\}d'$ following Theorem 15.
7 **return** $B(f)$

Theorem 15 also proposes a possible (not necessarily optimal[21]) step size α. This allows us to formulate, in Algorithm 3, the iterative scheme outlined in §3.2.1. 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, we choose $U \in \mathcal{U}$ such that $U \cap A_{s+1} = \emptyset$, i.e., $A^*(f) \cap U \subseteq \bigcup_{r=0}^{s} S_r$ (which exists since Algorithm 1 returned "unsatisfiable"), define R so that $U \cap A^*(f) \subseteq \bigcup_{r \in R} S_r$, and compute an M-deactivating direction d' where $M \supseteq \bigcup_{r \in R} S_r$ using Theorem 13. Since $A^*(f) \cap U \subseteq M$, we can update WCSP f using Theorem 15. Consequently, the bound $B(f)$ strictly improves after each update on line 6.

In Algorithm 3 we additionally used a heuristic analogous to *capacity scaling* in network flow algorithms. We replace the active tuples $A^*(f)$ with "almost" active (θ-active) tuples $A_\theta^*(f) = \{t \in T \mid f_t \geq \max_{t' \in U(t)} f_{t'} - \theta\}$ for some threshold $\theta > 0$.[22] This forces the algorithm to disprove satisfiability using tuples which are far from 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.[23]

6 Experiments

We implemented two versions of Algorithm 3 (incl. capacity scaling), differing in the local consistency used to attempt to disprove satisfiability of CSP $A^*(f)$:

- *Virtual singleton arc consistency via super-reparametrizations (VSAC-SR)* [24] uses singleton arc consistency. Precisely, we alternate between AC and SAC propagators: whenever a tuple (i, k) is removed by SAC, we step back to enforcing AC until no more AC propagations are possible, and repeat.

---

[21] Finding an optimal 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 instances, we find only a sub-optimal step size: we find the first break (i.e., non-differentiable) point of the function with a lower objective. This step size is decreased to β if $\gamma > \beta$ so that no maximum increases. This is analogous to the *first-hit strategy* in [11, §3.1.4].

[22] This is similar to the notion of $\text{Bool}_\theta(f)$ in [5, §11.1], tolerance δ in [12, §4.2], and $\text{mi}_\epsilon[f]$ in [20, §6.2.4].

[23] Precisely, we initialized $\theta = \max_{k,l} g_{ij}(k, l) - \min_{k,l} g_{ij}(k, l) + \max_k g_{i'}(k) - \min_k g_{i'}(k)$ where $\{i, j\} \in E$ and $i' \in V$ is the edge and variable with the lowest index (based on indexing in the input instance). The terminating condition was $\theta \leq 10^{-6}$.

[24] In analogy to [5, 19], let us call a WCSP instance f *virtual X-consistent* (e.g., virtual AC or virtual RPC) if $A^*(f)$ has a non-empty X-consistency closure. Then, a *virtual X-consistency algorithm* naturally refers to an algorithm to transform a given WCSP instance to a virtual X-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 algorithm VSAC-SR.

■ *Virtual cycle consistency via super-reparametrizations (VCC-SR)* is the same as VSAC-SR except that SAC is replaced by CC.[25] Though our implementation is different than [17] (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 9, 11, and 10. In SAC and CC it is useful to step back to AC whenever possible since, as described in §4.1, deactivating directions of AC correspond to reparametrizations and thus avoid increasing the values of individual assignments, which is beneficial as discussed in §3.2.1.

We compared the bounds calculated by VSAC-SR and VCC-SR with the bounds provided by EDAC [9], VAC [5], pseudo-triangles (option `-t=8000` in toulbar2, adds up to 8 GB of ternary weight functions), PIC, EDPIC, maxRPC, and EDmaxRPC [19] which are implemented in toulbar2 [26].

We did the comparison on the Cost Function Library benchmark [8]. 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 options `-A -bt=0` found an optimal solution). 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 resp. B_b is the worst resp. best bound for the instance among all the methods. Thus, the best bound[26] 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 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.[27]

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 not much worse than the other strong consistency methods. VSAC-SR seems particularly good at spinglass_maxcut [25], planning [7] and qplib [13] 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).

---

[25] We chose the cycles in VCC-SR as follows: if $2|E|/|V| \leq 5$ (i.e, average degree of the nodes 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). No additional edges are added to the graph. Note, [17] experimented with grid graphs (where 4-cycles and 6-cycles of the grid were used) and complete graphs (where 3-cycles were used).

