Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Block-Coordinate Descent and Local Consistencies in Linear Programming

Dissertation Thesis

Ing. Tomáš Dlask

Supervisor: doc. Ing. Tomáš Werner, PhD.

Study Programme: Electrical Engineering and Information Technology
Branch of Study: Artificial Intelligence and Biocybernetics

Prague, May 2022

Abstract

Even though linear programming (LP) problems can be solved in polynomial time, solving large-scale LP instances using off-the-shelf solvers may be difficult in practice, which creates demand for specialized scalable methods. One such method for large-scale problems is block-coordinate descent (BCD). However, the fixed points of this method need not be global optima even for convex optimization problems. Despite this limitation, various BCD algorithms (also called `convergent message-passing algorithms') are successfully used for approximately solving the dual LP relaxation of the weighted constraint satisfaction problem (WCSP, also known as MAP inference in graphical models) and their fixed points can be characterized using local consistencies, typically variants of arc consistency.

In this work, we focus on optimizing linear programs by BCD or constraint propagation and theoretically relating these approaches. To this end, we propose a general constraint-propagation-based framework for approximate optimization of large-scale linear programs whose applicability is evaluated on publicly available benchmarks. In detail, we employ this approach to approximately optimize the dual LP relaxation of weighted Max-SAT and an LP formulation of WCSP. In the latter case, we show that one can use any classical CSP constraint propagation method in order to obtain an upper bound on the optimal value. This is in contrast to existing methods that needed to be tailored to a specific chosen kind of local consistency. However, the cost for this is that our approach may not preserve the properties of the input WCSP instance, such as the set of optimal assignments, and only provides an upper bound on its optimal value, which is nevertheless important for pruning the search space during branch-and-bound search.

Although one can use our general framework with any constraint propagation method in a system of linear inequalities, we identify the precise form of constraint propagation such that the stopping points of the resulting algorithm coincide with the fixed points of BCD. In other words, we identify the kind of local consistency that is enforced by BCD in any linear program. Depending on the problem being solved, this condition may be interpreted, e.g., as arc consistency or positive consistency. Thanks to these results, we characterize linear programs that are optimally solvable by BCD by refutation-completeness of the associated propagator (i.e., whether it can always detect infeasibility of a certain class of systems of linear inequalities and equalities). This allows us to identify new classes of linear programs exactly solvable by BCD, including, e.g., an LP formulation of the maximum flow problem or LP relaxations of some combinatorial problems.

We believe that this work may initiate further research on large-scale non-smooth constrained convex optimization problems.

Keywords: Linear Programming, Block-Coordinate Descent, Local Consistency, Constraint Propagation, Weighted Constraint Satisfaction Problem

Abstrakt

Prestoze problemy linearn ho programovan (LP) lze resit v polynomialn m case, bezne metody nemus byt v praxi dostatecne pro problemy velkeho rozsahu, coz vytvar poptavku po specializovanych skalovatelnych metodach. Takovou metodou pro rozsahle problemy je sestup po bloc ch souradnic (BCD), jej z xn body ovsem nemus byt globaln optima ani pro konvexn optimalizacn problemy. Navzdory tomuto omezen jsou ruzne BCD algoritmy uspesne pouz vany pro priblizne resen dualn LP relaxace problemu s vazenymi omezen mi (WCSP, take znameho jako MAP inference v grafovych modelech) a jejich xn body mohou byt charakterizovany pomoc lokaln ch konzistenc , typicky variant hranove konzistence.

V teto praci se zamerujeme na optimalizaci linearn ch programu pomoc BCD nebo propagace podm nek a teoreticky oba tyto pr stupy propojujeme. Abychom dosahli tohoto c le, navrhneme obecne schema zalozene na propagaci podm nek pro pribliznou optimalizaci linearn ch programu velkeho rozsahu, jehoz pouzitelnost vyhodnocujeme na verejne dostupnych instanc ch. Konkretne tento pr stup aplikujeme k priblizne optimalizaci dualn LP relaxace vazeneho Max-SAT problemu a LP formulace WCSP. V pr pade LP formulace WCSP ukazeme, ze k z skan horn meze na optimaln hodnotu je mozne pouz t jakoukoli klasickou metodu propagace podm nek v CSP, coz se lis od stavaj c ch metod, ktere bylo nutno prizpusobit konkretn mu zvolenemu druhu lokaln konzistence. Nevyhodou naseho pr stupu je, ze nemus zachovavat vlastnosti vstupn WCSP instance, jako napr klad mnozinu optimaln ch resen , ale poskytuje pouze horn mez na optimaln hodnotu, ktera je nicmene dulezita pro prorezavan prohledavaneho prostoru behem metody vetv a mez .

Ackoli je mozne pouz t nase obecne schema s jakoukoli metodou propagace podm nek v soustave linearn ch nerovnost , nasli jsme takovy konkretn zpusob propagace, ze xn body vysledneho algoritmu se shoduj s xn mi body BCD. Jinymi slovy, nasli jsme druh lokaln konzistence, ktery BCD vynucuje v jakemkoli linearn m programu. V zavislosti na resenem problemu se tato lokaln konzistence muze interpretovat napr klad jako hranova konzistence nebo pozitivn konzistence. D ky temto vysledkum charakterizujeme linearn programy, ktere jsou optimalne resitelne pomoc BCD, pomoc uplnosti odpov daj c metody propagace podm nek (tj. jestli dana metoda vzdy dokaze detekovat nesplnitelnost jiste tr dy linearn ch rovnic a nerovnic). Tyto vysledky umoznuj identi kovat nove tr dy linearn ch programu, ktere lze optimalne resit pomoc BCD. Takove linearn programy zahrnuj napr klad LP formulaci problemu hledan maximaln ho toku v s ti nebo LP relaxace urcitych kombinatorickych problemu.

Ver me, ze tato prace muze podn tit dals vyzkum v oblasti konvexn ch nehladkych optimalizacn ch problemu velkeho rozsahu s omezen mi.

Kl cova slova: Linearn programovan , Sestup po bloc ch souradnic, Lokaln konzistence, Propagace podm nek, Vazene CSP

Preklad nazvu: Sestup po bloc ch souradnic a lokaln konzistence v linearn m programovan

Acknowledgement

I am thankful to my supervisor Tomas Werner for his guidance, frequent discussions, collaboration, and useful advice during my studies. I would also like to thank Daniel Prusa and Simon de Givry for collaboration on joint papers. Furthermore, I appreciate the possibility to discuss ongoing research with other members of the Machine Learning group at the department. Last but not least, I wish to express my sincere gratitude to my family and friends for their support.

The support of the Grant Agency of the Czech Technical University in Prague (grants SGS19/170/OHK3/3T/13 and SGS22/061/OHK3/1T/13), the Czech Science Foundation (grant 19-09967S), and OP VVV project CZ.02.1.01/0.0/0.0/16_019/0000765 is gratefully acknowledged.

Contents

Appendix

Introduction

Optimization is nowadays ubiquitous in machine learning, computer vision, and artificial intelligence in general. Optimization problems also frequently emerge in industrial applications and, in the era of big data, may feature a very large number of variables and constraints. Even if the constraints are sparse, using classical off-the-shelf solvers is usually not suitable in practice which results in demand for specialized algorithms that could tackle such large-scale problems by utilizing their (usually regular) structure and sparsity.

To illustrate this, despite linear programming (LP) problems are solvable in polynomial time, even verifying feasibility of LP relaxations of some hard combinatorial problems may take significant amount of time, maybe even exceed the overall time limit for solving the original (non-relaxed) problem [47, §3.2]. As another example, LP instances originating in computer vision may have millions of constraints and variables [154, 119, Example 4.2]. Off-the-shelf LP solvers typically cannot be applied to such large instances [154, 129, 128, 73, 114] due to their super-linear time and/or space complexity. Since LP relaxations of many classical NP-hard problems are as hard to solve as any linear program [114], designing more efficient exact methods for these relaxations may result in improving upon the best known general-purpose LP solvers, which is unlikely.

One of scalable methods is block-coordinate descent (BCD, a.k.a. block-coordinate minimization). This is an iterative method for (approximate [1]) optimization of a multivariate function which, in each iteration, chooses a subset (also called a block) of variables and optimally solves the problem over this subset of variables while keeping the other variables constant. By repeating this iteration for different blocks, the method can eventually converge to a `local' minimum (understood w.r.t. block-coordinate moves) which is optimal w.r.t. all blocks of variables, or even a global minimum. In the simplest setting, the blocks correspond to single variables, which is usually called coordinate-wise minimization. In this case, the optimization subproblems are univariate and can be easily solved, sometimes even in closed-form, which results in simple and efficient algorithms.

BCD has been successfully applied to a number of optimization problems, such as support vector machine training [77, 112], non-negative least squares [60], non-negative matrix factorization [78], regression [152, 63], or semidefinite programs with diagonal constraints [143, 144].

However, except for special cases [155, 15§2.7, 12, 137], BCD may not even converge to the set of global minima for general constrained or non-differentiable convex optimization problems and the BCD local minima may be arbitrarily far from global minima. Despite this fundamental limitation, BCD (a.k.a. convergent message passing in this context) is a powerful heuristic to approximately optimize unconstrained convex piecewise-affine (hence non-differentiable) functions emerging as various forms of the dual LP relaxation of the weighted constraint satisfaction problem (WCSP) [65, 134, 88, 87, 135]. These methods have been generalized and applied to LP relaxations of other large-scale combinatorial problems, such as minimum cost multicut problem [128] or graph matching [130], within

[1] Throughout the thesis, whenever we write `approximately optimize', we mean `attempt to find some (hopefully good) solution that can be however arbitrarily bad in theory', i.e., there are no formal guarantees on its quality.

a general framework of BCD applied to a Lagrange dual decomposition of combinatorial problems [129]. Other related approaches are [73] and [97].

Let us now return to the WCSP. The WCSP is a combinatorial NP-hard optimization problem where the task is to maximize a function of many discrete variables that is expressed as the sum of weight functions where each weight function depends only on a (small) subset of the variables. Its dual LP relaxation can be interpreted as minimizing an upper bound on the optimal value of a WCSP over a subset of its reparametrizations (i.e., WCSPs with the same objective value for all assignments) and the fixed points of various BCD methods applied to this linear program can be characterized by certain local consistencies. On the other hand, there are also methods that (approximately) optimize this dual LP relaxation by directly enforcing (possibly soft) local consistencies [43, 98, 33, 107, 95] and, in case of [33, 95], attain fixed points of the same nature as the BCD algorithms mentioned above. Although these algorithms may not solve the LP relaxation exactly, they provide a bound on its optimal value which is essential for pruning the search space during branch-and-bound search which is the usually accepted method for optimally solving WCSPs.

A similar connection between coordinate-wise minimization and local consistencies was identified in [148] for general unconstrained convex piecewise-affine functions. In detail, [148] found a connection between a certain local consistency condition and fixed points of coordinate-wise minimization with a special update rule. This was further developed in the author's master thesis [48a].

The above-mentioned update rule from [148] is in fact a special case of the relative-interior rule, which was proposed in [150, 151a] and is one of the baselines for this work. To motivate this rule, realize that the set of block-minimizers in BCD may generally contain multiple elements and, in such case, one has to choose a single element from this set to perform the BCD update. The relative-interior rule [150, 151a] additionally requires that the minimizer is chosen from the relative interior of the set of block-minimizers. Although this rule is not worse than any other update rule for choosing non-unique block-minimizers, the local minima of BCD following this rule can still be arbitrarily bad.

In this dissertation, we aim to extend the aforementioned constraint-propagation-based methods to the more general setting of optimizing arbitrary LP problems, investigate applicability of BCD to linear programs, and, still focusing on linear programs, link theory of BCD to constraint programming.

Structure and Contributions

Let us now overview the structure and aims of the thesis which is divided into five main chapters:

- §1: We begin with providing an overview of the necessary background and related work that spans over five areas:
  - §1.1: Linear programs and inference in systems of linear inequalities and equalities.
  - §1.2: BCD applied to convex optimization problems with focus on linear programs, relative-interior rule, properties of BCD, and types of local minima occurring in BCD.
  - §1.3: Order theory which forms the theoretical basis for constraint propagation.

§1.4: Constraint satisfaction problem (CSP), local consistencies, and their enforcing by propagators.

§1.5: WCSP and LP-based methods for bounding its optimal value.

- §2: Using constraint propagation to optimize linear programs (or even other convex optimization problems) is rare aside from the aforesaid approaches for bounding the optimal value of the WCSP. We propose a theoretical framework for (approximate) optimization of large-scale linear programs using constraint propagation, which generalizes the previously mentioned approaches. In this framework, we assume that an initial feasible point is available which we then try to improve, ideally make it optimal. We apply this framework to approximately optimize the dual LP relaxation of weighted Max-SAT and experimentally verify the quality of its stopping points.

  This part follows the explanation given by Dlask and Werner in the conference paper [52a] with some new insights.

- §3: It is usually not easy to generalize local consistencies from ordinary CSPs to WCSPs so that their enforcing by reparametrizing the WCSP improves the bound on its optimal value. Such a generalization may be even impossible because the level of local consistency that can be achieved via reparametrizations without introducing new weight functions (likely of higher arity) is limited. In cases when this is possible, the method for enforcing a (possibly soft) local consistency in the WCSP typically needs to be tailored to the particular chosen kind of local consistency.

  In contrast, we propose a method that is able to improve the bound on the WCSP optimal value using any kind of constraint propagation without introducing new weight functions. For this, we use an LP formulation (i.e., not a relaxation) of the WCSP which can be interpreted as minimizing an upper bound over its super-reparametrizations (i.e., WCSPs whose objective value is the same or greater for all assignments). Although this formulation was already proposed [92], we newly show that it can be approximately optimized using any method that can (at least sometimes) detect unsatisability of a CSP. The resulting algorithm can be seen as an instance of the general framework for optimizing large-scale linear programs (in this case, with an exponential number of constraints) by constraint propagation from §2.

  The properties of this optimization problem and of super-reparametrizations have not been thoroughly studied in the literature and we aim to fill in this gap. As already mentioned, one of the benefits of this optimization problem is that it allows us to enforce arbitrarily strong local consistencies without introducing higher-arity weight functions, thus saving memory. The cost for this is that our method provides only a bound on the optimal value but may not preserve other properties of the input WCSP, such as the objective value for the individual assignments or the set of optimal assignments.

  Our results in §3 are based on the journal submission [56a] which is an improved version of the conference paper [55a], both authored by Dlask, Werner, and de Givry.

- §4: Seeing that the stopping points of some constraint-propagation-based algorithms for bounding the WCSP optimal value are related to the fixed points of several BCD algorithms (applied to its dual LP relaxation), we explain this connection and generalize it to arbitrary linear programs with any blocks of variables. As a by-product, we characterize the types of local minima encountered in BCD by local consistency conditions, thus link BCD to constraint propagation.

More generally, we de ne a class of optimization methods based on enforcing a suitable local consistency whose stopping points are related to xed points of BCD in an analogous way. In other words, we identify the precise constraint propagation rule that corresponds to BCD. Depending on the problem being optimized, this can be interpreted, e.g., as enforcing arc consistency or performing unit propagation.

The results described in §4 are an improved version of the conference paper [54a] by Dlask and Werner.

- §5: The classes of convex optimization problems for which the BCD xed points are global optima are currently not much broader than unconstrained smooth convex functions. Such classes also include the dual LP relaxation of acyclic, supermodular, or pairwise Boolean WCSPs.

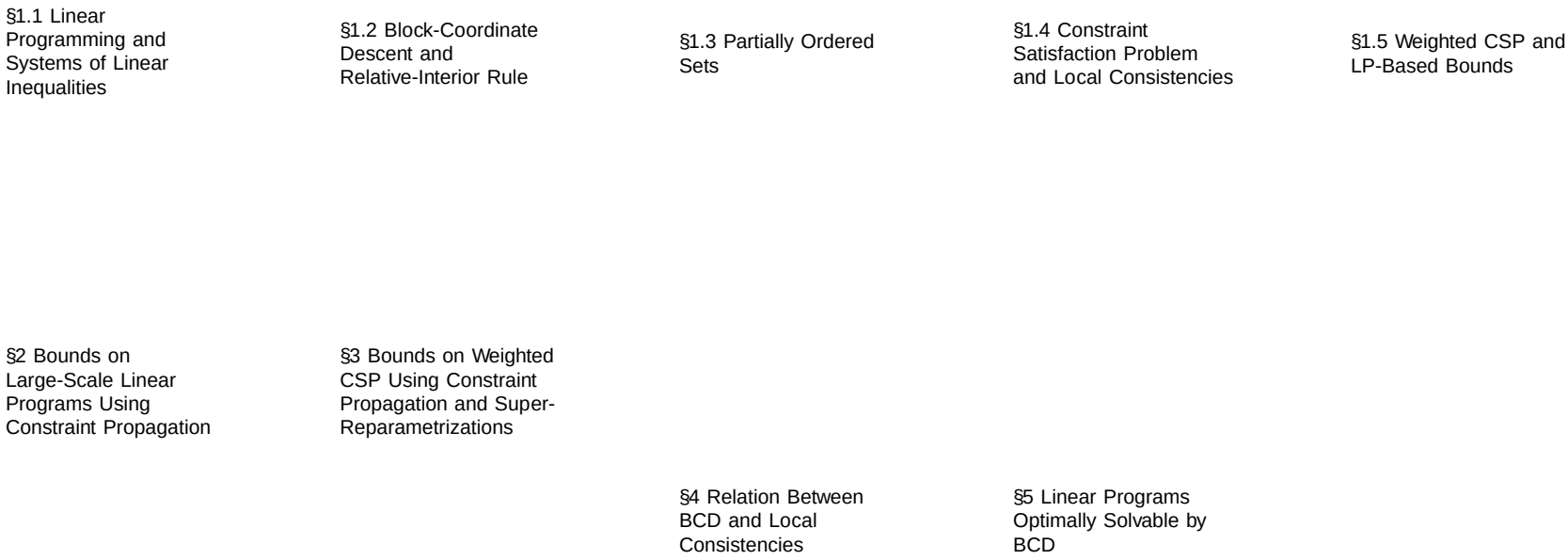
  Focusing on linear programs, we identify new classes of such problems. In fact, we even provide a characterization of linear programs that are optimally solvable by BCD in terms of constraint propagation. To be precise, the question of optimality of BCD xed points for linear programs can be translated to the question whether a precisely de ned constraint propagation rule can always detect infeasibility of a certain class of systems of linear inequalities and equalities. The newly identi ed classes of linear programs include, e.g., a suitable formulation of the maximum ow problem, LP relaxations of certain combinatorial problems, or the aforementioned LP formulation of a WCSP. Finally, we also explain why applicability of BCD depends on the precise formulation of the optimization problem and exemplify this phenomenon on non-trivial optimization problems, namely maximum ow, LP relaxation of weighted vertex cover, and a special LP relaxation of pairwise WCSP with Potts interactions.

  This chapter is mainly based on the results of Dlask and Werner in the journal paper [53a] (which is an improved version of the conference paper [51a] by the same authors) combined with a few insights from the conference papers [54a] (by Dlask and Werner) and [151a] (by Werner, Prusa, and Dlask), and some new parts.

We conclude the thesis by summarizing the achieved results and discussing possible directions for future research. The diagram on the next page visualizes the dependencies among sections in §1 and subsequent chapters in the body of the thesis.

Let us have a few remarks on the style of the presentation. Our contributions (in §2-§5) are organized in a `method-oriented' (rather than `application-oriented') manner: an idea/approach is always rst explained for the case of general linear programs and, after that, we exemplify how it manifests itself in speci c applications. The applications include not only linear programs connected to the WCSP but also LP relaxations/formulations of other problems. This implies that we may encounter the same optimization problem multiple times throughout the thesis, each time seen from a di erent perspective.

Finally, let us note that the topic of this dissertation is interdisciplinary { we base our research on local consistencies from the eld of constraint programming, BCD methods from optimization, and message-passing algorithms from computer vision and machine learning. This implies that there are typically multiple options for notation, nomenclature etc. that were developed independently in di erent elds. Therefore, e.g., a reader coming from the eld of optimization may nd some notation from constraint programming unusual and vice versa. We provide an overview of our notation and used abbreviations in the appendix.



Dependencies among the sections in §1 and subsequent chapters. Black arrows indicate that a section/chapter signi cantly depends on another whereas gray arrow is only a slight dependence. The full set of dependencies is obtained as the transitive closure of this diagram.

Chapter 1

Background

In this chapter, we provide the necessary background for linear programming, systems of linear inequalities, block-coordinate descent, partially ordered sets, and constraint satisfaction problems. We also review the weighted constraint satisfaction problem and approaches that are used for obtaining an upper bound on its optimal value. Up to a few minor insights, we do not present any novel results in this chapter but only overview already known pieces of knowledge which also help us introduce and exemplify our notation.

1.1 Linear Programming and Systems of Linear Inequalities

Linear programming is well known and finds its use in various applications [103, 24, §4.3.1]. In general, a linear program seeks to minimize or maximize a linear objective function of a finite number of variables over a set determined by a finite number of linear inequalities and equalities (i.e., a polyhedron). To each linear program, one can write its dual linear program and these optimization problems are connected by duality theorems. This construction is symmetric, so we can talk about being mutually dual.

Although linear programs come in various forms (e.g., containing both inequalities and equalities, non-negative and real-valued variables etc.), we consider the following form of a primal-dual pair

$$\max c^\top x \qquad \min b^\top y \tag{1.1a}$$

$$Ax = b \qquad y \in \mathbb{R}^m \tag{1.1b}$$

$$x \geq 0 \qquad A^\top y \geq c \tag{1.1c}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ are constants and $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ are variables. We denote by x_j the j-th component of vector x (similarly for y, b, c) and by A^i and A_j the i-th row and j-th column of A where $i \in [m] = \{1, \ldots, m\}$ and $j \in [n] = \{1, \ldots, n\}$, respectively. $A^\top$ is the transpose of A. We will refer to the left-hand problem (1.1) as the primal and to the right-hand problem (1.1) as the dual.