[26] 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.

[27] 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 5731 instances in total.

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 are faster but need significantly more memory.

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). Besides this, we did no more attempts to fine-tune our implementation for efficiency. Thus, the set $A^*(f)$ was always calculated by iterating through all tuples. SAC was checked on all active tuples without warm-starting or using any faster SAC algorithm than SAC1 [2, 10]. Perhaps most importantly, we did not implement inter-iteration warm-starting as in [31, 11], i.e., after updating the weights on line 6 of Algorithm 3, some deactivating directions in the sequence which 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 vectors, the code was the same for VSAC-SR and VCC-SR. We implemented everything in Java.

7   Concluding Remarks

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).

Special cases of our approach are the VAC / Augmenting DAG algorithm [5, 18, 32] which uses arc consistency and the algorithm in [17] which uses cycle consistency. We have newly 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.

The approach can be straightforwardly extended to WCSPs with different domain sizes, weight functions of any arities, and some weights equal to minus infinity (i.e., some constraints being hard). Note in particular that SAC is not restricted to binary CSPs. Of course, further experiments would be needed to evaluate the quality of the bounds in this case.

Our approach in general requires to store 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 to prune the search space during branch-and-bound search, when solving WCSP instances to optimality. However, we have done no experiments with this, so it is open whether during search the tighter bounds would outweigh the higher complexity of the algorithm. We leave this for the future research. 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.

■ **Table 1** Results on instances from Cost Function Library: Average normalized bounds.

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
/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 ≥ 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.

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
/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

References

1 Dhruv Batra, Sebastian Nowozin, and Pushmeet Kohli. 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*, pages 146–154, 2011.

2 Christian Bessiere, Stephane Cardon, Romuald Debruyne, and Christophe Lecoutre. Efficient algorithms for singleton arc consistency. *Constraints*, 16(1):25–53, 2011.

3 Christian Bessiere and Romuald Debruyne. Theoretical analysis of singleton arc consistency. In *Workshop on Modelling and Solving Problems with Constraints*, pages 20–29, 2004.

4 Martin C. Cooper. Cyclic consistency: a local reduction operation for binary valued constraints. *Artificial Intelligence*, 155(1-2):69–92, 2004.

5 Martin C. Cooper, Simon de Givry, Martí Sanchez, Thomas Schiex, Matthias Zytnicki, and Tomáš Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478, 2010.

6 Martin C. Cooper, Simon de Givry, and Thomas Schiex. Optimal soft arc consistency. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, volume 7, pages 68–73, 2007.

7 Martin C. Cooper, Marie de Roquemaurel, and Pierre Régnier. A weighted CSP approach to cost-optimal planning. *AI Communications*, 24(1):1–29, 2011.

8 `https://forgemia.inra.fr/thomas.schiex/cost-function-library`, commit `356bbb85`.

9 Simon De Givry, Federico Heras, Matthias Zytnicki, and Javier Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *IJCAI*, volume 5, pages 84–89, 2005.

10 Romuald Debruyne and Christian Bessiere. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97*, pages 412–417, 1997.

11 Tomáš Dlask. Minimizing convex piecewise-affine functions by local consistency techniques. *Master's Thesis*, 2018.

12 Tomáš Dlask and Tomáš Werner. Bounding linear programs by constraint propagation: Application to Max-SAT. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 2020.

13 Fabio Furini, Emiliano Traversi, Pietro Belotti, Antonio Frangioni, Ambros Gleixner, Nick Gould, Leo Liberti, Andrea Lodi, Ruth Misener, Hans Mittelmann, et al. QPLIB: a library of quadratic programming instances. *Mathematical Programming Computation*, 11(2):237–265, 2019.

14 Amir Globerson and Tommi S Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in Neural Information Processing Systems*, pages 553–560, 2008.

15 Eric Grégoire, Bertrand Mazure, and Cédric Piette. On finding minimally unsatisfiable cores of CSPs. *International Journal on Artificial Intelligence Tools*, 17(04):745–763, 2008.

16 Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006.

17 Nikos Komodakis and Nikos Paragios. Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles. In *European conference on computer vision*, pages 806–820. Springer, 2008.

18 V. K. Koval and M. I. Schlesinger. Dvumernoe programmirovanie v zadachakh analiza izobrazheniy (Two-dimensional programming in image analysis problems). *Automatics and Telemechanics*, 8:149–168, 1976. In Russian.

19 Hiep Nguyen, Christian Bessiere, Simon de Givry, and Thomas Schiex. Triangle-based consistencies for cost function networks. *Constraints*, 22(2):230–264, 2017.