Any linear program can be easily transformed into the form of the primal or the dual (1.1) (see [103, §1.1 and §4.1]). We note that whenever we write a pair of mutually dual linear programs, we always write a constraint and the corresponding Lagrange multiplier on the same line, as in (1.1). For any primal-dual pair, strong duality holds:

Theorem 1.1 (Strong duality [123, §7.4, 103, §6.1]). For any primal-dual pair, only one of the following cases can happen:

(a) Both the primal and the dual are feasible and their optimal values coincide.

(b) Both the primal and the dual are infeasible.

(c) The primal is unbounded and the dual is infeasible or vice versa.

Optimality conditions for feasible solutions of linear programs are given by the well-known complementary slackness conditions that are stated in the following theorem.

Theorem 1.2 (Complementary slackness [103, 110]) Let $x \in R^n$ and $y \in R^m$ be feasible for the primal and the dual (1.1), respectively. The following are equivalent:

(a) x and y are optimal for the primal and the dual, respectively,

(b) $\forall j \in [n]: x_j (A_j^\top y - c_j) = 0$, i.e., $\forall j \in [n]: (x_j = 0) \vee (A_j^\top y = c_j)$.

For brevity of notation, we define the mappings $\alpha : R^n \to 2^{[n]}$ and $\beta : R^m \to 2^{[n]}$ by

$$\alpha(x) = \{ j \in [n] \mid x_j = 0 \} \tag{1.2a}$$

$$\beta(y) = \{ j \in [n] \mid A_j^\top y = c_j \}; \tag{1.2b}$$

so that $\alpha(x)$ is the index set of the primal constraints (1.1c) that are active (i.e., satisfied with equality [2]) at x. Similarly, $\beta(y)$ is the index set of the dual constraints (1.1c) that are active at y. Using this notation, statement (b) in Theorem 1.2 can be expressed as $\beta(y) \cup \alpha(x) = [n]$.

1.1.1 Relative Interior and Strict Complementarity

In the sequel, we will frequently utilize the notion of relative interior of a convex set. We now recall its definition and important properties.

Definition 1.1 ([100, Definition 2.1.1]). Let $S \subseteq R^n$ be a convex set. The *relative interior* of S, denoted by $\operatorname{ri} S$, is the topological interior of S relative to the affine hull of S, i.e.,

$$\operatorname{ri} S = \{ x \in S \mid \exists r > 0 : B_r(x) \cap \operatorname{aff} S \subseteq S \} \tag{1.3}$$

where $B_r(x) = \{ y \in R^n \mid \|x - y\| \leq r \}$ is the ball centered at x with radius $r > 0$ and $\operatorname{aff} S$ is the affine hull of S.

Example 1.1 ([100, Table 2.1.1, 150, §4, 151a, §3]). Let $x, x^0 \in R^n$. We have that $\operatorname{ri}\{x\} = \{x\}$, i.e., the relative interior of a singleton set is the set itself. Furthermore, if $x \neq x^0$, then $\operatorname{ri}[x, x^0] = [x, x^0) = [x, x^0] \setminus \{x, x^0\}$ where $[x, x^0] = \{ x + (1 - \lambda) x^0 \mid 0 \leq \lambda \leq 1 \}$ is the line segment between x and x^0. ◇

It follows directly from Definition 1.1 that, for a convex set S, $\operatorname{ri} S \subseteq S$, and it is known [100, Theorem 2.1.3] that $\operatorname{ri} S = \emptyset$ if and only if $S = \emptyset$.

Consequently, since the set of optimal solutions of a linear program is always a convex set, its non-emptiness is equivalent to non-empty relative interior. The *strict complementary slackness* condition can be used to determine whether a primal solution x and a dual solution y lie in the relative interior of the set of optimal solutions of a primal and a dual linear program, respectively.

Theorem 1.3 (Strict complementary slackness [66, 80, 69, 158]) Let $x \in R^n$ and $y \in R^m$ be feasible for the primal and the dual (1.1), respectively. The following are equivalent:

[2] In general, an inequality $c^\top x \leq d$ is called active for some x if $c^\top x = d$ [122, §5.5, 62, 24, §4.1.1].

(a) x and y are in the relative interior of the set of optimizers of the primal and the dual, respectively,

(b) $\{\mathcal{A}(x), \mathcal{I}(y)\}$ is a partition of $[n]$, i.e., $\forall j \in [n]: (x_j = 0) \not\equiv (A_j^\top y = c_j)$ where $\not\equiv$ denotes exclusive disjunction.

As a corollary, for any x and y in the relative interior of the set of optimal solutions of the primal and the dual, respectively, the partition $\{\mathcal{A}(x), \mathcal{I}(y)\}$ is the same and is typically referred to as the optimal partition of $[n]$ [80, 4, 69, 104].

1.1.2 Convex Piecewise-Affine Objective

In multiple places throughout this thesis, we will utilize the well-known trick [16, §1.3, 24, §4.3.1] that allows us to formulate the problem of minimizing a convex piecewise-affine function over a polyhedron as a linear program and vice versa.

Formally, we address the optimization problem

$$\min \sum_{k \in K} \max_{l \in L_k} (c_{k,l}^\top x + d_{k,l}) \tag{1.4a}$$

$$x \in X \tag{1.4b}$$

where $X \subseteq \mathbb{R}^n$ is a polyhedron, K is a finite set, L_k a finite non-empty set for each $k \in K$, and $c_{k,l} \in \mathbb{R}^n$ and $d_{k,l} \in \mathbb{R}$ are given vectors and real numbers for each $k \in K, l \in L_k$. The objective (1.4a) is a convex piecewise-affine[3] function of x.

It is easy to see that (1.4) can be reformulated as the linear program

$$\min \sum_{k \in K} z_k \tag{1.5a}$$

$$z_k \geq c_{k,l}^\top x + d_{k,l} \qquad \forall k \in K, l \in L_k \tag{1.5b}$$

$$x \in X \tag{1.5c}$$

$$z \in \mathbb{R}^K \tag{1.5d}$$

where we introduced auxiliary variables $z \in \mathbb{R}^K$. The reason is that for any $x \in X$, we can define

$$z_k = \max_{l \in L_k} (c_{k,l}^\top x + d_{k,l}) \qquad \forall k \in K \tag{1.6}$$

so that (x, z) is feasible for (1.5) and the objective values for both (1.4) and (1.5) coincide. On the other hand, for any (x, z) feasible for (1.5), x is also feasible for (1.4) and the objective (1.4a) is lower than or equal to the objective (1.5a) due to $z_k \geq \max_{l \in L_k} (c_{k,l}^\top x + d_{k,l})$ for all $k \in K$ by (1.5b). Consequently, the optimal values of (1.4) and (1.5) are equal.

1.1.3 Systems of Linear Inequalities and Linear Inference

Let us now point our attention to the logical view of linear inequalities. We briefly overview the basic properties of inference in a system of linear inequalities (and possibly equalities) and Farkas' lemma.

[3] This is called piecewise-linear in [16, §1.3]. However, the terminology is not unified [24, Example 3.5] as a linear function may not contain a constant term in some formalisms, so we use the unambiguous term piecewise-affine, as in [148, 48a].

In the sequel, we will call a system of linear inequalities (and equalities) *feasible* if it has a solution (i.e., the polyhedron defined by this system is non-empty). Otherwise, it is *infeasible*.

Moreover, for $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $d \in \mathbb{R}$, we will say that a system $Ax \leq b$ *implies* $c^\top x \leq d$ if $\forall x \in \mathbb{R}^n : Ax \leq b \Rightarrow c^\top x \leq d$ (i.e., if $c^\top x \leq d$ holds for all $x \in \mathbb{R}^n$ satisfying $Ax \leq b$). Analogously, a system $Ax \leq b$ *implies* $c^\top x = d$ if $\forall x \in \mathbb{R}^n : Ax \leq b \Rightarrow c^\top x = d$.[4]

As we will discuss, the logic of linear inequalities is simple in the sense that any implied inequality can be derived as a suitable combination of the existing inequalities (up to certain technical details). In particular, a contradictory inequality can be derived from any infeasible system in this way [76, §17.2–§17.3, 103, §6.4]. In contrast to the discrete setting of constraint propagation, which is considered later in §1.4.1, deciding whether a system of linear inequalities implies another given linear inequality is solvable in polynomial time by posing the problem as a linear program.

Formally, the logic of linear inequalities is described by the *affine form of Farkas' lemma*:

Theorem 1.4 (Affine form of Farkas' lemma [123, §7.6]). Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $d \in \mathbb{R}$. The following are equivalent:

(a) $Ax \leq b$ implies $c^\top x \leq d$,

(b) $Ax \leq b$ is infeasible or there is $y \geq 0$ such that $A^\top y = c$ and $b^\top y \leq d$.

Proof. Direction (b) $\Rightarrow$ (a) is immediate: for y satisfying (b) and any x satisfying $Ax \leq b$, we have $c^\top x = y^\top Ax \leq y^\top b \leq d$.

The other direction (a) $\Rightarrow$ (b) follows from[5] strong duality applied to the primal-dual pair (1.1) after changing A to $A^\top$, interchanging b with c and x with y. In particular, if $A^\top y \geq c$ implies $b^\top y \geq d$, then the dual (1.1) is infeasible or it is feasible and bounded and its optimal value is at least d. If it is feasible and bounded, the primal is too, and its optimal solution x satisfies $Ax = b; x \geq 0$ and $c^\top x \geq d$. □

Clearly, $c^\top x \leq d'$ implies $c^\top x \leq d$ if and only if $d' \leq d$ (or if $c^\top x \leq d'$ is infeasible). Thus, one can interpret Theorem 1.4 as follows: a feasible system $Ax \leq b$ implies $c^\top x \leq d$ if and only if a non-negative combination of the inequalities in $Ax \leq b$ implies $c^\top x \leq d$ (where the non-negative combination of inequalities is $c^\top x \leq d'$ where $c = A^\top y$ and $d' = b^\top y$). The vector y from Theorem 1.4 contains the coefficients of the non-negative combination and thus stores how the implied inequality was derived. Therefore, such a vector y can be called the *certificate* or the *cause vector* of the implied inequality.

Example 1.2. Let the system $Ax \leq b$ be given by the constraints

$$2x_1 + x_2 \leq 0 \tag{1.7a}$$
$$x_2 \leq 2 \tag{1.7b}$$
$$x_1 + x_2 \leq 1. \tag{1.7c}$$

[4] For completeness, we note that $Ax \leq b$ does not imply $c^\top x \leq d$ if $\exists x \in \mathbb{R}^n$ such that $Ax \leq b$ and $c^\top x < d$. Also, $Ax \leq b$ does not imply $c^\top x = d$ if $\exists x \in \mathbb{R}^n$ such that $Ax \leq b$ and $c^\top x \neq d$.

[5] Conversely, Theorem 1.4 can be used to prove strong duality in linear programming [76, §17.3.1].

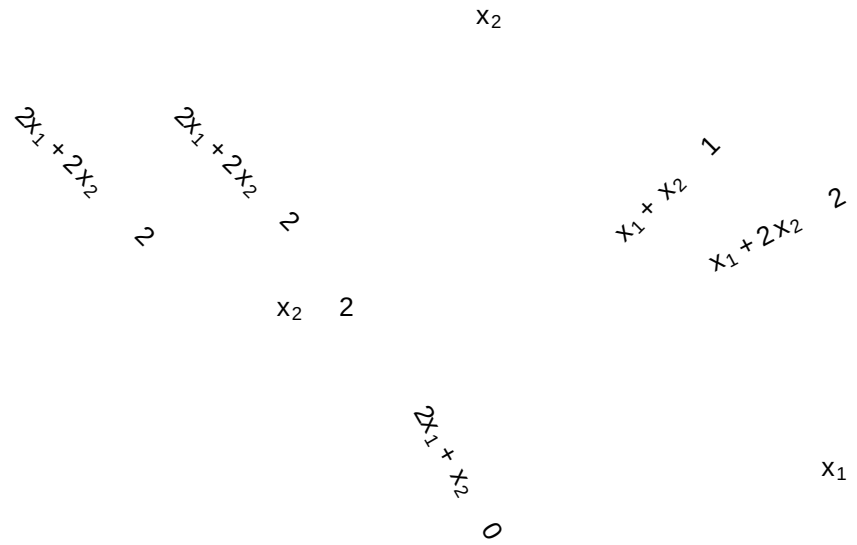


Figure 1.1: Illustration to Example 1.2. The half-planes determined by the constraints (1.7) are indicated by full lines with arrows and their intersection is shaded. The half-planes determined by the three implied inequalities from Example 1.2 are indicated by dashed lines with arrows.

This system implies $x_1 + 2x_2 \quad 2$, i.e., $c^\top x \quad d$ for $c = (\ 1; 2)$ and $d = 2$ because for $y = (\frac{1}{4}; \frac{1}{4}; \frac{3}{2})$, we have that $A^\top y = (\ 1; 2) = c$ and $b^\top y = 2 = d$. Symbolically, this can be expressed as

$$\tfrac{1}{4}\ (2x_1 + x_2 \quad 0) + \tfrac{1}{4}\ (x_2 \quad 2) + \tfrac{3}{2}\ (\ x_1 + x_2 \quad 1) = (\ x_1 + 2x_2 \quad 2): \qquad (1.8)$$

Another implied inequality is, e.g., $2x_1 + 2x_2 \quad 2$ whose cause vector is $y^0 = (1; 1; 0)$. In detail, we have $A^\top y^0 = (2; 2)$ and $b^\top y^0 = 2$. Thus, (1.7) also implies $2x_1 + 2x_2 \quad d$ for any $d \quad 2$ (e.g., $d = \quad 2$) because for the same y^0, we have $A^\top y^0 = (2; 2)$ and $b^\top y^0 = 2 \quad d$.

The set defined by (1.7) along with the three mentioned implied inequalities is depicted in Figure 1.1. 4

The famous Farkas' lemma can be obtained as a corollary of Theorem 1.4:

Corollary 1.1 (Farkas' lemma [123, §7.3, 103, §6.4]). Let $A \in \mathbb{R}^{m \quad n}$ and $b \in \mathbb{R}^m$. Exactly one of the following systems is feasible:

(a) $Ax \quad b$,

(b) $y \quad 0;\ A^\top y = 0;\ b^\top y > 0$.

Proof. We proceed analogously to the proof in [76, Corollary 62]. Let us define $u \in \mathbb{R}^m$ by $u_i = 1$ for all $i \in [m]$ (so that vector u contains only ones as its components). First, note that the system $Ax + tu \quad b$ (with an additional variable $t \in \mathbb{R}$) is always feasible because t can be set large enough. Second, it is easy to see that $Ax \quad b$ is infeasible if and only if $Ax + tu \quad b$ implies $t \quad d$ for any $d > 0$. Applying Theorem 1.4 to this implication yields that $Ax + tu \quad b$ implies $t \quad d$ if and only if there is y satisfying

$$y \quad 0;\ A^\top y = 0;\ u^\top y = 1;\ b^\top y \quad d: \qquad (1.9)$$

Since $d > 0$ can be chosen small enough, system (b) in the corollary and system (1.9) are equisatisfiable (i.e., both systems are feasible or both are infeasible). In detail, any $\mathbf{y}$ satisfying (1.9) with $d > 0$ also satisfies system (b) and any y satisfying system (b) can be scaled to satisfy (1.9) for some $d > 0$. □

See that any y satisfying system (b) in Corollary 1.1 is a certificate vector of inequality $0^\top x \geq d$ with $d = b^\top y > 0$ which is clearly infeasible. Thus, one can interpret Farkas' lemma as follows: system $Ax \geq b$ is infeasible if and only if a non-negative combination of the linear inequalities in $Ax \geq b$ is infeasible (i.e., if the system implies an infeasible inequality). Thus, any y satisfying system (b) in Corollary 1.1 is a certificate of infeasibility of system (a). System (b) is called the Farkas alternative to system (a).

There exist several variants of Farkas' lemma and its affine form, each corresponding to a different form of the system, possibly including also equality constraints or non-negative variables [123, §7.3, 103, §6.4]. For each such system, there is a corresponding Farkas alternative system enjoying the same properties as in Corollary 1.1.[6] We show an example below for the system

$$Ax = b \tag{1.10a}$$
$$x \geq 0. \tag{1.10b}$$

Theorem 1.5. Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $d \in \mathbb{R}$. The following are equivalent:

(a) $Ax = b,\ x \geq 0$ implies $c^\top x \geq d$,

(b) $Ax = b,\ x \geq 0$ is infeasible or there is $y \in \mathbb{R}^m$ such that $b^\top y \geq d$ and $A^\top y \leq c$.

In case that (1.10) is feasible and implies $c^\top x \geq d$, any y satisfying the conditions in statement (b) in Theorem 1.5 is a certificate of $c^\top x \geq d$ and it encodes how this inequality can be obtained from the original system (1.10). Indeed, y contains the coefficients with which the equalities in (1.10a) can be combined to obtain the equality $(y^\top A)x = y^\top b$. By non-negativity of x, we have $c^\top x \geq (y^\top A)x$ due to $c \geq A^\top y$, so $c^\top x \geq (y^\top A)x = y^\top b \geq d$ holds for any x feasible for (1.10). The corresponding form of Farkas' lemma for system (1.10) is the following:

Corollary 1.2 ([123, Corollary 7.1d, 103, Proposition 6.4.1]). Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Exactly one of the following systems is feasible:

(a) $Ax = b,\ x \geq 0$,

(b) $A^\top y \geq 0,\ b^\top y < 0$.

Consequently, any y satisfying system (b) in Corollary 1.2 is a certificate of infeasibility of system (a). Note, such y encodes an equality in the form $(y^\top A)x = y^\top b$ where the left-hand side is a non-negative sum of non-negative variables but the right-hand side is negative. Finally, system (b) in Corollary 1.2 is called the Farkas alternative to system (a).

Remark 1.1. Theorem 1.5 and Corollary 1.2 apply to the system (1.10) which is the set of feasible solutions of the primal (1.1). We chose to present Theorem 1.4 and Corollary 1.1 (which consider a system in the form of the dual constraints (1.1b)-(1.1c)) earlier because the aforementioned concepts are easier to illustrate for such a form.

[6] This is analogous to the fact that (strong) duality, complementary slackness etc. in linear programming hold for problems in any form (e.g., including both equalities and inequalities, non-negative and real-valued variables etc.) even though we presented the theory only for the particular case of (1.1).

1.1.3.1 Always-Active Inequalities

For some systems of linear inequalities, it may happen that an inequality constraint is active for any solution of the system, i.e., it always holds as an equality constraint. Such constraints are called *always active* [62] (a.k.a. implied equalities [68] or implicit equalities [131, 123, §8.1, 122, §5.6]). A formal definition is given below.

Definition 1.2. Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $i \in [m]$. The inequality $A^i x \leq b_i$ is always active within the system $Ax \leq b$ if $Ax \leq b$ implies $A^i x = b_i$.

All always-active inequalities in a system of linear inequalities (and possibly equalities) can be characterized using relative interior. We present this result for systems in the form $Ax = b, x \geq 0$ and $Ax \leq b$ in Theorem 1.6 and Theorem 1.7, respectively.

Theorem 1.6 (cf. [68, Theorem 8]). Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $j \in [n]$, and[7]

$$x \in \operatorname{ri} \{ x \in \mathbb{R}^n \mid Ax = b, x \geq 0 \}. \tag{1.11}$$

System $Ax = b, x \geq 0$ implies $x_j = 0$ if and only if $j \in \mathcal{I}(x)$, i.e., $x_j = 0$.

Proof. Consider the primal-dual pair (1.1) with $c = 0$, i.e., zero objective function in the primal. Trivially, feasibility is equivalent to optimality for the primal, so the primal (and hence also the dual) is feasible and bounded.

Let y be from the relative interior of optimizers of the dual (1.1). Any x feasible for the primal necessarily satisfies complementary slackness with dual-optimal y, i.e., $\mathcal{I}(y) \subseteq \mathcal{I}(x) = [n]$. Therefore, the system implies $x_j = 0$ for all $j \in [n] \setminus \mathcal{I}(y) = \mathcal{I}(x)$ where the set equality $[n] \setminus \mathcal{I}(y) = \mathcal{I}(x)$ is given by strict complementary slackness. Since x satisfies $Ax = b, x \geq 0$ and $x_j > 0$ for all $j \in [n] \setminus \mathcal{I}(x)$, the inequality $x_j \geq 0$ for $j \in [n] \setminus \mathcal{I}(x)$ is not always active. □

Theorem 1.7 ([68, Theorem 8]). Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $x \in \operatorname{ri} \{ x \in \mathbb{R}^n \mid Ax \leq b \}$, and $i \in [m]$. System $Ax \leq b$ implies $A^i x = b_i$ if and only if $A^i x = b_i$.

Proof. Analogous to Theorem 1.6. □

1.2 Block-Coordinate Descent and Relative-Interior Rule

In this section, we will formally define block-coordinate descent (BCD)[8], state some of its properties, and pay particular attention to the relative-interior rule that was proposed in [150] and later developed in [151a]. Some parts of this section are based on the review that was given in [54a].

For the purposes of the sequel, we define new notation. For $B = \{i_1, \ldots, i_{|B|}\} \subseteq [m]$ and $y \in \mathbb{R}^m$, y_B stands for the restriction of y onto the set B, i.e., $y_B = (y_{i_1}, \ldots, y_{i_{|B|}})$ where the order of the components is defined by the total order on B inherited from the natural total order on $[m]$ given by $\leq$.

[7] By assuming the existence of such x, we implicitly assume that $Ax = b, x \geq 0$ is feasible.

[8] Also called block-coordinate minimization. In case that the objective is to be maximized, one usually speaks about block-coordinate ascent

Suppose we minimize a continuous convex function $f : Y \to \mathbb{R}$ on a non-empty closed convex set $Y \subseteq \mathbb{R}^m$. Here, we assume for simplicity that the set $\operatorname{argmin}_{y \in Y} f(y)$ is non-empty, hence f is bounded from below on Y. For brevity of notation, we formulate this optimization problem as the unconstrained minimization of the extended-valued function $f : \mathbb{R}^m \to \mathbb{R}_{+\infty}$ (where $\mathbb{R}_{+\infty} = \mathbb{R} \cup \{+\infty\}$) defined by

$$f(y) = \begin{cases} f(y) & \text{if } y \in Y \\ +\infty & \text{if } y \notin Y \end{cases}. \tag{1.12}$$

Furthermore, it is assumed that a set $\mathcal{B} \subseteq 2^{[m]}$ of blocks of variables and an initial feasible solution $y^1 \in Y$ are provided. BCD in each iteration chooses a single block $B \in \mathcal{B}$ and minimizes the function f over variables y_B while keeping the remaining variables $y_{[m] \setminus B}$ fixed, i.e., updates y^k to some y^{k+1} satisfying

$$y^{k+1}_B \in \operatorname*{argmin}_{y' \in \mathbb{R}^B} f\left(y', y^k_{[m] \setminus B}\right) \tag{1.13a}$$

$$y^{k+1}_i = y^k_i \qquad \forall i \in [m] \setminus B. \tag{1.13b}$$

By repeatedly performing updates (1.13) with different blocks $B \in \mathcal{B}$, the points y^k remain in the feasible set Y and the sequence of objective values $f(y^k)$ is non-increasing.

Remark 1.2. To avoid any ambiguity, let us comment on a slight abuse of notation in (1.13a). There, we have a vector $y' \in \mathbb{R}^B$ and a vector $y^k_{[m] \setminus B} \in \mathbb{R}^{[m] \setminus B}$. The components of these vectors can be arranged into a single vector $y \in \mathbb{R}^m = \mathbb{R}^{[m]}$ and $\left(y', y^k_{[m] \setminus B}\right)$ then corresponds to y. For example, if $m = 5$ and $B = \{2, 3\}$, we have $y' = (y'_2, y'_3)$, $y^k_{[m] \setminus B} = (y^k_1, y^k_4, y^k_5)$, and $y = \left(y', y^k_{[m] \setminus B}\right) = (y^k_1, y'_2, y'_3, y^k_4, y^k_5)$.

Example 1.3. To illustrate the iterations (1.13), let $f(y) = 5y_1 + 2y_2 + y_3$, $\mathcal{B} = \{\{1\}, \{2, 3\}\}$, $Y = \{y \in \mathbb{R}^3 \mid y_1 \geq 0, y_2 \geq 2, y_2 + y_3 = 1\}$, and $y^1 = (4, 7, -6) \in Y$. In the first iteration, we update along $B = \{1\}$ which means that we optimize over y_1 while keeping $y_{\{2,3\}}$ (i.e., y_2 and y_3) constant. This yields the updated point $y^2 = (0, 7, -6)$ (which satisfies (1.13) for $k = 1$ and $B = \{1\}$). In the second iteration, we update along $B = \{2, 3\}$, i.e., we optimize over $y_{\{2,3\}}$ and keep y_1 constant which results in the updated point $y^3 = (0, 2, -1)$ (which satisfies (1.13) for $k = 2$ and $B = \{2, 3\}$). ◁

The class of convex optimization problems for which BCD provably converges to global minima (or where the fixed points are global minima) is currently quite narrow, including, e.g., unconstrained continuously differentiable convex functions with unique univariate minima [15, §2.7], unconstrained differentiable pseudoconvex functions [155], or continuously differentiable functions with a single linear equality constraint and box constraints [12]. For other examples, we refer to [137] and references therein. For general convex non-differentiable or constrained functions, the fixed points of this approach may not be global minima.

1.2.1 Relative-Interior Rule

The set $\operatorname{argmin}_{y' \in \mathbb{R}^B} f\left(y', y^k_{[m] \setminus B}\right)$ of block-wise minimizers from (1.13a) is a non-empty convex set that may in general contain more than one element. In practical implementations, one needs to choose a single element from this set.

It was proposed in [150, 151a] to additionally require that y^{k+1}_{B} is chosen from the relative interior of this set, i.e.,

$$y^{k+1}_{B} \in \operatorname{ri} \operatorname*{argmin}_{y' \in \mathbb{R}^{B}} f(y', y^{k}_{[m] \setminus B}) \tag{1.14a}$$

$$y^{k+1}_{i} = y^{k}_{i} \qquad \forall i \in [m] \setminus B. \tag{1.14b}$$

As discussed previously in §1.1.1, the relative interior of a non-empty convex set is non-empty, so an update satisfying (1.14) is always possible.

We will now summarize the main results of [150, 151a].

Definition 1.3 ([150, 151a]) A point $y \in Y$ is

- a local minimum (LM) of f on Y w.r.t. $\mathcal{B}$ if

$$y_{B} \in \operatorname*{argmin}_{y' \in \mathbb{R}^{B}} f(y', y_{[m] \setminus B}) \tag{1.15}$$

  (i.e., (1.13) for $y^{k+1} = y^{k} = y$) holds for all $B \in \mathcal{B}$,

- an interior local minimum (ILM) of f on Y w.r.t. $\mathcal{B}$ if

$$y_{B} \in \operatorname{ri} \operatorname*{argmin}_{y' \in \mathbb{R}^{B}} f(y', y_{[m] \setminus B}) \tag{1.16}$$

  (i.e., (1.14) for $y^{k+1} = y^{k} = y$) holds for all $B \in \mathcal{B}$,

- a pre-interior local minimum (pre-ILM) of f on Y w.r.t. $\mathcal{B}$ if [9] there is an ILM y' of f on Y w.r.t. $\mathcal{B}$ such that y is in a face of the set Y containing y' in its relative interior.

For brevity, when f, Y, or $\mathcal{B}$ is known from context, we omit `of $f$ on $Y$' or `w.r.t. $\mathcal{B}$'. In §5.4, we will also address maximization problems where the analogous terms *local maximum*, *interior local maximum*, and *pre-interior local maximum* are used. In case of coordinate-wise minimization, i.e., when $\mathcal{B} = \{\{i\} \mid i \in [m]\}$, we say that we optimize $\min_{y \in Y} f(y)$ *coordinate-wise* and simply write, e.g., `ILM w.r.t. individual variables' instead of `ILM w.r.t. $\mathcal{B} = \{\{i\} \mid i \in [m]\}$'. By a minimum/maximum without adjectives, we always mean global minimum/maximum.

Clearly, the fixed points of BCD algorithm following the updates (1.13) are local minima. We emphasise that this is different from the usual notion of a local minimum: here (by Definition 1.3), the objective in a local minimum cannot be improved by any single update (1.13) instead of an arbitrary update within some neighborhood. The fixed points of BCD with the relative-interior rule (1.14) are interior local minima.

Note, every ILM is a pre-ILM and every pre-ILM is an LM [150, 151a]. Crucial properties of BCD with and without the relative-interior rule are given below.

Theorem 1.8 ([150, 151a]) Let $(B_k)_{k=1}^{\infty}$ be a sequence of blocks $B_k \in \mathcal{B}$ that contains each element of $\mathcal{B}$ an infinite number of times. Let $(y^k)_{k=1}^{\infty}$ be a sequence produced by the BCD method, where the blocks are visited in the order given by $(B_k)_{k=1}^{\infty}$.

(a) If $(y^k)_{k=1}^{\infty}$ satisfies (1.14) and y^1 is an ILM, then y^k is an ILM for all k.

[9] As in [150, 151a], this definition of a pre-ILM applies only when f is linear. Nevertheless, as we do not use the definition of a pre-ILM explicitly, we omit a general definition of a pre-ILM. Instead, we will rely on the characterization of pre-ILMs that is given by Theorem 1.8 stated later.

(b) If $(y^k)_{k=1}^{1}$ satisfies (1.14) and y^1 is a pre-ILM, then y^k is an ILM for some k.

(c) If $(y^k)_{k=1}^{1}$ satisfies (1.13) and y^1 is a pre-ILM, then $f(y^k) = f(y^1)$ for all k.

(d) If $(y^k)_{k=1}^{1}$ satisfies (1.14) and y^1 is not a pre-ILM, then $f(y^k) < f(y^1)$ for some k.

By Theorem 1.8c, if y^1 is a pre-ILM, the objective cannot be improved by any further BCD iterations (1.13), even with the relative-interior rule (1.14). On the other hand, if y^1 is not a pre-ILM, BCD with the relative-interior rule (1.14) inevitably improves the objective after a finite number of iterations by Theorem 1.8d. In this sense, the relative-interior rule is not worse than any other update rule for choosing non-unique block-minimizers.

Even with the relative-interior rule, the fixed points of BCD need not be global minima, as the following example shows.

Example 1.4 (cf. [148, Example 2, 119, Figure 7.3, 63, Figure 4]). Let $Y = \mathbb{R}^2$ and $f : Y \to \mathbb{R}_+$ be defined by $f(y_1, y_2) = \max\{2y_2 - y_1, 2y_1 - y_2, 0\}$. Even though f is convex, any point $y \in \mathbb{R}^2$ satisfying $y_1 = y_2 > 0$ is an ILM w.r.t. individual variables (i.e., w.r.t. $B = \{\{1\}, \{2\}\}$) but not a global minimum. Figure 1.2a depicts several contours of f and also a non-optimal ILM. Indeed, the highlighted point is an ILM w.r.t. individual variables because any movement from this point along any single coordinate increases the objective. 4

Example 1.5 (cf. [150, §2, 151a, §2]). Let $Y = \{y \in [0,2]^2 \mid y_1 + y_2 \geq 1\}$ and $f(y_1, y_2) = y_2$. Suppose that we initialize BCD with point y^1, as indicated in Figure 1.2b, and optimize it coordinate-wise, i.e., we apply BCD with blocks $B = \{\{1\}, \{2\}\}$.

First, we update y^1 to y^2 by decreasing coordinate y_2 to improve the objective and attain the unique coordinate-wise optimum. Point y^2 is a local minimum because both components are coordinate-wise optimal. However, the choice for the optimizer w.r.t. coordinate y_1 is not unique and y_1^2 is not in the relative interior of optimizers, so we change the y_1 coordinate of y^2 and move from y^2 to y^3. Now, coordinate y_2 can be again decreased to improve the objective. After two similar iterations, we attain the point y^6 which is an ILM w.r.t. individual variables and also a global minimum.

Denoting by $[y, y']$ the line segment between y and y' (as in Example 1.1), the set of all LMs of f on Y w.r.t. individual variables is $[(0,1),(1,0)] \cup [(1,0),(2,0)]$. Pre-ILMs are $[(1,0),(2,0)]$ and ILMs are $\operatorname{ri}[(1,0),(2,0)]$. Global minima coincide with pre-ILMs. 4

Although the following corollary was not explicitly stated in [150, 151a], it is an immediate consequence of Theorem 1.8.

Corollary 1.3. The following are equivalent:

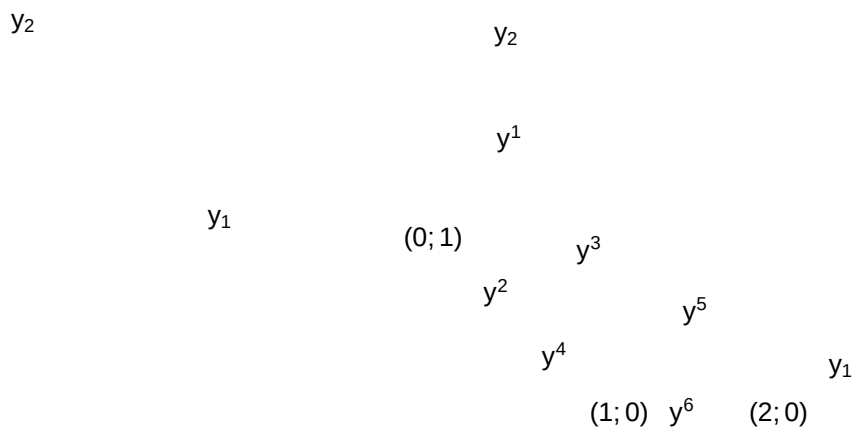
(a) every ILM of f on Y w.r.t. B is a global minimum,

(b) every pre-ILM of f on Y w.r.t. B is a global minimum.

Moreover, if these statements hold, then the set of global minima (i.e., $\operatorname{argmin}_{y \in Y} f(y)$) coincides with the set of pre-ILMs of f on Y w.r.t. B.

Proof. The implication (b) $\Rightarrow$ (a) is clear because every ILM is also a pre-ILM. For the other direction, let y^1 be a pre-ILM. By Theorem 1.8b and 1.8c, after performing a finite number of relative-interior updates (1.14) from y, we attain an ILM with the same objective.

One inclusion in the last statement follows already from (b). We prove the remaining part by contradiction: if a global minimum is not a pre-ILM, then, by Theorem 1.8d, the objective must improve after a finite number of updates, which is impossible. □



(a) Contours of function f from Example 1.4. The set of optimizers is shaded in gray and a non-optimal ILM is highlighted.

(b) Five iterations of coordinate-wise minimization with the relative-interior rule starting from the initial point y^1.

Figure 1.2: Illustrations to Examples 1.4 and 1.5.

As a consequence of Corollary 1.3, we will be able to say that `(pre-)ILMs of f on Y w.r.t. $\mathcal{B}$ are global minima'.

Remark 1.3. The version of BCD considered in [150, 151a] is more general. In detail, one has a set $\mathcal{I}$ of subspaces of $\mathbb{R}^m$ and instead of (1.13), the updates are formulated as

$$y^{k+1} \in \operatorname{argmin}\{f(y') \mid y' \in (y^k + I) \cap Y\} \quad I \in \mathcal{I} \tag{1.17}$$

where $I \in \mathcal{I}$ is a chosen subspace along which we update and $y^k + I = \{y^k + y' \mid y' \in I\}$. This subsumes not only the previously described block-coordinate formulation but also optimization along a set of directions (i.e., when all subspaces from $\mathcal{I}$ have dimension 1). For our purposes, we do not need such a general formalism.

1.2.2 Convergence

The convergence properties of BCD with the relative-interior rule were discussed and analyzed in [150, §4.3, 151a, §5.2]. Here, we overview the parts that are most important for the sequel.

Formally, suppose that the block-coordinate updates (1.14) are determined by some mappings $u_B : Y \to Y$ for each $B \in \mathcal{B}$, i.e., for any $B \in \mathcal{B}$ and $y \in Y$,

$$u_B(y)_{|B} \in \operatorname*{ri\,argmin}_{y' \in \mathbb{R}^B} f(y', y_{|[m] \setminus B}) \tag{1.18a}$$

$$u_B(y)_i = y_i \qquad \forall i \in [m] \setminus B. \tag{1.18b}$$

In other words, the update of $y^k \in Y$ along a block of variables $B \in \mathcal{B}$ satisfying the relative-interior rule yields a point $y^{k+1} = u_B(y^k)$ that satisfies (1.14).

Furthermore, let us fix an ordering on the elements of $\mathcal{B}$ so that $\mathcal{B} = \{B_1, \ldots, B_n\}$ and define the composed mapping $U : Y \to Y$ by

$$U = \underbrace{\underbrace{(u_{B_n} \circ \cdots \circ u_{B_1})}_{\text{a single cycle of updates (1.14) in the specified order}} \circ \cdots \circ (u_{B_n} \circ \cdots \circ u_{B_1})}_{m+1 \text{ cycles of updates}}. \tag{1.19}$$

The mapping U performs $m+1$ cycles of updates along individual blocks from $\mathcal{B}$ in the specified order. For this setting, we have the following results from [150, 151a]:

Theorem 1.9 ([150, Theorem 18, 151a, Theorem 20]) Let $y \in Y$ and $\mathcal{B} \subseteq 2^{[m]}$. If $f(U(y)) = f(y)$, then y is a pre-ILM of f on Y w.r.t. $\mathcal{B}$ and $U(y)$ is an ILM of f on Y w.r.t. $\mathcal{B}$.

Theorem 1.10 ([150, Corollary 25, 151a, Corollary 22]) Let the mappings u_B, $B \in \mathcal{B}$ be continuous, Y^* be the set of pre-ILMs of f on Y w.r.t. $\mathcal{B}$, $y^1 \in Y$, and the sequence $(y^k)_{k=1}^{\infty}$ be defined by $y^{k+1} = U(y^k)$, $k \in \mathbb{N}$. If the sequence $(y^k)_{k=1}^{\infty}$ is bounded, then

$$\lim_{k \to \infty} d(Y^*, y^k) = 0 \tag{1.20}$$

where $d(Y^*, y^k) = \inf_{y \in Y^*} d(y, y^k)$ is the distance of y^k to the set Y^* w.r.t. any metric $d : Y \times Y \to \mathbb{R}_+$. I.e., the sequence $(y^k)_{k=1}^{\infty}$ converges to the set of pre-ILMs.

1.2.3 Reformulations of Problems

It is known that optimization problems come in different forms that are equivalent in the sense that a solution of one formulation can be easily constructed from the solution of a different formulation [24, §4.1.3] (precisely, they can be reduced to each other in linear time). Such changes may include, e.g., a reformulation of constraints or a change of variables. One such transformation was already shown in §1.1.2.

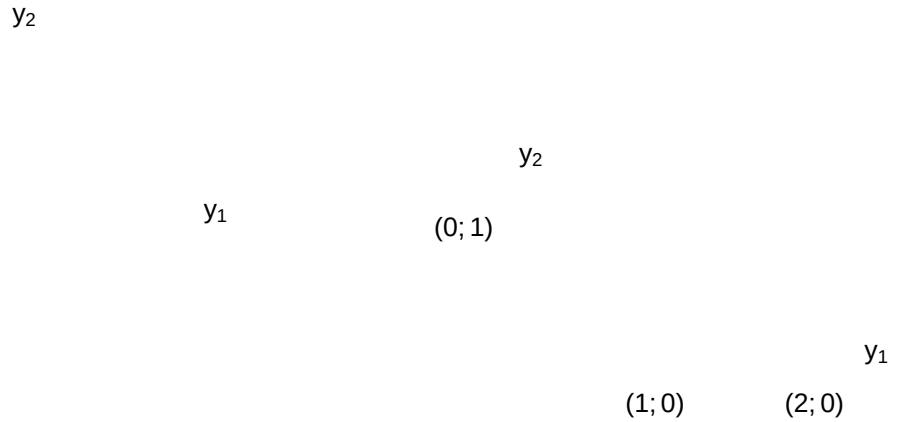
Although this was not explicitly mentioned in the literature before, it is easy to see that the quality of fixed points of BCD highly depends not only on the choice of blocks of variables, but also on the precise formulation of the optimization problem [51a, 53a]. This phenomenon is illustrated by the following examples.

Example 1.6 ([53a, Example 2]). The linear program $\min\{y_1 + y_2 \mid y_1, y_2 \geq 0\}$ has one ILM w.r.t. individual variables which coincides with the unique global minimum, namely $(y_1, y_2) = (0, 0)$. If the redundant constraint $y_1 = y_2$ is added to the linear program, then any feasible point becomes an ILM w.r.t. individual variables because the redundant constraint blocks changing the variable y_1 without changing y_2 and vice versa. ◁

Example 1.7. Let f and Y be as in Example 1.4 and $g : \mathbb{R}^2 \to \mathbb{R}^2$ be the bijection defined by $g(y_1, y_2) = (-y_1 + 2y_2, 2y_1 + y_2)$. After transforming the variables by g, we obtain $f(g(y_1, y_2)) = \max\{3y_1, 3y_2, 0\}$. Several contours of function $f \circ g$ are shown in Figure 1.3a.

Any point $y \in Y = \mathbb{R}^2$ with $y_1 = y_2 > 0$ is a local minimum of $f \circ g$ on Y w.r.t. individual variables, but no longer an ILM because the components of such points are not in the relative interior of coordinate-wise optimizers. In fact, any ILM of $f \circ g$ on Y w.r.t. individual variables is a global minimum.[10] ◁

[10] Optimizing $f \circ g$ coordinate-wise is in correspondence with optimizing f along subspaces $\operatorname{span}\{(1, 2)\}$ and $\operatorname{span}\{(2, 1)\}$ (recall Remark 1.3) which follows from the definition of g.



(a) Contours of function f g from Example 1.7. The set of optimizers is shaded.

(b) Feasible region Y^0 of the problem de ned in Example 1.8.

Figure 1.3: Illustrations to Examples 1.7 and 1.8.

Example 1.8 (cf. [150, §2, 151a, §2]). Let f and Y be as in Example 1.5. Adding the redundant constraint $y_1 + 2y_2 \quad 2$ reduces the feasible set to $Y^0 = f y \ 2 \ Y \ j \ y_1 + 2y_2 \quad 2g$, which is depicted in Figure 1.3b. This change preserves the set of global minima, but the set of pre-ILMs and ILMs is extended by a non-optimal point $(0; 1)$ where no coordinate-wise moves within the feasible set Y^0 are possible. 4

Example 1.9 (cf. [53a, Example 3]). Finally, we analyze the transformation from §1.1.2. Suppose that we minimize the unconstrained univariate convex piecewise-a ne function $f(x_1) = \max f x_1; 0g + \max f \quad 2x_1; \quad 2g$. When optimizing in the form (1.4), any ILM w.r.t. individual variables is trivially a global minimum. However, in the LP formulation (1.5), there are ILMs w.r.t. individual variables that are not globally optimal, such as $x_1 = z_1 = z_2 = 0$. 4

Fact 1.1 ([53a, Example 3]). Optimizing (1.4) coordinate-wise (i.e., applying BCD along individual variables) is in a precise sense equivalent to BCD applied to (1.5) along blocks of variables, each containing a single x_i variable and all the z variables. Formally, any ILM x of (1.4) w.r.t. individual variables yields an ILM $(x; z)$ of (1.5) w.r.t. the above-de ned blocks where z is de ned by (1.6). On the other hand, for any ILM $(x; z)$ of (1.5) w.r.t. the above-de ned blocks, x is an ILM of (1.4) w.r.t. individual variables. We will use this property multiple times in the sequel.

This was discussed for the special case of $jK j = 1$ in [150, §5, 151a, §2]. However, if $jK j = 1$, coordinate-wise minimization of (1.4) is equivalent to coordinate-wise minimization of (1.5) and one need not consider blocks of variables in (1.5).

1.3 Partially Ordered Sets

Let us review the basic concepts of order theory in this section. We begin by recalling the notion of a lattice, semilattice, and complete lattice. Then, we focus on the connection between complete lattices and closure operators. We conclude by analyzing iterative applications of isotone and intensive mappings. This part is based mainly on [22] and [42].