20 Bogdan Savchynskyy. Discrete graphical models – an optimization perspective. *Foundations and Trends in Computer Graphics and Vision*, 11(3-4):160–429, 2019.

21 M. Schlesinger. Sintaksicheskiy analiz dvumernykh zritelnikh signalov v usloviyakh pomekh (syntactic analysis of two-dimensional visual signals in noisy conditions). *Kibernetika*, 4(113-130):2, 1976.

22 H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal of Discrete Mathematics*, 3(3):411–430, 1990.

23 David Sontag and Tommi Jaakkola. Tree block coordinate descent for MAP in graphical models. In *Artificial Intelligence and Statistics*, pages 544–551, 2009.

24 David Sontag, Talya Meltzer, Amir Globerson, Tommi Jaakkola, and Yair Weiss. Tightening LP relaxations for MAP using message passing, 2008.

25 `https://software.cs.uni-koeln.de/spinglass`.

26 `https://miat.inrae.fr/toulbar2`.

27 Siddharth Tourani, Alexander Shekhovtsov, Carsten Rother, and Bogdan Savchynskyy. MPLP++: Fast, parallel dual block-coordinate ascent for dense graphical models. In *Proceedings of the European Conference on Computer Vision*, pages 251–267, 2018.

28 Siddharth Tourani, Alexander Shekhovtsov, Carsten Rother, and Bogdan Savchynskyy. Taxonomy of dual block-coordinate ascent methods for discrete energy minimization. In *International Conference on Artificial Intelligence and Statistics*, pages 2775–2785. PMLR, 2020.

29 Stanislav Živný. *The Complexity of Valued Constraint Satisfaction Problems*. Cognitive Technologies. Springer, 2012.

30 Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

31 Tomáš Werner. A linear programming approach to max-sum problem: A review. Technical Report CTU-CMP-2005-25, Center for Machine Perception, Czech Technical University, December 2005.

32 Tomáš Werner. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, July 2007.

33 Tomáš Werner. 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, August 2010.

34 Tomáš Werner. Marginal consistency: Upper-bounding partition functions over commutative semirings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1455–1468, July 2015.

A Appendix
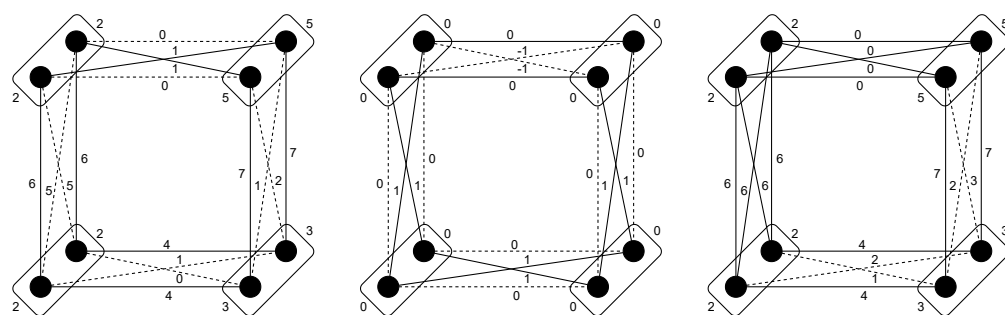
(a) WCSP f, $A^*(f)$ unsatisfiable. **(b)** Certificate h of unsatisfiability **(c)** WCSP $f+h$, $B(f+h) < B(f)$.
of $A^*(f)$ for f.

■ **Figure 1** Example of a single iteration of the scheme. Variables (elements of V) are depicted as rounded rectangles, tuples (elements of T) as circles and line segments, and weights f_t are written next to the circles and line segments. Black circles and solid lines depict active tuples, dashed lines depict inactive tuples.

Proof of Proposition 12. First, if $d'_t \leq -1$ for all $t \in S$, then $d'' = d'$ satisfies the required condition immediately. Otherwise, $\delta > 0$ since $d_t < 0$ for all $t \in S$ by definition and $-1 - d'_t < 0$ due to $d'_t > -1$ in definition of δ. We will show that d'' satisfies the conditions in Definition 5.

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

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

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

To show (c): for any $x \in D^V$, $F(x|d'') = F(x|d') + \delta F(x|d) \geq 0$ by $\delta \geq 0$. ◄

Proof of Theorem 13. The fact that $M \supseteq \bigcup_{r \in R} S_r$ is obvious due to $S_{\max R} \subseteq M$ by initialization on line 1 and $S_r \subseteq M$ for any $r \in R$, $r < \max R$ because in such case the update on line 6 is performed.