Let S be a set and $\sqsubseteq$ be a partial order on S, i.e., $\sqsubseteq$ is a binary relation on S that is reflexive, anti-symmetric, and transitive. Firstly, recall the duality principle in partially ordered sets [22, §1, 42, §1.20]: for any property that concerns the partially ordered set $(S; \sqsubseteq)$, there is a corresponding property that concerns its dual ordered set $(S; \sqsubseteq^{-1})$ where $\sqsubseteq^{-1}$ is the inverse order (a.k.a. dual order), i.e., $s_1 \sqsubseteq^{-1} s_2 \Leftrightarrow s_2 \sqsubseteq s_1$ for all $s_1, s_2 \in S$. The corresponding dual property is obtained by replacing all (both explicit and implicit) occurrences of $\sqsubseteq$ in the property by $\sqsubseteq^{-1}$.

Inspired by the notation in [22], for $Q \subseteq S$, we define

$$Q^{\sqsubseteq}_S = \{ s \in S \mid \forall q \in Q : q \sqsubseteq s \} \tag{1.21a}$$

$$Q^{\sqsupseteq}_S = \{ s \in S \mid \forall q \in Q : s \sqsubseteq q \} \tag{1.21b}$$

where $Q^{\sqsubseteq}_S$ and $Q^{\sqsupseteq}_S$ denote the set of all *upper bounds* and *lower bounds* on Q in S, respectively.

Furthermore, if $q \in Q^{\sqsubseteq}_S$ satisfies $q \sqsubseteq q'$ for all $q' \in Q^{\sqsubseteq}_S$, then it is called the *least upper bound* on Q in S and we denote it by $q = \bigvee_S Q$. Analogously, if $q \in Q^{\sqsupseteq}_S$ satisfies $q \sqsupseteq q'$ for all $q' \in Q^{\sqsupseteq}_S$, then it is called the *greatest lower bound* on Q in S and is denoted by $q = \bigwedge_S Q$. The operations $\bigvee_S$ and $\bigwedge_S$ are called the *join* and the *meet*, respectively. We emphasise that, in general, a partially ordered set need not have the least upper bound or greatest lower bound for each of its subsets.

For convenience, for two-element sets $Q = \{q_1, q_2\}$, we may write $q_1 \sqcup_S q_2$ instead of $\bigvee_S \{q_1, q_2\}$ and $q_1 \sqcap_S q_2$ instead of $\bigwedge_S \{q_1, q_2\}$.

In particular, we have that $S^{\sqsubseteq}_S$ is either empty or a singleton set because if $s_1, s_2 \in S^{\sqsubseteq}_S$, then, by definition (1.21a), $s_1 \sqsubseteq s_2$ and $s_2 \sqsubseteq s_1$, hence $s_1 = s_2$ by anti-symmetry. If $S^{\sqsubseteq}_S$ is non-empty, then $S^{\sqsubseteq}_S = \{\top\}$ where $\top = \bigwedge_S S \in S$ is the top element of S. By the duality principle, $S^{\sqsupseteq}_S$ is also either empty or a singleton set. If $S^{\sqsupseteq}_S = \{\bot\}$, then $\bot = \bigvee_S S \in S$ is the bottom element of S.[11] Again, a partially ordered set need not contain the top or the bottom element in general.

In the sequel, we simplify our notation as follows: if the set S w.r.t. which the upper bounds, lower bounds, joins, and meets are taken is clear from context, then we omit the subscript S, i.e., we simplify $Q^{\sqsubseteq}_S$, $Q^{\sqsupseteq}_S$, $\bigvee_S$, and $\bigwedge_S$ to $Q^{\sqsubseteq}$, $Q^{\sqsupseteq}$, $\bigvee$, and $\bigwedge$.

1.3.1 Lattices

Definition 1.4. A partially ordered set $(S; \sqsubseteq)$ is:

- a *meet-semilattice* if for any $s_1, s_2 \in S$, $s_1 \sqcap s_2$ exists in S,
- a *join-semilattice* if for any $s_1, s_2 \in S$, $s_1 \sqcup s_2$ exists in S,
- a *lattice* if it is a meet-semilattice and a join-semilattice,

[11] For the top and the bottom element, we adopt notation from [42, §1.21] to avoid using 0 and 1 which we reserve for their numerical meaning.

ˆ a complete lattice if for any $Q \subseteq S$, both $\bigsqcup Q$ and $\bigsqcap Q$ exist in S.

It is easily shown by induction [42, §2.11] that for a meet-semilattice $(S; \sqsubseteq)$ and any non-empty finite $Q \subseteq S$, $\bigsqcap Q$ exists in S. Consequently, any non-empty finite meet-semilattice $(S; \sqsubseteq)$ has a bottom element, namely $\bigsqcap S$. By the duality principle, for any join-semilattice $(S; \sqsubseteq)$ and any non-empty finite $Q \subseteq S$, $\bigsqcup Q$ exists in S. In particular, any non-empty finite join-semilattice $(S; \sqsubseteq)$ has a top element, $\bigsqcup S$. A complete lattice always has both the top and the bottom element [22, Theorem 2.10].

Lemma 1.1 ([106, Theorem 2.5, 22, Theorem 2.11, 42, Theorem 2.31]). Let $(S; \sqsubseteq)$ be a partially ordered set. The following are equivalent:

(a) $\bigsqcup Q$ exists in S for any non-empty $Q \subseteq S$ and $(S; \sqsubseteq)$ has a top element $\top$,

(b) $\bigsqcap Q$ exists in S for any non-empty $Q \subseteq S$ and $(S; \sqsubseteq)$ has a bottom element $\bot$,

(c) $(S; \sqsubseteq)$ is a complete lattice.

Proof. The implication (c) $\Rightarrow$ (a) is clear, so we proceed to prove (a) $\Rightarrow$ (c). Since $(S; \sqsubseteq)$ has top element $\top$, we have that $\bigsqcup \emptyset = \top$. Consequently, for any $Q \subseteq S$, $\bigsqcup Q$ exists in S.

Now, it remains to show that $\bigsqcap Q$ exists in S for any $Q \subseteq S$. We define $q = \bigsqcup Q^{\sqsubseteq}$ and claim that $q = \bigsqcap Q$. First, notice that $q_1 \sqsubseteq q_2$ for all $q_1 \in Q$ and all $q_2 \in Q^{\sqsubseteq}$ (by definition of an upper bound). Consequently, $q_1 \sqsubseteq \bigsqcup Q^{\sqsubseteq} = q$ holds for all $q_1 \in Q$, so q is an upper bound on Q. By definition of $\bigsqcup$, q is the least upper bound on Q in S.

The equivalence (b) $\Leftrightarrow$ (c) is obtained dually. □

Theorem 1.11 ([106, Theorem 2.4, 42, Corollary 2.25]). Let $(S; \sqsubseteq)$ be a non-empty finite partially ordered set. The following are equivalent:

(a) $(S; \sqsubseteq)$ is a meet-semilattice with top element $\top$,

(b) $(S; \sqsubseteq)$ is a join-semilattice with bottom element $\bot$,

(c) $(S; \sqsubseteq)$ is a complete lattice,

(d) $(S; \sqsubseteq)$ is a lattice.

Proof. Together with the previously stated facts, Lemma 1.1 yields (a) $\Leftrightarrow$ (b) $\Leftrightarrow$ (c). The implication (c) $\Rightarrow$ (d) is clear by definition of a complete lattice. To prove (d) $\Rightarrow$ (a), notice that any finite lattice has the top element $\bigsqcup S$ and is also a meet-semilattice. □

By Theorem 1.11, it is possible to augment any finite meet-semilattice by introducing a new artificial top element to obtain a complete lattice. Dually, one can include a bottom element in a finite join-semilattice which is known as the lifting operation [42, §1.22].

1.3.2 (Dual) Closure Operators and Chaotic Iterations

Let us now recall the connection between closure operators and complete lattices. In detail, any closure operator defined on a complete lattice defines its subset which is also a complete lattice and, under additional assumptions, such a subset defines a closure operator.

Before we state these results formally, we overview useful properties of mappings on partially ordered sets in Definition 1.5.

De nition 1.5 ([22, §1.4]). Let $(S; \leq)$ be a partially ordered set. Mapping $f : S \to S$ is

- extensive if $\forall s \in S: s \leq f(s)$,
- intensive if $\forall s \in S: f(s) \leq s$,
- idempotent if $\forall s \in S: f(s) = f(f(s))$,
- isotone if $\forall s_1, s_2 \in S: s_1 \leq s_2 \Rightarrow f(s_1) \leq f(s_2)$,
- a closure operator if it is extensive, idempotent, and isotone,
- a dual closure operator if it is intensive, idempotent, and isotone.

Let us denote the image of a mapping $f : S \to S$ by

$$\operatorname{im} f = \{f(s) \mid s \in S\}. \tag{1.22}$$

Notice that it follows directly from De nition 1.5 that the set of xed points of an idempotent mapping f coincides with its image, i.e., $\{s \in S \mid f(s) = s\} = \operatorname{im} f$. In detail, inclusion in the direction follows from idempotence and the other direction is clear from the de nition of a xed point [22, §1.4]. In particular, this holds for any (dual) closure operator f.

For the purpose of the following theorem, we need to present an auxiliary lemma and also introduce additional terminology: for complete lattices $(S; \leq)$ and $(Q; \sqsubseteq)$ with $Q \subseteq S$, we say that their meet operations coincide if for any $Q' \subseteq Q$, we have $\bigwedge_S Q' = \bigwedge_Q Q'$. In words, this is the case if for any $Q' \subseteq Q$, the greatest lower bound on Q' in S is the same as the greatest lower bound on Q' in Q. Analogously, their join operations coincide if for any $Q' \subseteq Q$, we have $\bigvee_S Q' = \bigvee_Q Q'$.

Lemma 1.2 ([42, Lemma 2.22(v)]). Let $(S; \leq)$ be a complete lattice and $S_1 \subseteq S_2 \subseteq S$. Then, $\bigwedge S_1 \geq \bigwedge S_2$ and $\bigvee S_2 \geq \bigvee S_1$.

Proof. The rst claim follows from the fact that $\bigwedge S_2 = \bigwedge S_1 \wedge \bigwedge (S_2 \setminus S_1)$ is a lower bound on $\bigwedge S_1$. The second claim follows from the duality principle. □

Theorem 1.12 (cf. [42, Theorem 7.3]) Let $(S; \leq)$ and $(Q; \sqsubseteq)$ be complete lattices such that $Q \subseteq S$.

(a) If the meet operations in $(S; \leq)$ and $(Q; \sqsubseteq)$ coincide, then the mapping $f : S \to Q$ de ned by $f(s) = \bigwedge \{s\}^{\uparrow}_Q$ is a closure operator.

(b) If the join operations in $(S; \leq)$ and $(Q; \sqsubseteq)$ coincide, then the mapping $g : S \to Q$ de ned by $g(s) = \bigvee \{s\}^{\downarrow}_Q$ is a dual closure operator.

Proof. We show only (a) since (b) then follows from the duality principle. Notice that we do not need to distinguish $\bigwedge_S$ and $\bigwedge_Q$ in the de nition of f because $\{s\}^{\uparrow}_Q$ is a subset of Q and the meet operations coincide.

We begin by extensivity: for $s \in S$, $s \in \{s\}^{\uparrow}_S$ holds by re exivity of $\leq$ and also $\{s\}^{\uparrow}_Q \subseteq \{s\}^{\uparrow}_S$ due to $Q \subseteq S$. Lemma 1.2 yields $f(s) = \bigwedge \{s\}^{\uparrow}_Q \geq \bigwedge \{s\}^{\uparrow}_S = s$.

For isotony, let $s_1, s_2 \in S$ such that $s_1 \leq s_2$. Then, $\{s_1\}^{\uparrow}_Q \supseteq \{s_2\}^{\uparrow}_Q$ by transitivity of $\leq$ and we obtain $f(s_1) = \bigwedge \{s_1\}^{\uparrow}_Q \leq \bigwedge \{s_2\}^{\uparrow}_Q = f(s_2)$ by Lemma 1.2.

Finally, to prove idempotency, we have that $f(s) \in \{f(s)\}^{\uparrow}_Q$, which yields $f(f(s)) = \bigwedge \{f(s)\}^{\uparrow}_Q = f(s)$. □

```
   inputs: partially ordered set (S; ≤), initial element s ∈ S, isotone and intensive
           mappings f_1, ..., f_n : S → S.
1  s' := s
2  while ∃i ∈ [n] : f_i(s') ≠ s' do
3  |   Find such i.
4  |   s' := f_i(s')
5  return s'
```

Algorithm 1.1: Iterations of a given set of mappings applied to an initial element $s \in S$ of a partially ordered set $(S; \leq)$.

Focusing on the definition of f and g in Theorem 1.12, for any $s \in S$, we have that $f(s)$ is the least upper bound on s in Q and $g(s)$ is the greatest lower bound on s in Q. Theorem 1.12 has the following practical corollary.

Corollary 1.4. Let $(S; \leq)$ and $(Q; \sqsubseteq)$ be complete lattices with $Q \subseteq S$. Define the mappings $f, g : S \to Q$ by $f(s) = \bigsqcap_Q \{s\}^{\sqsubseteq}_Q$ and $g(s) = \bigsqcup_Q \{s\}^{\sqsubseteq}_Q$. For any $s \in S$, it holds that $s \in Q \Leftrightarrow f(s) = s \Leftrightarrow g(s) = s$.

Proof. If $s \in Q$, $s \in \{s\}^{\sqsubseteq}_Q$ and $s \leq q$ holds for any $q \in \{s\}^{\sqsubseteq}_Q$ by definition, so $s = \bigsqcap_Q \{s\}^{\sqsubseteq}_Q = f(s)$. If $s \notin Q$, $f(s) \neq s$ due to $\operatorname{im} f = Q$. The part with g is obtained dually. □

Remark 1.4. The converse connection between (dual) closure operators and complete lattices also holds. In detail, for any complete lattice $(S; \leq)$ and any (dual) closure operator $f : S \to S$, $(\operatorname{im} f; \leq)$ is a complete lattice [22, Theorem 2.14, 42, §7].

Finally, suppose that we are given a set of isotone and intensive mappings on some finite partially ordered set $(S; \leq)$ and an element $s \in S$. To find the greatest common fixed point s' of these mappings such that $s' \leq s$, one can use Algorithm 1.1 whose correctness is given by Theorem 1.13.

Theorem 1.13 (cf. [6, Theorem 1]). Let $(S; \leq)$ be a finite partially ordered set, $s \in S$, and $f_1, \ldots, f_n : S \to S$ be isotone and intensive mappings. Algorithm 1.1 terminates in a finite number of steps and returns the greatest common fixed point s' of mappings $f_1, \ldots, f_n$ that satisfies $s' \leq s$.

Proof. In each iteration of the algorithm, the current s' strictly decreases w.r.t. $\leq$ by intensivity of f_i. This implies that $s' \leq s$ and, by finiteness of S, that the algorithm terminates after a finite number of iterations. The fact that s' is a common fixed point of all the mappings f_i follows directly from the condition on line 2 of the algorithm.

To prove that s' is the greatest common fixed point, we proceed by induction. In detail, we show that $s \leq s'$ holds during the whole run of the algorithm for any common fixed point $s \in S$ such that $s \leq s$. The base case is clear: $s \leq s = s'$. For the inductive step, it follows from isotony of f_i that $s = f_i(s) \leq f_i(s')$ where the equality is given by s being a common fixed point. □

Note that, by Theorem 1.13, the value returned by Algorithm 1.1 is independent of the way of choosing i on line 3. Similar algorithms are known as chaotic iterations [5, §2].

1.4 Constraint Satisfaction Problem and Local Consistencies

In this section, we recall the constraint satisfaction problem (CSP), the notion of local consistency, and constraint propagation. Furthermore, we also revisit the connection between local consistencies, closure operators, and iterations of propagators. Throughout the thesis, we follow our notation for CSPs from [55a, 56a] and we reuse some parts of these papers to define the notation and well-known terms in the beginning of this section.

The structure of a CSP is defined by a triple $(V; D; C)$ where V is a finite set of variables, D is a common finite domain of each variable, and $C \subseteq 2^V$ is a non-empty set of non-empty scopes of constraints so that $(V; C)$ can be interpreted as a hypergraph. The structure $(V; D; C)$ gives rise to the set of all tuples

$$T = \{ (S; k) \mid S \in C,\ k \in D^S \}; \tag{1.23}$$

which can be naturally partitioned into the sets

$$T_S = \{ (S; k) \mid k \in D^S \} \tag{1.24}$$

for each $S \in C$.

In machine learning, hypergraphs are often equivalently represented by so-called factor graphs which is convenient as it allows us to avoid the more complex hypergraph terminology. In analogy to [83], the factor graph of a hypergraph $(V; C)$ (or, factor graph of a CSP with a set of variables V and constraint scopes C) is the bipartite graph whose nodes correspond to variables in V and scopes in C. The factor graph contains an edge between a variable node $i \in V$ and a scope node $S \in C$ if $i \in S$.[12] We show an example of a factor graph in Figure 1.4.

Even though most of the results in the thesis do not rely on this, we will for simplicity of presentation generally assume that $\{i\} \in C$ for each $i \in V$ and that the factor graph of $(V; C)$ is connected (unless specified otherwise). We also implicitly assume that there is at most one constraint with each scope because C is a set (instead of a multiset).

An instance of the constraint satisfaction problem (CSP) is defined by the quadruple $(V; D; C; A)$ where $(V; D; C)$ is the structure and $A \subseteq T$ is the set of allowed tuples (while the tuples $T \setminus A$ are forbidden). In the sequel, we will not need to change the structure $(V; D; C)$ of the CSP, so we will refer to CSP instances only as A for brevity. In other words, we identify CSP instances with subsets of T.

In analogy to §1.2, for an assignment[13] $x \in D^V$ and $S \subseteq V$, $x_{|S}$ stands for the restriction of x onto the set S, i.e., for $S = \{i_1, \ldots, i_{|S|}\}$ we have $x_{|S} = (x_{i_1}, \ldots, x_{i_{|S|}}) \in D^S$ (where the order of the components is defined by the total order on S inherited from some arbitrary fixed total order on V). An assignment $x \in D^V$ uses a tuple $(S; k) \in T$ if $x_{|S} = k$. An assignment $x \in D^V$ is a solution to CSP A if it uses only allowed tuples,

[12] The factor graph is isomorphic to the primal constraint graph [46, §2.1.3] of the hidden transformation [8, Definition 7] of the CSP. Also recall that a sequence $(i_1, S_1, i_2, S_2, \ldots, S_n, i_{n+1})$ with $n \geq 2$ and $i_1 = i_{n+1}$ is a Berge cycle [14, §17] of a hypergraph $(V; C)$ if $i_1, \ldots, i_n$ are distinct vertices from V, $S_1, \ldots, S_n$ are distinct scopes from C, and $i_k, i_{k+1} \in S_k$ holds for all $k \in [n]$. A hypergraph is Berge-acyclic if it does not contain a Berge cycle. It is easy to see that the factor graph of $(V; C)$ is acyclic if and only if the hypergraph $(V; C)$ is Berge-acyclic.

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

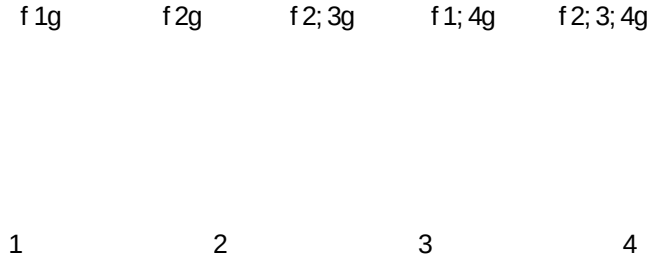


Figure 1.4: Factor graph of the hypergraph $(V; C)$ where $V = f 1; 2; 3; 4g$ and $C = ff 1g; f 2g; f 2; 3g; f 1; 4g; f 2; 3; 4gg$. The factor graph contains 5 nodes corresponding to elements of C (shown as rectangles), 4 nodes corresponding to elements of V (shown as circles), and 9 edges in total.

i.e., $S; x_{j_S} \; 2 \; A$ for all $S \; 2 \; C$. Note that each assignment $x \; 2 \; D^V$ uses exactly one tuple from each T_S, $S \; 2 \; C$.

As usual [5, 17, 81, 46, 67], the solution set of a CSP A is denoted by $SOL(A) \quad D^V$. We say that CSPs $A; A^0 \quad T$ are equivalent if $SOL(A) = SOL(A^0)$ [5, 6, 105]. CSP A is satis able if $SOL(A) \; 6 \; ;$ and unsatis able otherwise. The problem of deciding whether a CSP is satis able is NP-complete (e.g., by reduction from 3-coloring [116, §8.6.1, 84]).

Note that one can interpret SOL as a mapping $SOL : 2^T \; ! \; 2^{(D^V)}$. Since allowing more tuples does not make the solution set smaller, the mapping SOL is isotone, i.e., $A \quad A^0$ implies $SOL(A) \quad SOL(A^0)$.

To avoid any ambiguity, we de ne a CSP to be Boolean if $jDj = 2$. On the other hand, a CSP is pairwise if $jSj \quad 2$ for all $S \; 2 \; C$.[14] Finally, we de ne $C_{\; 2} = f S \; 2 \; C \; j \; jSj \quad 2g$ to be the set of all non-unary scopes.

Example 1.10 (cf. [56a, Example 2]). Let $V = f 1; 2g$, $D = f a; bg$, and $C = ff 1g; f 2g; f 1; 2gg$. For this structure, the set of tuples is

$$\begin{aligned} T = f & (f 1g; a); \; (f 1g; b); \; (f 2g; a); \; (f 2g; b); \\ & (f 1; 2g; (a; a)); \; (f 1; 2g; (a; b)); \; (f 1; 2g; (b; a)); \; (f 1; 2g; (b; b)) g; \end{aligned} \tag{1.25}$$

i.e., $T = T_{f 1g} [\; T_{f 2g} [\; T_{f 1;2g}$ where, e.g., $T_{f 2g} = f (f 2g; a); (f 2g; b) g$.

The assignment $x = (a; b) \; 2 \; D^V$ (i.e., $x_1 = a$, $x_2 = b$) uses tuples $(f 1g; a)$, $(f 2g; b)$, and $(f 1; 2g; (a; b))$. CSP A_1 from Figure 1.5a is de ned by

$$A_1 = f \; (f 1g; a); \; (f 1g; b); \; (f 2g; b); \; (f 1; 2g; (a; b)); \; (f 1; 2g; (b; b)) \; g \quad T \tag{1.26}$$

and is equivalent to A_2 in Figure 1.5b since $SOL(A_1) = SOL(A_2) = f (a; b); (b; b) g$.

CSPs A_3 and A_4 in Figures 1.5c and 1.5d are both unsatis able (and therefore equivalent). Since the structure of all these CSPs is the same, they are all Boolean and pairwise. 4

[14] We intentionally avoid the term `binary' that sometimes refers to `Boolean' whereas in other works it means `pairwise'.

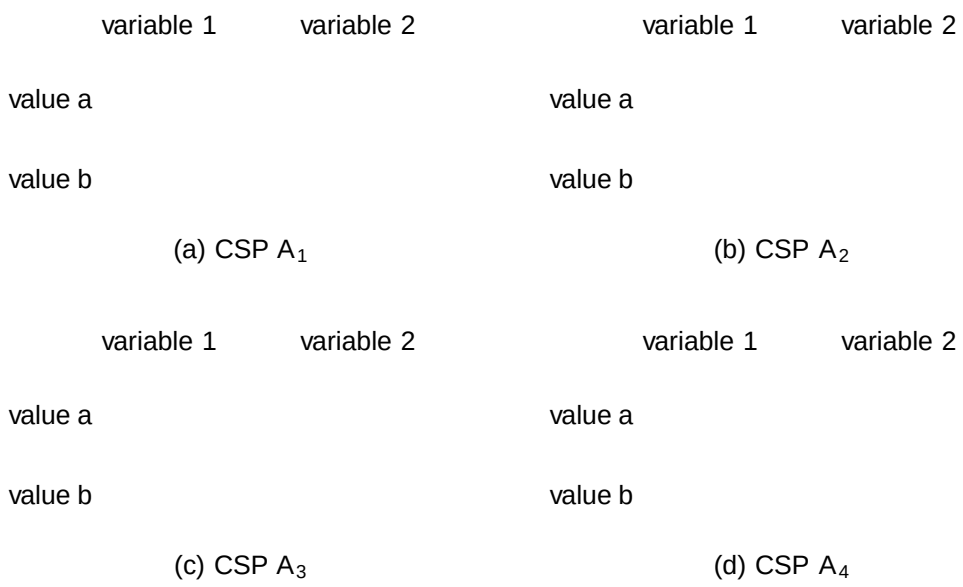


Figure 1.5: Visualisations of four CSPs with the same structure. Variables (elements of V) are depicted as rounded rectangles, tuples (elements of T) as circles and line segments. Black circles and full lines indicate allowed tuples, whereas white nodes and dashed lines indicate forbidden tuples. Our visualisation convention for (W)CSPs is similar to the ones considered in [149, 146, 134, 119, 92, 46][15] for CSPs, this in fact depicts its micro-structure [82].

1.4.1 Local Consistencies and Constraint Propagation

Let us now focus on the connection of local consistencies, consistency closures, and propagators. Local consistencies are a key concept in the constraint programming community that is typically used to prune the search space when searching for a solution of a CSP. The topic of local consistencies is broad and well-studied [17, 46, §3, 99, §3-§7].

A *local consistency* is a (usually simple) condition that is necessary for allowed tuples of a CSP to be used by (some of) its solutions. Enforcing a local consistency means finding an equivalent CSP that satisfies the given condition. For this, constraint propagation is used. In general, constraint propagation explicitly infers some knowledge that was only implicit before. This could mean, e.g., forbidding tuples that are not used by any solution or introducing new constraints that are satisfied by all solutions. In the sequel, we focus only on the former case, i.e., reducing the constraints while maintaining equivalence which is known as constraint reduction process [5, §3.2]. In other words, we develop formalism only for a subset of local consistencies that preserve the structure of a CSP and are applied to constraints that are given in extension (i.e., by enumerating the allowed tuples).

In this part, we first formally define the properties of local consistencies sufficient for defining the associated (dual) closure operators. Then, we point our attention to how this closure can be obtained using a set of propagators and exemplify the shown notions on arc consistency. Finally, we discuss that some classes of CSPs can be solved by enforcing local consistencies without any search. We base our description mainly on [5] and [17].

[15] On the other hand, our convention is in contrast to some other papers, such as [136, 107, 98, 43, 33, 70] where forbidden non-unary tuples are emphasised whereas we emphasise the allowed tuples.

Remark 1.5. In some formalisms, e.g., [17, 5, 8, 99], the domains of the individual variables are explicitly reduced during constraint propagation, i.e., the set D is different for each variable and is gradually made smaller. However, we do not reduce the domains of individual variables explicitly in our notation and set D is kept unchanged. Instead, we expect that there is a unary constraint on each variable and removing value $k \in D$ from the domain of variable $i \in V$ is performed by forbidding the tuple (i, k). Both approaches are semantically equivalent [17, §3.1].

To be more formal, consider a local consistency and the following properties.

Property 1.1 ([17]). If CSPs $\mathcal{A}$ and $\mathcal{A}'$ are -consistent, $\mathcal{A} \times \mathcal{A}'$ is also -consistent.

Property 1.2 ([17]). CSP $\langle \emptyset, \emptyset \rangle$ is -consistent.

Property 1.3. If a CSP $\mathcal{A}$ satisfies that for all $(S, k) \in \mathcal{A}$ there exists $x \in \mathrm{SOL}(\mathcal{A})$ such that $x_{|S} = k$, then $\mathcal{A}$ is -consistent.

Property 1.1 is called stability under union (cf. [17, Definition 3.17]) and is satisfied by typical local consistencies[16]. Property 1.2 can be assumed by convention, as stated in [17, Theorem 3.19]. Property 1.3 is a formalization of the statement that is a necessary condition for the allowed tuples to be extendable to a solution, i.e., we should not forbid tuples if such an action changes the solution set. Note, Property 1.3 implies Property 1.2.[17] For now, we analyze the implications of the first two properties and we will focus on the importance of Property 1.3 later.

For a local consistency satisfying Properties 1.1 and 1.2, the set of all -consistent CSPs (with the fixed structure (V, D, C)) equipped by the partial order given by the set inclusion is by Theorem 1.11 a complete lattice since it contains the bottom element $\mathcal{T}$ and its join operation is the set union. By Theorem 1.12b[18], this gives rise to a dual closure operator $\mathrm{C} : 2^{\mathcal{T}} \to 2^{\mathcal{T}}$ defined by

$$\mathrm{C}(\mathcal{A}) = \bigcup \{ \mathcal{A}' \mid \mathcal{A}' \subseteq \mathcal{A} \text{ and } \mathcal{A}' \text{ is -consistent} \} \tag{1.27}$$

which is the greatest -consistent CSP that is a subset of $\mathcal{A}$. In addition, Corollary 1.4 yields that CSP $\mathcal{A}$ is -consistent if and only if $\mathrm{C}(\mathcal{A}) = \mathcal{A}$.

Remark 1.6. Although (1.27) is sometimes only referred to as a closure (e.g., as in arc consistency closure [8, 33, 17]), it is indeed a dual closure in the sense of §1.3.2 because it is intensive, i.e., it reduces the set of allowed tuples. This distinction is only technical as it can be easily corrected by either considering the dual setting (as in [5]) where the ordering is formally reversed, or by defining the CSP by the set of forbidden (rather than allowed) tuples. To be consistent with the literature, we will sometimes omit `dual' and say, e.g., `arc consistency closure' instead of `arc consistency dual closure'.

[16] For an example of a local consistency that is not stable under union, see [17, Example 3.18].

[17] On the other hand, Property 1.3 is not implied by Properties 1.1 and 1.2. As an example, if $\langle \emptyset, \emptyset \rangle$ is the only -consistent CSP, then this notion of -consistency satisfies Properties 1.1 and 1.2 but does not satisfy Property 1.3.

[18] In more detail, the set of -consistent CSPs is a subset of $2^{\mathcal{T}}$ which is also partially ordered by the set inclusion and $(2^{\mathcal{T}}, \subseteq)$ is a complete lattice. Both of these complete lattices have the same join operation, namely the set union $\bigcup$.

We emphasise that, for the partially ordered set of -consistent CSPs to be a complete lattice, we required only Properties 1.1 and 1.2, not Property 1.3. The same holds for the existence of the dual closure operator C. However, we will show that if a local consistency does not satisfy Property 1.3, then the corresponding dual closure operator C may not return an equivalent CSP.

To concisely describe this result, let us recall positive consistency [7, §III, 20, Definition 3] (related terms are minimal network and minimal CSP from [105, §3, 46, §2.3.2, 101, §7.4, 57, Definition 2, 67, §1.1]).

Definition 1.6 ([7, 20]). A CSP A is positively consistent if for all $(S, k) \in A$, there is $x \in \mathrm{SOL}(A)$ such that $x_{|S} = k$.

It is easy to see that forbidding any tuple in a positively consistent CSP changes its solution set. This new notion allows us to restate Property 1.3 equivalently as follows: any positively consistent CSP is also -consistent. Clearly, positive consistency is stable under union (i.e., satisfies Property 1.1) and $\mathrm{CSP}(\emptyset, \emptyset)$ is positively consistent, hence the corresponding dual closure operator (1.27) is defined and will be denoted by C_{pos}. The following proposition states an expected property of the mapping C_{pos}.

Proposition 1.1. Let $A \subseteq T$. $\mathrm{SOL}(A) = \mathrm{SOL}(C_{\mathrm{pos}}(A))$.

Proof. Let $A' = A \cup \{(S, k) \in T \mid \forall x \in \mathrm{SOL}(A) : x_{|S} \neq k\}$. $\mathrm{CSP}(A')$ is clearly positively consistent and $A \subseteq A'$, so $A' \subseteq C_{\mathrm{pos}}(A) \subseteq A$. Moreover, $\mathrm{SOL}(A') = \mathrm{SOL}(A)$ holds by definition of A'. Applying isotony of SOL results in $\mathrm{SOL}(A') \supseteq \mathrm{SOL}(C_{\mathrm{pos}}(A)) \supseteq \mathrm{SOL}(A)$, hence $\mathrm{SOL}(C_{\mathrm{pos}}(A)) = \mathrm{SOL}(A)$. □

Using Proposition 1.1, we are now in position to formalize the importance of Property 1.3 in Theorem 1.14.

Theorem 1.14. Let be a local consistency satisfying Properties 1.1 and 1.2. The following are equivalent:

(a) -consistency satisfies Property 1.3,

(b) $\forall A \subseteq T : \mathrm{SOL}(C(A)) = \mathrm{SOL}(A)$.

Proof. (a) $\Rightarrow$ (b): Since $C_{\mathrm{pos}}(A)$ is -consistent and $C_{\mathrm{pos}}(A) \subseteq A$, it follows that $C_{\mathrm{pos}}(A) \subseteq C(A)$ by definition of C in (1.27). Proposition 1.1 yields $\mathrm{SOL}(C_{\mathrm{pos}}(A)) = \mathrm{SOL}(A)$. Combining this with $C_{\mathrm{pos}}(A) \subseteq C(A) \subseteq A$ (where we used intensivity of C) and isotony of SOL, we obtain $\mathrm{SOL}(C(A)) = \mathrm{SOL}(A)$ analogously to the proof of Proposition 1.1.

(b) $\Rightarrow$ (a): By contrapositive: let Property 1.3 not be satisfied and let A be the positively consistent CSP that is not -consistent. Since A is not -consistent, we have $C(A) \neq A$ and thus $C(A) \subset A$ by intensivity of C. This implies $\mathrm{SOL}(C(A)) \supset \mathrm{SOL}(A)$ because forbidding any tuple in a positively consistent CSP changes its solution set. □

To summarize, if a local consistency satisfies Properties 1.1 and 1.3 (and thus also satisfies Property 1.2), then C is a dual closure operator and for any $\mathrm{CSP}(A)$, $C(A)$ and A are equivalent, i.e., $\mathrm{SOL}(A) = \mathrm{SOL}(C(A))$. In the following parts, we will assume that satisfies these properties.

```
   inputs:  CSP A, local consistency .
1  A^0 := A
2  while A^0 is not -consistent do
3  |  Find R  A^0, such that R 6= ; and SOL(A^0  R) = SOL( A^0).
4  |  A^0 := A^0  R
5  return  A^0
```

Algorithm 1.2: Propagation algorithm enforcing -consistency of CSP A.

1.4.1.1 Propagation Algorithm and Propagators

In practice, the dual closure (1.27) is not computed in a single step, but instead by iteratively applying multiple propagators [17, §3.7]. Typically, there is a propagation algorithm that gradually forbids some tuples that are identified to be inconsistent (which implies that they are not used by any solution). After these tuples are forbidden, the algorithm may detect that other tuples became inconsistent and thus, forbids them too. The algorithm eventually stops when it is unable to forbid any other tuples, i.e., the instance satisfies the local consistency condition.

This is the principle of constraint propagation applied to a CSP that we (for now slightly informally) outline in Algorithm 1.2. The input of the propagation algorithm is a CSP A and a local consistency . First, the algorithm initializes $A^0 := A$ and then, in each iteration, the algorithm finds a non-empty subset R of allowed tuples in A^0 that were identified not to be used by any solution of the CSP by the local consistency and forbids them.[19] Note that such a subset always exists by Property 1.3: if a CSP is not -consistent, then there exists at least one allowed tuple that is not used by any solution. Such updates are repeated until A^0 becomes -consistent. Note that the returned CSP A^0 is equivalent to the input CSP A. In particular, if CSP A^0 is empty, then the input CSP A is unsatisfiable.

We will now show an alternative formalism for enforcing a local consistency that is based on propagators. Let p_i, $i \in P$ be propagators indexed by a finite set P, i.e., for all $i \in P$, $p_i : 2^T \to 2^T$ is an isotone and intensive mapping such that $SOL(A) = SOL(p_i(A))$ holds for all $A \subseteq T$.[20] Suppose that we repeatedly apply these propagators to an input CSP, as outlined in Algorithm 1.3. Clearly, this algorithm is an instantiation of Algorithm 1.1, so, by Theorem 1.13, the output of Algorithm 1.3 is the greatest CSP (w.r.t.) that is a subset of A and a fixed point of all propagators p_i, $i \in P$.

Typically [6], the set of propagators is defined so that their common fixed points characterize the desired local consistency , i.e., for all $A \subseteq T$, we have that

$$A \text{ is } \text{-consistent} \iff \forall\, i \in P : p_i(A) = A. \tag{1.28}$$

In such case, Algorithm 1.3 computes $C_\ (A)$ and is a more formal version of the previously shown Algorithm 1.2. In detail, the conditions on line 2 of both algorithms become equivalent due to (1.28). Also, the task of finding a non-empty subset R of -inconsistent tuples

[19] As stated in [99, §3.2], usual local consistencies are nogood-identifying, i.e., whenever a CSP is not -consistent, it is because at least one allowed tuple is found not to be used by any solution of the CSP by the local consistency . Such tuples are called -inconsistent [99, Definition 3.12].

[20] Up to the requirement of isotony, this is a constraint reduction function [5, Definition 3.5].

```
   inputs: CSP A, set of propagators p_i, i ∈ P, i.e., each p_i : 2^T → 2^T is an isotone
           and intensive mapping satisfying ∀A' ⊆ T : SOL(A') = SOL(p_i(A')).
1  A' := A
2  while ∃i ∈ P : p_i(A') ≠ A' do
3  |   Find such i.
4  |   A' := p_i(A')
5  return A'
```

Algorithm 1.3: Propagation algorithm based on application of individual propagators to the input CSP.

from A' boils down to finding $i \in P$ such that $p_i(A) \subset A$. If such a propagator p_i exists, the set $R = A' \setminus p_i(A')$ satisfies the required conditions stated on line 3 of Algorithm 1.2.

1.4.1.2 Example: Arc Consistency

We now exemplify the previously discussed concepts on arc consistency (AC). Recall that a CSP A is (generalized[21]) arc consistent [149, §4.1, 146, §3, 119, §6.2.2] if

$$(\{i\}, k) \in A \implies \exists (S, \ell) \in A : \ell_i = k \tag{1.29}$$

holds for all $S \in C_2$, $i \in S$, and $k \in D$. It is readily verified that AC satisfies Properties 1.1 and 1.3.

Remark 1.7. In constraint programming [17, 141, 8, 101, 46, 99], a more common definition requires

$$(\{i\}, k) \in A \implies \exists (S, \ell) \in A : (\ell_i = k \wedge \forall j \in S : (\{j\}, \ell_j) \in A) \tag{1.30}$$

instead of (1.29). To enforce AC in this sense, it is not necessary to forbid tuples corresponding to non-unary constraints. Both definitions are equivalent in terms of the forbidden unary tuples (i.e., the reduced domains) and have the same strength. In detail, any CSP that is arc consistent in the sense of (1.29) is also arc consistent in the sense of (1.30) (e.g., CSP in Figure 1.5a). Moreover, it can be easily shown that for any CSP A that is arc consistent in the sense of (1.30) (e.g., CSP in Figure 1.5b), the CSP

$$A' = A \setminus \{(S, k) \in A \mid S \in C_2, \exists i \in S : (\{i\}, k_i) \notin A\} \tag{1.31}$$

is arc consistent in the sense of (1.29). Note that we only forbid some tuples of the non-unary constraints in (1.31).

To obtain the AC closure, one uses AC propagators. Formally, for $S \in C_2$, $i \in S$, and $k \in D$, we define the propagator $p_{S,i,k} : 2^T \to 2^T$ by

$$p_{S,i,k}(A) = \begin{cases} A & \text{if (1.29) is satisfied} \\ A \setminus \{(\{i\}, k)\} & \text{if (1.29) is not satisfied and } (\{i\}, k) \in A : \\ A \setminus \{(S, \ell) \in T_S \mid \ell_i = k\} & \text{if (1.29) is not satisfied and } (\{i\}, k) \notin A \end{cases} \tag{1.32}$$

[21] Sometimes [17, §3.3, 33] (but not always), arc consistency for CSPs with higher-order constraints is called generalized arc consistency. We do not use this name in the thesis and simply call the local consistency `arc consistency'.

Clearly, $A \equiv T$ is arc consistent if and only if $p_{S;i;k}(A) = A$ holds for all such triples $(S; i; k)$. Moreover, the propagators always return an equivalent CSP and are intensive, isotone, and even idempotent.

Also, for any $S \in C_2$, $i \in S$, $k \in D$, and $A \equiv T$, it is easy to verify that the CSP $p_{S;i;k}(A)$ satisfies (1.29) for this triple $(S; i; k)$. By repeated application of different propagators $p_{S;i;k}$, as in Algorithm 1.3, more and more tuples become forbidden. Eventually, when $p_{S;i;k}(A) = A$ holds for all triples $(S; i; k)$, CSP A is arc consistent and this result coincides with the AC closure of the initial CSP.

1.4.1.3 CSPs Solved by Enforcing Local Consistencies

In general, a local consistency is neither a necessary nor a sufficient condition of satisfiability [5, 8] – e.g., an arc consistent CSP need not be satisfiable, but a CSP that is not arc consistent may be satisfiable.

For a local consistency satisfying the properties from §1.4.1 and $A \equiv T$, if $C(A)$ is empty, then CSP A is unsatisfiable because an empty CSP is unsatisfiable and $\mathrm{SOL}(A) = \mathrm{SOL}(C(A))$. On the other hand, non-emptiness of $C(A)$ does not in general imply that A is satisfiable.

However, for some CSP instances, enforcing a local consistency is sufficient to decide whether they are satisfiable (or even find a solution). Formally, we say that a local consistency is refutation complete for a class of CSPs $\mathcal{A}$ if for any $A \in \mathcal{A}$ with non-empty $C(A)$, A is satisfiable.[22] Otherwise, we say that the local consistency is refutation incomplete (for some class of CSPs).

Example 1.11. Restricting ourselves to pairwise CSPs, it is well known [61] that AC is refutation complete for CSPs where the graph $(V; C_2)$ is a tree (or, more generally, a forest). For CSPs with higher-order constraints, AC is refutation complete if the factor graph of this CSP is acyclic (i.e., if the hypergraph is Berge-acyclic) and this is the only structural restriction where AC is refutation complete [31, Theorem 1].

Recall that a CSP A with a total order $\sqsubset$ on its domain D is max-closed [81, Definition 2.5] if for each scope $S \in C$, $(S; k) \in A$ and $(S; \ell) \in A$ implies $(S; k \sqcup \ell) \in A$ where $k \sqcup \ell \in D^S$ contains the element-wise maximal elements of k and ℓ w.r.t. the total order $\sqsubset$ given by $\sqsubset$. Enforcing AC is refutation complete for max-closed CSPs [30, Example 6.39]. The same results also hold for min-closed CSPs that are defined dually.

For other classes and more details, we refer the interested reader to [31, 161, 30] and references therein. ◁

Example 1.12. Positive consistency is refutation complete for all CSPs. Indeed, by Definition 1.6, we have that $C_{\mathrm{pos}}(A)$ is non-empty if and only if A is satisfiable. However, enforcing positive consistency, deciding whether a CSP is positively consistent, or even finding a solution to a positively consistent CSP is likely intractable [67, 20, 57]. ◁

[22] Synonymous terms are that enforcing is a decision procedure for CSPs from $\mathcal{A}$ [33, 161, 133] or that decides CSPs from $\mathcal{A}$ [31, Definition 2]. Alternatively, one can say that enforcing is a complete refutation method [76, §3.2.1] (for a class of problems).

1.5 Weighted CSP and LP-Based Bounds

In this part, we formally define the weighted constraint satisfaction problem (WCSP) [23] which is an NP-hard combinatorial optimization problem [146]. Although there exist also other approaches [74], successful exact WCSP solvers usually rely on branch-and-bound search [136] which creates demand for good upper bounds that could efficiently prune the search space. Thus, we also give an overview of LP-based methods that can be used for obtaining such bounds. Throughout this section, we generally follow the notation that we used in [55a] or [56a] and also reuse certain parts of these papers.