It remains to show that d' is M-deactivating, which we will do by induction. We claim that vector d' is always M-deactivating direction for A_r on line 2 and M-deactivating direction for A_{r+1} on line 4.

Initially we have $d' = d^r$, so d' is S_r-deactivating (i.e., M-deactivating since $M = S_r$ before the loop is entered) for A_r. Also, when vector d' is first queried on line 4, r decreased by 1 due to update on line 3, so d' is M-deactivating for A_{r+1}. The required property thus holds when the condition on line 4 is first queried with $r = \max R - 1$.

We proceed with the inductive step. If the condition on line 4 is not satisfied, then necessarily $d'_t = 0$ for all $t \in S_r$. So, if d' is M-deactivating for A_{r+1}, then it is also M-deactivating for $A_r = A_{r+1} \cup S_r$, as can be seen from Definition 5.

If the condition on line 4 is satisfied, d' is M-deactivating for A_{r+1} before the update on lines 5-6. Since $A_{r+1} = A_r - S_r$ and d^r is S_r-deactivating for A_r, Proposition 12 can be applied to d^r and d' to obtain an $(M \cup S_r)$-deactivating direction for A_r. After updating M on line 6, it becomes M-deactivating for A_r.

Eventually, when $r = 0$, d' is M-deactivating for $A_0 = A$ by line 1 in Algorithm 1. ◄

Proof of Theorem 15. We have $\beta > 0$ because $d_{t'} > 0$ implies t' is an inactive tuple, so $\max_{t \in U(t')} f_t > f_{t'}$. We have $\gamma > 0$ because in $f_t - f_{t'}$ tuple t is always active and t' is inactive, hence $f_t > f_{t'}$.

To prove (a), let $A^*(f) \cap U \nsubseteq S$, so there is $t^* \in U$ such that $t^* \in A^*(f)$ and $t^* \notin S$. Hence, by Definition 5, $d_{t^*} = 0$ and value $\max_{t \in U} f'_t$ does not decrease for any α since $f'_{t^*} = f_{t^*} + \alpha d_{t^*} = f_{t^*}$. To show that the maximum does not increase, consider a tuple $t' \in U$ such that $d_{t'} > 0$ (due to $\alpha \geq 0$, tuples with $d_{t'} \leq 0$ can not increase the maximum). It follows that $\alpha \leq \beta \leq (\max_{t \in U} f_t - f_{t'})/d_{t'}$, so $f'_{t'} = f_{t'} + d_{t'} \alpha \leq \max_{t \in U} f_t$.

To prove (b), let $A^*(f) \cap U \subseteq S$. For all $t \in U \cap S$, we have $f'_t = f_t + \alpha d_t < f_t$ by $d_t < 0$ and $\alpha > 0$, i.e., $\max_{t \in U \cap S} f'_t < \max_{t \in U \cap S} f_t$. We proceed to show that $f'_t \leq \max_{t' \in U \cap S} f'_{t'}$ for every $t' \in U - S$. Let $t^* \in U \cap S$ satisfy $f'_{t^*} = \max_{t \in U \cap S} f'_t$. If $d_{t'} > d_{t^*}$, $\alpha \leq \gamma \leq (f_{t^*} - f_{t'})/(d_{t'} - d_{t^*})$ 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, $\max_{t' \in U - S} f'_{t'} \leq \max_{t \in U \cap S} f'_t < \max_{t \in U \cap S} f_t = \max_{t \in U} f_t$.

To prove (c), let $A^*(f) \cap U \nsubseteq S$. Following (a), we have $\max_{t \in U} f_t = \max_{t \in U} f'_t$. If $t \in (A^*(f) - S) \cap U$, then $d_t = 0$ and such tuples remain active by $f'_t = f_t$. Tuples $t \in S \cap U$ become inactive since $f'_t = f_t + d_t \alpha < f_t = \max_{t' \in U} f_{t'}$ by $d_t < 0$ and $\alpha > 0$. Tuples $t \notin A^*(f)$ either satisfy $d_t \leq 0$ and can not become active or satisfy $d_t > 0$ and by $\alpha < \beta \leq (\max_{t' \in U} f_{t'} - f_t)/d_t$, $f'_t = f_t + d_t \alpha < \max_{t' \in U} f_{t'}$, so $t \notin A^*(f')$. ◄