The structure of a WCSP is the same as of a CSP, i.e., a triple (V, D, C). Similarly as with CSPs, we will assume that the structure is fixed and a WCSP is thus defined only by a collection of its weight functions $g_S : D^S \to \mathbb{R}$, $S \in C$. The task is to find an assignment $x \in D^V$ maximizing the objective function

$$F(x \mid g) = \sum_{S \in C} g_S(x_S) ; \tag{1.33}$$

or to find the optimal value of WCSP g, i.e., $\max_{x \in D^V} F(x \mid g)$.

Notice that the weights of all the weight functions can be stacked into a single real-valued vector $g \in \mathbb{R}^T$ where T is the set of all tuples, as defined in (1.23). Analogously to CSPs, we will identify WCSP instances with vectors from $\mathbb{R}^T$. The components of $g \in \mathbb{R}^T$ can thus be indexed by $t \in T$, e.g., $g_t = g_S(k)$ if $t = (S, k)$.

Example 1.13 ([56a, Example 1]). Let $V = \{1, 2, 3, 4\}$, $D = \{a, b\}$, and $C = \{\{1\}, \{2\}, \{2, 3\}, \{1, 4\}, \{2, 3, 4\}\}$. In such a setting, we want to maximize the expression

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

over $x_1, x_2, x_3, x_4 \in \{a, b\}$. In analogy to CSP terminology introduced in §1.4, this WCSP is Boolean but not pairwise. The factor graph of this WCSP is depicted in Figure 1.4. ◁

Remark 1.8. In some formalisms, the objective (1.33) is minimized. For our purposes, these settings are equivalent as one can invert the sign of all weights and maximize instead.

We emphasise that we make no assumption on the sign of the weights, as opposed to, e.g., [33, §7, 107, §2, 133, §II, 36, §3] where minimization and non-negative weights are assumed. When only non-negative weights are allowed, it is usual to assume that $\emptyset \in C$ since the weight $g_\emptyset$ then constitutes a bound on the optimal value. In contrast, we will need both positive and negative weights later in §3, so we require $\emptyset \in C$ for simplicity of notation (the empty scope would not yield any bound by itself anyway).

In more general setting, hard constraints can be introduced by allowing $g \in \mathbb{R}_1^T$ where $\mathbb{R}_1 = \mathbb{R} \cup \{-\infty\}$. We remark (without proof) that we see no obstacle to generalizing our results from later chapters to WCSPs with such constraints but we do not allow hard constraints for the sake of simplicity.

[23] This problem is also known as finite-valued CSP [132, 133], partial CSP [94], discrete energy minimization [88, 83, 119, 74, 135], max-sum (or min-sum) labeling problem [146, 120], or maximum a posteriori inference in graphical models (or Markov random fields) [125, 139, 146, 119, 134, 135, 87, 74]. It is also the main task in cost function networks [39]. Some of these formalisms however also allow infinite weights (i.e., hard constraints).

1.5.1 Linearity and Marginal Polytope

The objective (1.33) is linear in the weight vector g, i.e., for any $f, g \in \mathbb{R}^T$ and any $\alpha, \beta \in \mathbb{R}$, we have $F(x \mid \alpha f + \beta g) = \alpha F(x \mid f) + \beta F(x \mid g)$ for all $x \in D^V$. This is made explicit by using a different notation for the WCSP objective that is common in machine learning [140, §3, 119, §4, 88, §2, 139, §1.1]. We introduce this notation only for the purposes of this subsection, Example 1.14, and Remark 3.1 given later.

Let us define an indicator map $\varphi : D^V \to \{0,1\}^T$ by

$$\varphi(x)_t = [x_S = k] \qquad \forall t = (S,k) \in T \tag{1.34}$$

where $[\![\cdot]\!]$ denotes the Iverson bracket which equals 1 if the logical expression in the bracket is true and 0 if it is false. In other words, for any $x \in D^V$, we have that $\varphi(x)_t = 1$ if and only if x uses tuple $t \in T$.

The WCSP objective (1.33) can be written as the dot product of vectors g, $\varphi(x) \in \mathbb{R}^T$, namely

$$F(x \mid g) = \sum_{S \in C} g_S[x_S] = \sum_{t \in T} g_t \varphi(x)_t = g^\top \varphi(x) \tag{1.35}$$

which makes explicit that the objective (1.33) is linear in the weight vector g. The optimal value of a WCSP can be thus also expressed as

$$\max_{x \in D^V} F(x \mid g) = \max_{x \in D^V} g^\top \varphi(x) = \max_{\varphi \in M} g^\top \varphi = \max_{\varphi \in \operatorname{conv} M} g^\top \varphi \tag{1.36}$$

where conv denotes the convex hull operator [24, §2.1.4, 10, §1] and

$$M = \{\varphi(x) \mid x \in D^V\} \subseteq \{0,1\}^T. \tag{1.37}$$

The last equality in (1.36) follows from the well-known fact that a linear function on a polytope attains its maximum in at least one vertex of the polytope [119, §3.3].

Note that M is defined only by the structure (V, D, C). 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 [139, 119, 127, 140, 142] and references therein).

1.5.2 Active Tuples and Upper Bound

A tuple $t = (S,k) \in T$ is active for WCSP $g \in \mathbb{R}^T$ if

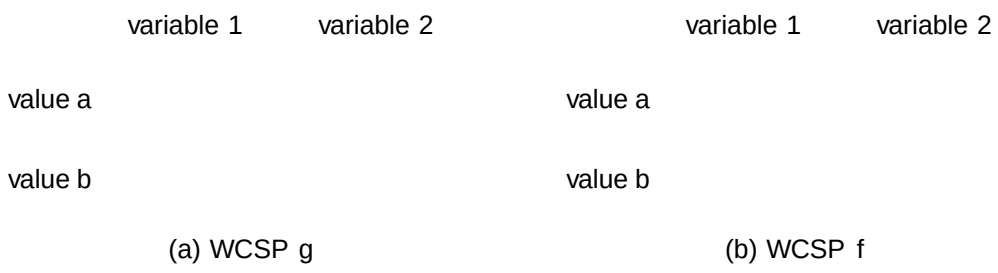
$$g_S(k) = \max_{\ell \in D^S} g_S(\ell); \qquad \text{i.e.,} \qquad g_t = \max_{t' \in T_S} g_{t'} \tag{1.38}$$

and is inactive otherwise. The set of all active tuples for g will be denoted[24] by $A(g)$. Since $A(g) \subseteq T$, $A(g)$ will be interpreted as a CSP (the active-tuple CSP).

We can define a tractable upper bound $B : \mathbb{R}^T \to \mathbb{R}$ on the optimal value of a WCSP g by

$$B(g) = \sum_{S \in C} \max_{k \in D^S} g_S(k) = \sum_{S \in C} \max_{t \in T_S} g_t \tag{1.39}$$

[24] The characteristic vector of the set $A(\cdot)$ is denoted by χ in [134, 146], by def in [149], and by $\mathrm{mi}[\cdot]$ in [119]. CSP $A(\cdot)$ is analogous to CSP $\mathrm{Bool}(\cdot)$ in [33, 107, 136].



(a) WCSP g (b) WCSP f

Figure 1.6: Visualisations of two WCSPs g and f with structure as in Examples 1.10 and 1.14. We depict WCSPs in analogy to CSPs: the active tuples are shown as black circles and full lines (because they are allowed in the active-tuple CSP) whereas inactive tuples are shown as white circles and dashed lines (because they are forbidden in the active-tuple CSP). The weights g_t (and f_t) are written next to the circles and line segments.

which is a convex piecewise-a ne function. Notice that the maxima in (1.39) are attained precisely by the active tuples[25] $t \in A(g)$. The properties that link function B to the set of active tuples are stated formally in the next theorem and corollary.

Theorem 1.15 ([146, §4, 149, Theorem 2]) Let $g \in \mathbb{R}^T$. For any $x \in D^V$, we have that

(a) $B(g) \geq F(x|g)$,

(b) $B(g) = F(x|g)$ if and only if $x \in \mathrm{SOL}(A(g))$.

Proof. Statement (a) is immediate after comparing expressions (1.33) and (1.39). Statement (b) follows from the de nition of an active tuple and a solution of a CSP. □

Corollary 1.5 ([146, 33]) Let $g \in \mathbb{R}^T$. The upper bound is tight, i.e., $\max_{x \in D^V} F(x|g) = B(g)$, if and only if $A(g)$ is satis able.

Proof. By Theorem 1.15, we have that $B(g) \geq \max_{x \in D^V} F(x|g)$ which holds as equality if and only if $\exists x \in D^V : x \in \mathrm{SOL}(A(g))$, which means that $A(g)$ is satis able. □

Example 1.14. Recall the structure $V = \{1, 2\}$, $D = \{a, b\}$, and $C = \{\{1\}, \{2\}, \{1, 2\}\}$ from Example 1.10.

An example of a WCSP g with this structure is shown in Figure 1.6a. The weight vector reads $g = (3, 4, 6, 2, \infty, 4, 1, 1) \in \mathbb{R}^T$ where the order of the tuples is given by (1.25). The objective value of WCSP g for $x = (a, b)$ is $F(x|g) = 3 + 2 \quad 4 = 1$ that can be also expressed as $g^\top \varphi(x) = 1$ where $\varphi(x) = (1, 0, 0, 1, 0, 1, 0, 0) \in \{0, 1\}^T$ (and the order of the tuples is again given by (1.25)).

The upper bound equals $B(g) = 4 + 6 + 1 = 11$ and is tight because CSP $A(g)$ is satisable. In particular, $F((b, a)|g) = 11$. On the other hand, for WCSP f from Figure 1.6b, $A(f)$ is unsatis able, so the upper bound $B(f) = 2 + 0 + 3 = 5$ is not tight. 4

[25] The term `active tuple' thus comes from the term `active inequality' (Footnote 2). Following §1.1.2, (1.39) can be calculated as the minimum of $\sum_{S \in C} z_S$ subject to $z_S \geq g_t \; \forall S \in C, t \in T_S$. At optimum, we have $z_S = \max_{t \in T_S} g_t$ and an inequality $z_S \geq g_t$ is active if and only if tuple t is active.

1.5.3 Reparametrizations and LP Relaxation

Definition 1.7 ([88, 87, 136, 117, 91]) Let $f, g \in \mathbb{R}^{\mathcal{T}}$. WCSP f is a reparametrization of WCSP g if $\mathrm{F}(x|f) = \mathrm{F}(x|g)$ for all $x \in \mathrm{D}^{\mathrm{V}}$.[26]

The binary relation `is a reparametrization of' on the set $\mathbb{R}^{\mathcal{T}}$ is an equivalence, i.e., it is symmetric, reflexive, and transitive. It is easy to see that if f is a reparametrization of g, then $\mathrm{B}(f)$ is not only an upper bound on the optimal value of f, but also on the optimal value of g. This suggests minimizing the upper bound on WCSP g over its reparametrizations, i.e.,

$$\min \mathrm{B}(f) \tag{1.40a}$$

$$\mathrm{F}(x|f) = \mathrm{F}(x|g) \qquad \forall x \in \mathrm{D}^{\mathrm{V}} \tag{1.40b}$$

$$f \in \mathbb{R}^{\mathcal{T}}: \tag{1.40c}$$

Although this optimization problem has an exponential number of constraints (1.40b), these constraints are linear in f by §1.5.1, hence the set of feasible solutions to (1.40) is an affine subspace of $\mathbb{R}^{\mathcal{T}}$.

We will now briefly focus on describing the set of all reparametrizations of a WCSP. Let $g \in \mathbb{R}^{\mathcal{T}}$ and $\phi_{S,i}(k) \in \mathbb{R}$ for each $S \in \mathcal{C}_2$, $i \in S$, and $k \in \mathrm{D}$. WCSP $g' \in \mathbb{R}^{\mathcal{T}}$ defined by

$$g'_{\{i\}}(k) = g_{\{i\}}(k) + \sum_{S \in \mathcal{C}_2 \,:\, i \in S} \phi_{S,i}(k) \qquad \forall i \in \mathrm{V},\ k \in \mathrm{D} \tag{1.41a}$$

$$g'_{S}(k) = g_{S}(k) - \sum_{i \in S} \phi_{S,i}(k_i) \qquad \forall S \in \mathcal{C}_2,\ k \in \mathrm{D}^{S} \tag{1.41b}$$

is a reparametrization of g [149, §3.2].[27] If the WCSP is pairwise and the graph $(\mathrm{V}, \mathcal{C}_2)$ is connected, one can obtain any reparametrization of g by an appropriate choice of ϕ [146, 88, 120]. In general, if the graph is not connected or the WCSP is not pairwise, not every reparametrization of g can be obtained by some choice of ϕ in (1.41).

Remark 1.9. There exist larger subsets of reparametrizations that can be described by means of a coupling scheme [149, §3] which is a subset $\mathcal{S} \subseteq \{(S, S') \mid S, S' \in 2^{\mathrm{V}} \setminus \emptyset,\ S \neq S'\}$. For a coupling scheme $\mathcal{S}$, let $\phi_{S,S'}(k) \in \mathbb{R}$ be scalars for each $(S, S') \in \mathcal{S}$ and $k \in \mathrm{D}^{S'}$. Then, WCSP $f \in \mathbb{R}^{\mathcal{T}}$ defined by[28]

$$f_S(k) = g_S(k) - \sum_{(S',S) \in \mathcal{S}} \phi_{S',S}(k) + \sum_{(S,S') \in \mathcal{S}} \phi_{S,S'}(k_{S'}) \qquad \forall S \in \mathcal{C},\ k \in \mathrm{D}^{S} \tag{1.42}$$

is a reparametrization of $g \in \mathbb{R}^{\mathcal{T}}$ [149, §3.2]. See that the transformation (1.41) is a special case of (1.42) for the coupling scheme $\mathcal{S} = \{(S, \{i\}) \mid S \in \mathcal{C}_2,\ i \in S\}$.

---

[26] It is also often said that f is equivalent to g [107, 134, 33, 135, 149, 146, 136, 36, 120, 98, 108]. Other related terms are that f is an equivalence-preserving (or equivalent) transformation of g.

[27] Especially in machine learning, variables ϕ are sometimes referred to as `messages' [146, 88, 125, 149].

[28] We will not define a specific notation for WCSPs obtained using (1.42) as we use only the (simpler and less general) transformation (1.41) in the sequel.

Replacing variables f in (1.40) by g^{ϕ} from (1.41) while introducing ϕ as variables results in the optimization problem $\min_{\phi} B(g^{\phi})$ which can be interpreted as an unconstrained minimization of a convex piecewise-affine function. Written explicitly, this is

$$\min \sum_{i \in V} \max_{k \in D} g_{\{i\}}(k) + \sum_{S \in C_{\geq 2}} \sum_{i \in S} \phi_{S;i}(k) + \sum_{S \in C_{\geq 2}} \max_{k \in D^S} g_S(k) + \sum_{i \in S} -\phi_{S;i}(k_i) \tag{1.43a}$$

$$\phi_{S;i}(k) \in \mathbb{R} \qquad \forall S \in C_{\geq 2},\ i \in S,\ k \in D, \tag{1.43b}$$

which can be formulated as a linear program, as discussed in §1.1.2.

The dual linear program corresponds to the basic LP relaxation of the WCSP g [146] that can be stated as[29]

$$\max \sum_{S \in C} \sum_{k \in D^S} g_S(k)\, \mu_S(k) \tag{1.44a}$$

$$\sum_{\ell \in D^S : \ell_i = k} \mu_S(\ell) = \mu_{\{i\}}(k) \qquad \forall S \in C_{\geq 2},\ i \in S,\ k \in D \tag{1.44b}$$

$$\sum_{k \in D} \mu_{\{i\}}(k) = 1 \qquad \forall i \in V \tag{1.44c}$$

$$\mu_S(k) \geq 0 \qquad \forall (S,k) \in T. \tag{1.44d}$$

Indeed, this is an LP relaxation of WCSP g since there is a bijection between assignments $x \in D^V$ and the integral vectors μ feasible for (1.44). This LP relaxation was proposed independently a number of times [121, 94, 28, 139, 38] and it is a powerful tool in the sense that it solves all WCSPs defined by a tractable constraint language (i.e., the set of allowed weight functions) [132].

Remark 1.10. In more detail, depending on the set of allowed weight functions, the resulting class of WCSPs can be either NP-hard or polynomially solvable [132][30]. In the latter case, the optimal value of the basic LP relaxation of g coincides with the optimal value of WCSP g [132].

If the WCSP is Boolean and pairwise, the LP relaxation (1.44) is half-integral and can be reduced to the minimum st-cut problem [119, §12, 117, 23, 146]. On the other hand, even for pairwise structure with $|D| \geq 3$, solving the problem (1.44) (or (1.43)) is as hard as solving a general linear program [115].

1.5.4 Methods for Obtaining Bounds Using Reparametrizations

Although linear programs can be solved in polynomial time, our ability to find optimal solutions is limited by the fact that the time complexity of current off-the-shelf LP solvers is super-linear which makes them impractical for large-scale instances which occur, e.g.,

[29] Formally, there should also be the constraint $\sum_{k \in D^S} \mu_S(k) = 1$ for each $S \in C_{\geq 2}$ but these constraints are already implied by (1.44b) together with (1.44c) and are typically not included in the basic LP relaxation [133, 132, §3.1].

[30] An analogous statement (the Feder-Vardi conjecture [58]) was proved for CSPs independently in [159] and [27]. As stated in [89], this implies that general-valued CSPs (i.e., WCSPs that additionally allow $-\infty$ weights) also exhibit a dichotomy.

in computer vision [154, 88, 127, 134, 115]. We will now give an overview of methods for bounding the optimal value of (1.43) { such methods were developed, to some extent independently, in computer vision/machine learning and constraint programming communities.

1.5.4.1 Methods Based on BCD / Message Passing

To obtain good upper bounds while avoiding solving the LP relaxation to optimality, a competitive approach is to apply BCD (in this context also called `message passing') to the problem (1.43). There exists a wide class of convergent message-passing algorithms that optimize di erent forms of a dual LP relaxation of WCSP by BCD. This family of algorithms originates from the eld of computer vision.

For pairwise WCSPs, such algorithms include, e.g., MPLP [65], max-sum di usion [96, 146], MPLP++ [134], or SPAM [135]. The xed points of these methods are related to non-empty AC closure[31] of $A(g')$. To be more precise, if the current solution is φ' and the AC closure of $A(g')$ is non-empty, then the aforementioned algorithms will not be able to improve the objective further. On the other hand, if the AC closure of $A(g')$ is empty, the objective will be improved after a nite number of BCD iterations.

For instances with weight functions of higher arity, there exist specialized algorithms, such as [87, 149] whose stopping conditions are also based on local consistencies.

Since the optimized objective is a convex piecewise-a ne function, the xed points of these algorithms need not be global minima. Indeed, non-empty AC closure of $A(g')$ is only a necessary condition for optimality of φ' for (1.43) (or optimality of $f = g'$ for (1.40)) but not su cient in general [146].

Fact 1.2. Consider any algorithm for (approximate) optimization of (1.43) that returns points φ' such that $A(g')$ has non-empty AC closure. If, upon termination, $A(g')$ is in some class of CSPs for which AC is refutation complete (recall Example 1.11), then $A(g')$ is satis able and $\mathrm{B}(g')$ is the optimal value of WCSPg by Corollary 1.5 [33, §10].

This is the case, e.g., when the factor graph of the WCSP is acyclic. Another example are instances where each g_S, $S \in \mathcal{C}$ is supermodular because then $A(g')$ is both max-closed and min-closed [33, §10, 149, §7] and AC is in such case refutation complete. Note that the transformation (1.41) preserves supermodularity [36, §4, 149, §7, 120, §2.3]. We also remark that pairwise WCSPs with supermodular weight functions can be solved by a reduction to minimum st-cut [29, 120, 119, §11].

Fact 1.3. In Boolean pairwise WCSPs, non-empty AC closure of $A(g')$ is a su cient condition for optimality of φ' for (1.43) [146] and the optimal value of the LP relaxation (1.43) can be computed by reduction to minimum st-cut [119, §12, 117, 23, 146] (also see [91]). However, the LP relaxation need not be tight.

Since non-empty AC closure of the active-tuple CSP need not be su cient for its satis ability (hence optimality of the obtained bound by Corollary 1.5), one can include zero weight functions of higher arity to improve the bound even further, as it is done in [127, 11, 149, 147, 126] and corresponds to a ne-grained version of the Sherali-Adams hierarchy [124] for WCSP. Even though adding such factors can strengthen the LP relaxation

[31] Called node-edge agreement in [119, 134] and non-empty kernel in [146].

or improve the fixed points of BCD algorithms [127, 149, 11, 126], this approach may significantly increase the memory requirements of the method.

Aside from the previously mentioned algorithms, there are also approaches that optimize an LP relaxation based on acyclic decompositions. However, the optimal value of such a relaxation is the same as of (1.43) [119, 93, 88]. This is, e.g., the TRW-S algorithm [88] whose fixed points satisfy a local consistency condition called *weak tree agreement*. Moreover, any feasible solution satisfying weak tree agreement cannot be improved by subsequent iterations of TRW-S. Based on the study [83], TRW-S is typically the fastest method for (approximately) optimizing the LP relaxation but seems to be recently surpassed by SPAM [135].

Remark 1.11. For pairwise WCSPs, the formulation (1.43) can be interpreted as a decomposition based on individual vertices and edges of the graph $(V; C_2)$ and non-empty AC closure is then a special case of weak tree agreement [134]. Moreover, the weak tree agreement condition is in a precise sense equivalent to non-empty AC closure (up to reformulation) [119, 88] (also mentioned in [146]). In [129], several BCD algorithms were presented in a unified view within the framework of Lagrange dual decomposition of combinatorial problems and the fixed point conditions of this approach generalize both arc consistency and weak tree agreement.

The connections among different formulations of (1.43) along with a BCD method on trees is given in [125]. Some of the mentioned BCD methods are also studied in a common framework in [142]. A more recent overview and taxonomy is presented in [135].

1.5.4.2 Methods Based on Enforcing (Soft) Local Consistencies

A different class of algorithms for obtaining an upper bound is based on enforcing (soft) local consistencies (instead of performing BCD). Such algorithms generally originated in the constraint programming community.

This is, e.g., the Virtual Arc Consistency (VAC) algorithm [33] that enforces non-empty AC closure of the active-tuple CSP and if the AC closure is found to be empty, the algorithm traces the operations of the AC propagator in anti-chronological order to find a reparametrization with a better bound. VAC algorithm is closely related to the Augmenting DAG algorithm [95, 146] that also enforces non-empty AC closure but was defined only for pairwise WCSPs. Because the terminating condition of VAC is non-empty AC closure of the active-tuple CSP, Facts 1.2 and 1.3 are also applicable.

Aside from VAC, there are also other methods based on arc consistency, such as EDAC [43], FDAC, or DAC [37, 98], which are faster at the cost of weaker bounds in general. A stronger local consistency is Optimal Soft Arc Consistency (OSAC) [38, 33] that is (by definition) satisfied by g' where $'$ is an optimal solution of (1.43).[32] OSAC is however limited to preprocessing, as it is too expensive to be maintained during search [33]. Higher-order consistencies for weighted CSP have been also studied in the constraint programming community, e.g., in [35].

However, it is known that for a given WCSP, there need not exist its reparametrization (with the same structure) such that its active-tuple CSP satisfies a given local consistency. Thus, stronger local consistencies (and thus better bounds) can be again obtained at the

[32] Assuming that there is only a single weight function for each scope, (dual of) the basic LP relaxation coincides with the OSAC formulation. In more general settings, this need not hold [90, Footnote 1, 133].

cost of introducing weight functions of higher arity (if such weight functions were not already present in the problem), e.g., these are ternary weight functions if one wants to enforce triangle-based consistencies, as it is done in [107].

Remark 1.12. As a technical note, the aforementioned algorithms based on enforcing local consistencies do not obtain a reparametrization using (1.41) explicitly. Instead, the weights are shifted between the individual weight functions by applying the operations project and extend [98, 43, 33, 108]. However, for soft arc consistencies, these operations are in one-to-one correspondence with increasing or decreasing individual values of θ in (1.41). In more general settings, the relation is analogous except that one uses a different coupling scheme (recall Remark 1.9) between the weight functions.

1.5.4.3 Applications of Reparametrizations and Approximate Solutions

The listed algorithms are not only useful for pruning the search space in branch-and-bound search by providing upper bounds, but can also provide heuristics on which variables to branch (see, e.g., [136]). An alternative approach for reducing the search space is to divide the initial problem into two parts: a `hard' part that is solved by an exact solver and an `easy' part where the LP relaxation (1.43) is tight [74]. This reduces the size of the problem that needs to be solved by a combinatorial algorithm. Moreover, to solve the `easy' part, it is not necessary to rely on an exact LP solver, but one can instead utilize the approximate approaches based on BCD or enforcing local consistencies [74].

Aside from exact solving, one can also use the reparametrized instance to obtain solutions that are acceptable in practice. As an example of such a primal heuristic, one can define an arbitrary total order on the variables V and then, in this order, choose for each variable such a value from D so that the objective (1.33) (where we sum only over the scopes whose variables are already instantiated) is the greatest. This greedy technique was proposed in [88, §4.3] and used in many computer vision applications [93, §4.2, 83, §4.5, 134, Equation (6), 129, §4.2, 92, §3.1], possibly with slight modifications.

1.5.5 Super-Reparametrizations

We now point our attention to the approach from [92] where super-reparametrizations[33] were introduced in order to reach tighter upper bounds on the WCSP optimal value.

Definition 1.8 ([92]). Let $f, g \in \mathbb{R}^{\mathcal{T}}$. WCSP f is a super-reparametrization of WCSP g if $\mathrm{F}(x|f) \geq \mathrm{F}(x|g)$ for all $x \in \mathrm{D}^{V}$.

In analogy to (1.40), it was proposed in [92, §2] to minimize the upper bound (1.39) over super-reparametrizations, i.e.,

$$\min \mathrm{B}(f) \tag{1.45a}$$

$$\mathrm{F}(x|f) \geq \mathrm{F}(x|g) \qquad \forall x \in \mathrm{D}^{V} \tag{1.45b}$$

$$f \in \mathbb{R}^{\mathcal{T}}: \tag{1.45c}$$

We note that this formulation does not belong to the classical hierarchy of LP relaxations based on introducing additional weight functions of higher arity, mentioned in §1.5.4.

[33] Originally, super-reparametrizations were called virtual potentials in [92] and sup-reparametrizations in [125]. We introduced the more descriptive term super-reparametrization in [55a].

Also, in contrast to (1.40), it is unlikely that there is a compact (i.e., polynomially sized) representation of the optimization problem (1.45) since its optimal value coincides with the optimal value of WCSP g, as given by the following theorem.

Theorem 1.16 ([92, Theorem 1, 125]) The optimal value of (1.45) is $\max_{x \in D^V} F(x|g)$.

Proof. We have that $B(f) \geq F(x|f) \geq F(x|g)$ for any $x \in D^V$, so $B(f) \geq \max_x F(x|f) \geq \max_x F(x|g)$. Consequently, the optimal value of (1.45) is at least $m = \max_x F(x|g)$.

To show that this value is attained, define f by

$$f_t = m/|C| \quad \forall t \in T. \tag{1.46}$$

It can be checked from (1.33) and (1.39) that $B(f) = F(x|f) = m$ for all $x \in D^V$. Due to $m \leq F(x|g)$ for all $x \in D^V$ by definition of m, f is feasible and optimal. □

To approximately solve (1.45), iterations of Augmenting DAG algorithm were interleaved with a so-called cycle-repair procedure in [92]. In detail, this procedure found inconsistent cycles (i.e., cycles that do not allow an assignment satisfying all constraints in the cycle) in the CSP $A(f)$ and, if such an inconsistent cycle was found, the weights of the tuples along the edges in this cycle were manipulated so that some of the tuples became inactive and the bound could be improved by Augmenting DAG algorithm. We note that this approach was defined only for pairwise WCSPs and its extension to higher-order WCSPs or other notions of local consistencies is not straightforward.

For certain instances, this method was able to obtain bounds superior to TRW-S and in some cases reported reaching optimal value of the original WCSP.

To the best of our knowledge, super-reparametrizations were not utilized nor analyzed except for [92] and [125]. However, [125] focuses almost solely on the relation between different formulations of reparametrizations, instead of super-reparametrizations.

Chapter 2

Bounds on Large-Scale Linear Programs Using Constraint Propagation

Although linear programs are solvable in polynomial time, solving very large sparse instances can be challenging in practice. Such linear programs occur in many areas, a prominent example being the computation of bounds in branch-and-bound search by LP relaxation. In such a setting, the applicability of classical o -the-shelf LP solvers is limited [154, 129, 128, 73, 47] which calls for the development of specialized (possibly approximate) methods. In this chapter, we present and exemplify our approach to approximately solving large-scale linear programs that we originally proposed in [52a].

Based on §1.1, we start in §2.1 by explaining how to practically perform constraint propagation (in this case, infer new implied inequalities) in a system of linear inequalities. Then, we review our general framework for approximate optimization of linear programs using constraint propagation, including a variant of capacity scaling that ensures its niteness. Furthermore, we show that the VAC algorithm [33] can be (up to technical details) interpreted as an algorithm belonging to this framework. Finally, we exemplify this approach on the LP relaxation of weighted Max-SAT.

Some of our motivations for this approach were the logic-theoretic view on LP duality [103, §6] and the concept of inference duality [76, §17]. However, our approach is not a straightforward application of inference duality as we put inference into an iterative optimization scheme and do it in a refutation-incomplete way. Some parts of this chapter were published in [52a].

Here, we broaden the notion of refutation-completeness so that it can be applied to settings where we try to detect infeasibility of a system of linear inequalities and equalities by speci c constraint propagation rules. Thus, we will more generally say that a propagation method is refutation complete for a given class of problems if it detects a contradiction in any infeasible/unsatis able problem from the class and is refutation incomplete otherwise.

2.1 Constraint Propagation for Linear Inequalities

Inference in a system of linear inequalities (and equalities) was discussed in §1.1.3. Here, we would like to determine whether such a system is feasible or not using constraint propagation.[34] Suppose that a xed set of inference rules is at our disposal. Using these rules (which are problem-dependent), we generate new linear inequalities until either no new inequality can be generated or a contradiction (e.g., the inequality $0 \geq 1$) is found.

[34] As mentioned in §1.4.1, constraint propagation can be also seen as inference of new constraints. In this part, we consider adding implied constraints (i.e., implied inequalities) to the system instead of tightening the original constraints (which was discussed in §1.4.1 for the CSP). We analyze a possible analogy of constraint tightening in §4. However, tightening a constraint $A^i x \geq b_i$ to $A^i x = b_i$ is in correspondence to inferring that $A^i x \geq b_i$.

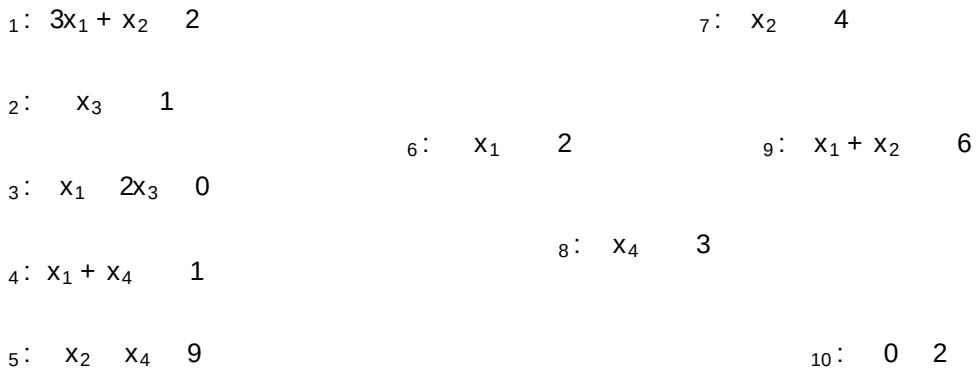


Figure 2.1: Propagation in a system of linear inequalities. The inequalities are indexed by 1{10. Inequalities 1{5 are initial, inequalities 6{10 are inferred. Edge weights indicate the coe cients of non-negative combinations.

Each time a new inequality is generated, its cause vector is stored, encoding how the inequality was obtained from the existing inequalities. In detail, the cause vector is given by the a ne form of Farkas' lemma (Theorem 1.4) and contains the coe cients with which the original inequalities were combined to obtain the new inequality, as discussed in x1.1.3. When a contradiction is found, a certi cate of infeasibility (i.e., a vector satisfying the Farkas alternative system) corresponds to its cause vector. However, as we will discuss next, one need not store all the cause vectors explicitly because, if a contradiction is found, the certi cate of infeasibility can be computed e ciently by tracking the newly generated inequalities back to the original system.

Remark 2.1. The procedure explained above is independent of the speci c form of the system to which it is applied. In the rest of this section, we choose to exemplify it on a system in the form $Ax \quad b$. Although any system of linear inequalities (and possibly equalities) can be transformed to this form in linear time, such a transformation is not needed as one can always use the appropriate version of a ne Farkas' lemma to derive new inequalities. E.g., we showed Farkas' lemma and its a ne form for a system in the form $Ax = b;\ x \quad 0$ in Corollary 1.2 and Theorem 1.5, respectively.

Let us focus on obtaining the certi cate of infeasibility. As a running example, consider the system $Ax \quad b$ of $m = 5$ initial inequalities on the left in Figure 2.1. From inequalities $_2$ and $_3$, we infer inequality $_6 = 2\ _2 + \ _3$.[35] Next, we gradually infer inequalities $_7 = \ _1 + 3\ _6$, $_8 = \ _4 + \ _6$, $_9 = \ _6 + \ _7$, and nally $_{10} = \ _5 + \ _7 + \ _8$. Since $_{10}$ reads $0 \quad 2$, the initial system $_1; : : : ; \ _5$ is infeasible.

The history of propagation is represented by a directed acyclic graph (DAG) $(V; E)$ where V is the set of all (initial and inferred) inequalities and $E \quad V \quad V$ is a set of directed edges with weights $w : E \ ! \ R_+$ so that each inferred inequality is given by $_i = \sum_{j 2 N_i^+} w_{ij} \ _j$ where $N_i^+ = f \ j \ 2 \ V \ j \ (i; j \ ) 2 \ E \ g$. By composing the inferences, each inequality $_i$ can be expressed in terms of the initial inequalities as $_i = \sum_{j=1}^{m} y_j^i \ _j$ where

[35] As in Example 1.2, we `sum' inequalities in the following informal (yet natural) way: $2 \quad (x_3 \quad 1) + (\ x_1 \quad 2x_3 \quad 0) = (\ x_1 \quad 2)$.

$y^i = (y^i_1; \ldots; y^i_m) \in R^m_+$ is the cause vector of α_i. For $i \leq m$, we have $y^i = e^i$ where e^i is the i-th standard-basis vector of R^m. For $i > m$, we have

$$y^i = \sum_{\substack{j \in N_i^+ \\ N_j^+ = \emptyset}} w_{ij}\, e^j + \sum_{\substack{j \in N_i^+ \\ N_j^+ \neq \emptyset}} w_{ij}\, y^j : \tag{2.1}$$

Note that $N_j^+ = \emptyset$ holds only for the initial inequalities α_j, $j \leq m$.

In the example, $V = \{1; \ldots; 10\}$, $y^6 = 2e^2 + e^3 = (0; 2; 1; 0; 0)$, $y^7 = e^1 + 3y^6$, $y^8 = e^4 + y^6$, $y^9 = y^6 + y^7$, and $y^{10} = e^5 + y^7 + y^8 = (1; 8; 4; 1; 1)$. Since $b^\top y^{10} = 2$ and $A^\top y^{10} = 0$, vector y^{10} is a certificate of infeasibility of the system $Ax \leq b$ by Farkas' lemma (Corollary 1.1).

More generally, the cause vector $y^i \in R^m_+$ for inequality α_i given by $(c^i)^\top x \leq d_i$ satisfies $A^\top y^i = c^i$ and $b^\top y^i \leq d_i$ (recall Theorem 1.4).

2.1.1 Computing Certificate of Infeasibility

As we need the cause vector only for the final (contradictory) inequality (α_{10} in the example), storing all cause vectors explicitly in the memory is wasteful. In addition, some inferred inequalities may not be needed for the proof of infeasibility (α_9 in the example). We show that any single cause vector can be computed more efficiently by dynamic programming.

Let α_i, $i \in [m]$ be an initial inequality and α_k, $k > m$ be an inferred inequality. Then, y^k_i is the sum of weight-products[36] of all directed paths from node k to node i in the DAG. Suppose we want to compute y^k for some single k. We can consider only the subgraph of the DAG reachable from node k along directed paths. We introduce auxiliary variables z_j, which are to equal the sum of weight-products of all directed paths from node k to node j. Initially, we set $y^k = 0$, $z_k = 1$, and $z_j = 0$ for all $j \neq k$. Then, we process the nodes i of the subgraph in a topological order as follows: if $N_i^+ = \emptyset$ then set $y^k_i := z_i$, otherwise update $z_j := z_j + w_{ij}\, z_i$ for all $j \in N_i^+$. Eventually, we have $y^k_i = z_i$ for all $i \in [m]$. The time and space complexity of this algorithm is linear in the size of the graph.

2.2 Bounding the Optimal Value of Linear Programs

In §2.1, we explained how constraint propagation in a system of linear inequalities can be performed and, if infeasibility is detected, how to compute a certificate of infeasibility. Here, we will show how constraint propagation can be used to improve a non-optimal solution of a linear program.

For this, suppose that we have a primal-dual pair of linear programs and a dual-feasible solution y. For this fixed y, we construct the complementary slackness conditions in terms of the primal variables, which is a system of linear inequalities and equalities that is feasible if and only if y is optimal for the dual. We will show that, if infeasibility of the complementary slackness conditions is detected, then any certificate of infeasibility constitutes a dual-improving direction that can be used to improve the current dual-feasible y.

[36] We define the weight-product of a path to be the product of all edge weights along the path.

In the sequel, we show this approach for the primal-dual pair in the form (1.1).[37] To this end, suppose that we have a feasible solution y of the dual (1.1) (by which we implicitly assume that the dual is feasible). We would like to determine whether it is optimal and, if it is not optimal, find a dual-feasible solution with an improved objective.

By Theorem 1.2 (complementary slackness), a feasible $y \in R^m$ is optimal for the dual (1.1) if and only if there exists $x \in R^n$ feasible for the primal (1.1) that satisfies complementary slackness conditions with y, i.e., if

$$Ax = b \tag{2.2a}$$
$$x_j \geq 0 \qquad \forall j \in J \tag{2.2b}$$
$$x_j = 0 \qquad \forall j \in [n] \setminus J \tag{2.2c}$$

with $J = J(y)$ (defined in (1.2b)) is feasible. By Farkas' lemma (recall Corollary 1.2), system (2.2) is infeasible if and only if system

$$b^\top y < 0 \tag{2.3a}$$
$$A_j^\top y \geq 0 \qquad \forall j \in J \tag{2.3b}$$

is feasible. Any vector y satisfying (2.3) is a certificate of infeasibility for (2.2). Note, the set of y feasible for (2.3) is a convex cone.

Moreover, it is easy to verify that any y feasible for (2.3) with $J = J(y)$ constitutes a dual-improving direction from y, i.e., one can update $y := y + \theta \Delta y$ for a suitable step size $\theta > 0$ and improve the dual objective, as shown in Proposition 2.1.

Proposition 2.1 (cf. [110, Theorem 5.2]) Let y be feasible for the dual (1.1), Δy satisfy (2.3) for $J = J(y)$, and

$$\bar{\theta} = \min_{\substack{j \in [n] \\ A_j^\top \Delta y < 0}} \frac{c_j - A_j^\top y}{A_j^\top \Delta y} > 0. \tag{2.4}$$

Then, $y' = y + \theta \Delta y$ is feasible for the dual (1.1) and $b^\top y > b^\top y'$ if and only if $0 < \theta \leq \bar{\theta}$.

Proof. First, note that $\bar{\theta} > 0$ which follows from the fact that if $A_j^\top \Delta y < 0$, then $j \in J = J(y)$ by (2.3b), hence $c_j - A_j^\top y < 0$ by definition of $J(y)$ in (1.2b). This together with $A_j^\top \Delta y < 0$ in (2.4) implies that each term in the minimum (2.4) is positive.[38]

We begin by the `if' direction. To prove feasibility of y', i.e., $A_j^\top y' \leq c_j$ for all $j \in [n]$, we consider two cases. If $A_j^\top \Delta y \geq 0$, then $A_j^\top y' = A_j^\top y + \theta A_j^\top \Delta y \geq A_j^\top y \leq c_j$ where we used $\theta > 0$ and the assumption that y is feasible. If $A_j^\top \Delta y < 0$, then $\theta \leq (c_j - A_j^\top y)/A_j^\top \Delta y$ by definition of $\bar{\theta}$, which is equivalent to $A_j^\top y' = A_j^\top y + \theta A_j^\top \Delta y \leq c_j$. Also, $b^\top y' = b^\top y + \theta b^\top \Delta y < b^\top y$ due to (2.3a) and $\theta > 0$.

For the `only if' direction: if $\theta \leq 0$, we have $b^\top y \leq b^\top y'$. If $\theta > \bar{\theta}$, then there is $j \in [n]$ such that $A_j^\top \Delta y < 0$ and $(c_j - A_j^\top y)/A_j^\top \Delta y < \theta$. This implies $A_j^\top y' = A_j^\top y + \theta A_j^\top \Delta y < c_j$, i.e., y' is infeasible. □

[37] In analogy to Remark 2.1, such an approach is applicable to a primal-dual pair in any form. The differences again lie only in the form of the complementary slackness conditions that determines the form of the certificate of infeasibility (i.e., improving direction).

[38] If there is no $j \in [n]$ with $A_j^\top \Delta y < 0$, the dual (1.1) is unbounded and θ can be chosen arbitrarily large.

```
inputs: instance of problem (1.1), dual-feasible solution y, constraint
        propagation rules for (2.2).
repeat
    Compute J = (y).
    Try to detect infeasibility of (2.2) by constraint propagation.
    if (2.2) is proved to be infeasible then
        Find an improving direction y satisfying (2.3).
        Compute (possibly non-optimal) step size > 0 so that y + y is feasible.
        Update y := y + y.
    else
        return y   (At this point, we are unable to prove that (2.2) is infeasible.)
```

Algorithm 2.1: Iterative scheme for approximate optimization of the dual (1.1).

Remark 2.2. Step size given by (2.4) is optimal in the sense that, for the fixed improving direction y, it provides the largest possible improvement of the dual objective.

These properties can be used to iteratively optimize a linear program if an initial dual-feasible solution y is provided. In detail, one can construct system (2.2) and, if it is infeasible, find an improving direction y satisfying (2.3), update y := y + y using Proposition 2.1, and improve the current objective while maintaining feasibility. By repeating this iteration, one obtains a better and better bound on the common optimal value of (1.1). Eventually, if (2.2) with J = (y) becomes feasible for current y, y is optimal for the dual. Note that, even though the dual objective improves after each iteration, this general scheme need not terminate in a finite number of steps or even converge to an optimal solution.

Remark 2.3. This optimization scheme is related to the primal-dual[39] algorithm [110, §5] (referred to as primal-dual method in [103, §7]) where complementary slackness conditions are not enforced strictly, but their violation is minimized instead. This change ensures convergence and finiteness of the algorithm. Practical algorithms for solving certain combinatorial problems that can be formulated as linear programs (e.g., shortest path problem or maximum flow) can be seen as instantiations of the primal-dual algorithm [110, 103].

However, since determining feasibility of (2.2) is in general as hard as solving a linear program, we propose to do it in a refutation-incomplete way by constraint propagation that may detect infeasibility only sometimes. In other words, we propose to apply some (problem-dependent) constraint propagation rules to the complementary slackness conditions (2.2) and, whenever infeasibility is detected, construct the certificate of infeasibility that turns out to be a dual-improving direction. The iterative scheme based on constraint propagation is outlined in Algorithm 2.1. In analogy to the primal-dual algorithm (Remark 2.3), we called this iterative scheme primal-dual approach in [54a].

We emphasise that points y returned by Algorithm 2.1 need not be optimal since one may not detect infeasibility of (2.2) even if (2.2) is infeasible. However, even non-optimal solutions can be useful in practice. As an example, if the primal (1.1) is an LP relaxation

[39] As remarked in [110, §5.2], this is a misnomer because only a dual-feasible solution is maintained.

of a hard combinatorial problem (which is to be maximized), then any solution feasible for the dual (1.1) provides an upper bound on the optimal value of the original combinatorial problem. As we discussed earlier, such bounds can be used in branch-and-bound search where it may not be necessary or e cient to solve the LP relaxation to optimality as there is a trade-o between the time spent in computing the bound and the time spent in search.

2.2.1 Finiteness and Capacity Scaling

As already indicated, Algorithm 2.1 need not generally terminate in a nite number of iterations. In this section, we state su cient conditions for its niteness.

First of all, notice that the improving direction y generated on line 5 of Algorithm 2.1 can be chosen as any vector satisfying (2.3). To guarantee niteness, we require a technical assumption, namely that there exists a nite set $\mathcal{Y}$ such that any improving direction used by Algorithm 2.1 is from this set. This condition is not satis ed trivially because the same set J may be encountered repeatedly during the run of Algorithm 2.1 and, in each such an iteration, the improving direction y may be chosen as a di erent vector satisfying (2.3).

The following theorem shows that if the dual is in addition bounded and there exists a positive lower bound on the step sizes used in Algorithm 2.1, then the algorithm terminates after a nite number of iterations.

Theorem 2.1. Let the dual (1.1) be feasible and bounded. Algorithm 2.1 terminates after a nite number of iterations (i.e., updates of y) provided that there exists $\varepsilon_{min} > 0$ and a nite set $\mathcal{Y} \subseteq R^m$ such that in every iteration:

(a) improving direction y found on line 5 belongs to $\mathcal{Y}$,

(b) step size computed on line 6 satis es $\geq \varepsilon_{min}$.

Proof. We follow a proof technique analogous to [48a, §3.2.4]: we show that there is a value $\delta > 0$ that depends only on the instance (i.e., on A, b, c) such that the dual objective improves at least by δ in each iteration. Thus, if the dual has an optimal solution y^* and the algorithm is initialized in y^0, it terminates after at most $\lceil (b^\top y^0 - b^\top y^*)/\delta \rceil$ iterations.

Without loss of generality, we assume that $b^\top y < 0$ for each $y \in \mathcal{Y}$. If some $y \in \mathcal{Y}$ violates this property, then it can be removed from $\mathcal{Y}$ as it is never used by the algorithm because it does not satisfy (2.3) for any $J \subseteq [n]$.

With this assumption, the objective improves in each iteration at least by $\delta = \min_{y \in \mathcal{Y}} (-\varepsilon_{min} b^\top y)$ because, after any update from y to $y + \alpha y$, $b^\top y - b^\top (y + \alpha y) = b^\top y - \alpha b^\top y \geq b^\top y - \varepsilon_{min} b^\top y$.

For completeness, if $\mathcal{Y} = \emptyset$, δ is not well-de ned. But, in such case, Algorithm 2.1 always terminates already with the initial point as it is not capable of providing any improving direction. □

We will now discuss how the assumptions of Theorem 2.1 can be satis ed in practice. We begin by commenting on condition (a).

Because set $[n]$ has only a nite number of subsets, the existence of a nite set $\mathcal{Y}$ satisfying condition (a) in Theorem 2.1 is equivalent to the following: for each $J \subseteq [n]$, there exists a nite set $\mathcal{Y}(J)$ satisfying

$$\mathcal{Y}(J) \supseteq \{ y \in R^m \mid \overbrace{b^\top y < 0}^{\text{conditions (2.3)}}, \overbrace{\forall j \in J : A_j^\top y \geq 0}^{} \} \tag{2.5}$$

such that, in any iteration, the improving direction chosen on line 5 in Algorithm 2.1 is from the set $\mathcal{Y}(\ (y))$ where y is the current dual-feasible solution. Note that $\mathcal{Y}(J) = ;$ if (2.2) is feasible because then (2.3) is infeasible. Also, if we are unable to prove infeasibility of (2.2) for some J, then one can set $\mathcal{Y}(J) = ;$.

In particular, if the way of constructing the improving direction on line 5 in Algorithm 2.1 is deterministic and depends only on J, then each set $\mathcal{Y}(J)$ is either empty or a singleton, so $\mathcal{Y} = \bigcup_{J \ [n]} \mathcal{Y}(J)$ satis es condition (a) in Theorem 2.1.

Next, in order to guarantee the existence of a positive lower bound on the step size, we introduce a heuristic analogous to capacity scaling in maximum ow algorithms [102, §7.3]. This heuristic was used to guarantee niteness of the Augmenting DAG algorithm [95, 146], VAC algorithm [33, §11.1][40], and later in [148, Equation (21)] and [48a, §3.1.7] in a more general setting of minimizing unconstrained convex piecewise-a ne functions.

Formally, we consider the set of dual constraints which are `almost' active. For this purpose, we de ne

$$(y) = \{ j \ 2 \ [n] \ j \ A_j^{>} y \quad c_j + \quad g \tag{2.6}$$

where $\quad 0$.

If (y) is replaced by (y) on line 2 in Algorithm 2.1, the scheme remains valid. In detail, if (2.2) is infeasible for $J = (y)$, then it is also infeasible for $J = (y)$ due to $(y) \quad (y)$ for any $\quad 0$. This is equivalent to the fact that any y feasible for (2.3) with $J = (y)$ is also feasible for (2.3) with $J = (y)$.

Next, we prove that, if the computed step size on line 6 of Algorithm 2.1 is optimal and (y) on line 2 is replaced by (y) for some constant > 0, then there exists a positive lower bound on the computed step sizes. Thus, condition (b) in Theorem 2.1 is implied by the aforementioned conditions which yields the next theorem.

Theorem 2.2. Let the dual (1.1) be feasible and bounded. Algorithm 2.1 terminates after a nite number of iterations (i.e., updates of y) provided that:

(a) there exists a nite set $\mathcal{Y} \quad R^m$ such that in every iteration, improving direction y found on line 5 belongs to $\mathcal{Y}$,

(b) (y) on line 2 is replaced by (y) where > 0 is a constant,

(c) the step size computed on line 6 is optimal, i.e., $=$ where is (2.4).

Proof. We prove that there exists $_{min} > 0$ such that for any computed by the algorithm (with the modi cations assumed in this theorem), it holds that $_{min}$. The claim will then follow from Theorem 2.1. Analogously to the proof of Theorem 2.1, we assume (without loss of generality) that for any $y \ 2 \ \mathcal{Y}$, there exists dual-feasibley such that y satis es (2.3) for $J = (y)$.

Let us de ne

$$= \max_{y2\mathcal{Y}} \ \max_{\substack{j \ 2 [n] \\ A_j^{>} y < 0}} \quad A_j^{>} y \tag{2.7}$$

so that $\quad A_j^{>} y > 0$ for any y used by Algorithm 2.1 and any j considered in de nition of step size (2.4). Clearly, is positive and well-de ned because the maxima are always taken over nite sets. The case with $\mathcal{Y} = ;$ is treated as in the proof of Theorem 2.1. On

[40] We comment on capacity scaling used in VAC in more detail in §2.3.

```
   inputs:  instance of problem (1.1), dual-feasible solution y, initial  > 0,
            constraint propagation rules for (2.2).
1  while  is not small enough or time limit is not reached do
2  |  Improve y by Algorithm 2.1 with  (y) replaced by  (y) on line 2.
3  |  Decrease  while keeping  > 0 (e.g.,  :=  =10).
4  return  y
```

Algorithm 2.2: Approximate optimization of the dual (1.1) with gradually decreasing .

the other hand, if 6= ;, but for all y 2 , we have 8j 2 [n]: $A_j^\top y \geq 0$, then the dual is unbounded (see Footnote 38), which violates our assumption on its boundedness.

Now, we claim that $_{\min} = =$ is positive (because > 0 and > 0) and constitutes a lower bound on any step size computed by the algorithm. For this, let y be any feasible solution for the dual and y 2 satisfy (2.3) for J = (y). We claim that $_{\min}$, i.e., for all j 2 [n] with $A_j^\top y < 0$, $(c_j - A_j^\top y)/(A_j^\top y) \geq _{\min}$. Indeed, as discussed in the proof of Proposition 2.1, if $A_j^\top y < 0$, then j 2 J = (y) due to (2.3b), so $A_j^\top y - c_j >$ by definition of (y). Combined with $A_j^\top y$, the claim follows. □

Note, instead of using a single fixed > 0, one can run the iterative algorithm multiple times, each time with a lower value of , thus gradually improving the current feasible solution y even further, as it was done in [33, §11.1] or later in [48a, Algorithm 14]. In this way, we obtain an anytime algorithmic scheme outlined in Algorithm 2.2. It is experimentally observed that this modification results in larger step sizes and faster decrease of the objective [33, 48a], which is why it was utilized in our implementations [52a, 55a] that will be described later in §2.4 and §3.

In the following sections, we exemplify Algorithm 2.1 on LP relaxations of two combinatorial problems. In detail, in §2.3, we show that the VAC algorithm [33] is its special case and, in §2.4, we newly apply the approach to the LP relaxation of weighted Max-SAT.

2.3 Example: Basic LP Relaxation and Arc Consistency

In §1.5.4.2, we mentioned the VAC algorithm. Using notation from §1.4 and §1.5, we will focus on this algorithm here in detail and proceed to show that it is in a precise sense subsumed by our previously outlined iterative scheme.

To this end, suppose that we aim to compute an upper bound on WCSP$_g$ 2 R^T by approximately minimizing (1.43) over variables ' . Recall from §1.5.4 that non-empty AC closure is necessary (but not sufficient) for optimality of ' for (1.43). Up to technical details (see Remark 1.12), the VAC algorithm improves a current solution ' of (1.43) by enforcing AC in the CSP A ($g^{'}$). In detail, if the AC closure of A ($g^{'}$) is empty, the algorithm traces the AC operations in anti-chronological order and changes the values of some components of to improve the objective. If the AC closure of A ($g^{'}$) turns out to be non-empty, the algorithm terminates.

By complementary slackness [146, 149, 119], is optimal for (1.43) if and only if there exists feasible for (1.44) such that $_t = 0$ for all t 2 T A ($g^{'}$). Written explicitly, this

is

$$\sum_{\ell \in D^S : \ell_i = k} \lambda_S(\ell) - \lambda_{\{i\}}(k) = 0 \qquad \forall S \in C_2,\ i \in S,\ k \in D \qquad (2.8a)$$

$$\sum_{k \in D} \lambda_{\{i\}}(k) = 1 \qquad \forall i \in V \qquad (2.8b)$$

$$\lambda_t \geq 0 \qquad \forall t \in A(g') \qquad (2.8c)$$

$$\lambda_t = 0 \qquad \forall t \in T \setminus A(g') \qquad (2.8d)$$

which can be interpreted as an LP relaxation of CSPA(g') [146]. For $t = (S, k) \in T$, we write $\lambda_S(k)$ and λ_t interchangeably, analogously to the notation used in §1.5.

Remark 2.4. Observe that the basic LP relaxation (1.44) is a linear program in the form of the primal (1.1), so complementary slackness conditions (2.8) are a special case of (2.2). The Farkas alternative system to (2.8) is therefore (2.3). Note that we changed (1.44b) to (2.8a) so that matrix A in (2.3) and (1.1) is defined unambiguously.

We will now apply a certain propagation rule to the system (2.8), in analogy to what was explained in §2.1. This propagation rule will be inferring zero values of the primal variables λ in (2.8) using the marginalization constraint (2.8a). Next, we will show how to formally infer this by deriving new equalities implied by system (2.8) and also argue that this process is equivalent to enforcing AC in the CSPA(g').

On a high level, proving that variable λ_t is zero for some $t \in T$ will be done indirectly [41] by inferring an equality $\lambda_t + p_t = 0$ where

$$p_t = \sum_{t' \in T} \alpha_{t'} \lambda_{t'} \quad \text{where constants } \alpha_{t'} \text{ satisfy } \begin{cases} \alpha_{t'} \geq 0 & \text{if } t' \in A(g') \\ \alpha_{t'} \in \mathbb{R} & \text{if } t' \notin A(g') \end{cases}. \qquad (2.9)$$

Due to constraints (2.8c)-(2.8d) we have $p_t \geq 0$, so $\lambda_t + p_t = 0$ implies $\lambda_t = 0$. The equality $\lambda_t + p_t = 0$ will be obtained as a linear combination of the equalities (2.8a)-(2.8b) and the coefficients of this linear combination will form its cause vector y^t. Eventually, if for some $i \in V$ we infer that variables $\lambda_{\{i\}}(k)$ are zero for all $k \in D$, this is contradictory with (2.8b) and we are able to construct a certificate y of infeasibility of system (2.8). Such a certificate satisfies the corresponding (recall Remark 2.4) system (2.3).

We now describe the propagation rules in detail, including the computation of cause vectors y^t. We note that all cause vectors y^t have the same dimension, equal to the number of constraints (2.8a)-(2.8b). We denote the standard-basis vector of this space corresponding to (2.8a) and (2.8b) by $e^{S,i,k}$ and e^i, respectively.

For $t \notin A(g')$, y^t is initialized to be the zero vector that thus represents equality $0 = 0$; indeed, this corresponds to the form $\lambda_t + p_t = 0$ as one can interpret it as $\lambda_t - \lambda_t = 0$ where $p_t = -\lambda_t$ conforms to (2.9). If $t \in A(g')$, y^t will be a non-zero vector representing an equality $\lambda_t + p_t = 0$, as discussed earlier. The propagation rules are as follows:

[41] In theory, we might infer directly $\lambda_t = 0$ if (1.44d) (and (2.8c)) were seen as constraints over real variables λ (instead of interpreting (1.44d) as the definition of non-negative variables λ). This change would introduce additional dual variables and the derivation would be slightly more technical and involved.

- If $\lambda_{fig}(k) = 0$ for some $i \in V$ and $k \in D$, constraints (2.8a) (together with non-negativity of λ variables) imply $\lambda_S(\ell) = 0$ for all $S \in \mathcal{C}_2$ and $\ell \in D^S$ such that $i \in S$ and $\ell_i = k$. For every such (S, ℓ), we can infer this formally as:

$$\underbrace{\lambda_{fig}(k) + p_{fig}(k) = 0}_{\text{equality certifying } \lambda_{fig}(k) = 0} + \sum_{\substack{\ell \in D^S \\ \ell_i = k}} \underbrace{\lambda_S(\ell)}_{\text{marginalization constraint (2.8a)}} \underbrace{\lambda_{fig}(k) = 0}_{} = \underbrace{\sum_{\substack{\ell \in D^S \\ \ell_i = k}} \lambda_S(\ell) + p_{fig}(k) = 0}_{\text{new inferred equality}} :$$

  Let us now fix a single tuple $t = (S, \ell)$ satisfying the conditions above. The new inferred equality can be simply transformed into the form $\lambda_t + p_t = 0$ by moving the other λ variables into the term p_t. Assuming that $p_{fig}(k)$ is in the form (2.9), p_t is in such a form too. The cause vector of this equality is $y^t = y^{(fig;k)} + e^{S;i;k}$. Note the similarity of this inference rule to the third case in the definition of the AC propagator (1.32).

- If for some $S \in \mathcal{C}_2$, $i \in V$, and $k \in D$ we have $\lambda_S(\ell) = 0$ for all $\ell \in D^S$ with $\ell_i = k$, constraint (2.8a) implies $\lambda_{fig}(k) = 0$. Inference in terms of equalities:

$$\sum_{\substack{\ell \in D^S \\ \ell_i = k}} \underbrace{\lambda_S(\ell) + p_S(\ell) = 0}_{\text{equality certifying } \lambda_S(\ell) = 0} \underbrace{\lambda_S(\ell)}_{\text{marginalization constraint (2.8a)}} \underbrace{\lambda_{fig}(k) = 0}_{} = \underbrace{\lambda_{fig}(k) + \sum_{\substack{\ell \in D^S \\ \ell_i = k}} p_S(\ell) = 0}_{\text{new inferred equality}} :$$

  The new inferred equality is clearly in the form $\lambda_t + p_t = 0$ for $t = (fig; k)$ where $p_t = \sum_{\substack{\ell \in D^S \\ \ell_i = k}} p_S(\ell)$ is in the form (2.9) if this is the case for each $p_S(\ell)$. The cause vector of the new inferred equality is $y^t = \sum_{\substack{\ell \in D^S \\ \ell_i = k}} y^{(S;\ell)} - e^{S;i;k}$. Again, one can notice the similarity of this inference rule to the second case in the definition of the AC propagator (1.32).

- If for some $i \in V$ we have $\lambda_{fig}(k) = 0$ for all $k \in D$, constraint (2.8b) is contradictory (domain wipeout). Inference in terms of equalities:

$$\sum_{k \in D} \underbrace{\lambda_{fig}(k) + p_{fig}(k) = 0}_{\text{equality certifying } \lambda_{fig}(k) = 0} \underbrace{\sum_{k \in D} \lambda_{fig}(k) = 1}_{\text{simplex constraint (2.8b)}} = \underbrace{\sum_{k \in D} p_{fig}(k) = -1}_{\text{inferred contradiction}} :$$

  The inferred equality is contradictory since every term p_t in the form (2.9) is non-negative under conditions (2.8c)-(2.8d). The cause vector of this contradictory equality is $y = \sum_{k \in D} y^{(fig;k)} - e^i$.

By properties of p_t and the fact that y encodes an equality in the form $\sum_{t \in T_{fig}} p_t = -1$, it is not hard to show that y is feasible for the Farkas alternative system (2.3) (recall Remark 2.4). In particular, the inferred contradictory equality is $\sum_{t \in T} {}_t \lambda_t = -1$ where ${}_t \geq 0$ for all $t \in A$ (g') by (2.9). Since the vector y stores the coefficients with which this equality was inferred, this inequality is in fact $y^\top A \lambda = y^\top b$ where $y^\top b = -1$ and $y^\top A = {}^\top$.

Consequently, y certifies infeasibility of (2.8) and constitutes an improving direction for the dual linear program to (1.44). However, let us note a subtlety. Strictly speaking,

optimization problem (1.43) is not the dual linear program to (1.44) because it is obtained by eliminating some of the dual variables (cf. §1.1.2), hence a subvector of y (where only the components corresponding to the θ variables are present[42]) is an improving direction for (1.43) from the current point θ'.

The described algorithm is `almost' equivalent to the VAC / Augmenting DAG algorithm [33, 95, 146]. The stopping points of both algorithms are characterized by the same property, namely non-empty AC closure of $\mathbb{A}(g')$. However, improving directions y constructed as above are in general different from the ones in [33, 95, 146], sometimes having larger absolute values of their components (and thus possibly allowing smaller step size α). The reason is that our algorithm does not take into account the values of the individual coefficients in the cause vectors y^t. A finer-grained version is possible, given that the cause vectors are computed in an anti-chronological order after the contradiction is detected instead of fixing their values already when inferring individual equalities. Such an approach would be analogous to what we described in [54a, Appendix B] or what will be explained later in §3.2.2.2.

It is known [33, Appendix A] that the VAC algorithm without capacity scaling can enter an infinite loop. In order to avoid this, it was proposed [33, §11.1] to replace the CSP $\mathbb{A}(g')$ by the CSP $\mathbb{A}_\varepsilon(g')$ formed by `almost' active tuples. In our notation[43], this reads

$$\mathbb{A}_\varepsilon(f) = \left\{ (S,k) \in T \;\middle|\; f_S(k) + \max_{\ell \in D^S} f_S(\ell) < \varepsilon \right\} \tag{2.10}$$

where $\varepsilon \geq 0$.

Despite finiteness of the VAC algorithm was already ensured in [33], we would like to point out that Theorem 2.2 is easily applicable here. First, under our assumption on finite weights in WCSP g (see §1.5), the basic LP relaxation (1.44) is clearly feasible and bounded. By strong duality, the dual is also feasible and bounded. Second, despite there might be many orders in which the AC operations can be applied to the CSP $\mathbb{A}(g')$, the set of tuples T is finite, so there are only finitely many ways in which the AC operations can be applied to each CSP. For each such order, the improving direction y is constructed deterministically (both by the VAC algorithm or by our propagation rules above). Finally, using $\varepsilon > 0$ and optimal step size, both of these approaches will terminate in finite time.

Remark 2.5. Since non-empty AC closure of $\mathbb{A}(g')$ is in general not sufficient for optimality of θ' for (1.43) [146, §5], applying the aforementioned rules to problem (2.8) is a refutation-incomplete method. In other words, these rules may not detect infeasibility of (2.8) in some cases. Consequently, VAC (or the algorithm just sketched above) may terminate in a non-optimal point θ'.

2.4 Example: LP Relaxation of Weighted Max-SAT

In this section, we outline how the iterative scheme based on constraint propagation from §2.2 applies to a problem different from WCSP. In particular, we focus on the

[42] To be precise, recall that the components of y are in one-to-one correspondence with the constraints (2.8a)-(2.8b). Partitioning y into two vectors, θ' (whose components correspond to constraints (2.8a)) and z (whose components correspond to (2.8b)), yields the improving direction θ' for (1.43) from the current point θ'.

[43] CSP $\mathbb{A}(\cdot)$ is denoted by Bool $(\cdot)$ in [33, §11.1], by $O(\cdot)$ in [134, Appendix B.1], and by mi $[\cdot]$ in [119, §6.2.4].

weighted Max-SAT problem. We note that the symbols V and C have a different meaning in this part than what was considered in the (W)CSP setting (in §1.4, §1.5, and §2.3).

In the weighted Max-SAT problem [138, §16, 21, §19.2], we are given a finite set V of Boolean variables and a finite set C of clauses with positive weights $w \in \mathbb{R}^C_{++}$. The task is to find an assignment to the Boolean variables such that the sum of the weights of satisfied clauses is maximized. Let V_c^+ and V_c^- denote the set of variables that occur in clause $c \in C$ non-negated and negated, respectively. Let $C_i = \{ c \in C \mid i \in V_c^+ \text{ or } i \in V_c^- \}$ denote the set of clauses where variable $i \in V$ occurs non-negated/negated. We denote the number of negated variables in a clause $c \in C$ by $n_c = |V_c^-|$ so that $n \in \mathbb{N}_0^C$. For any $S \subseteq V$, we will use the shortcut $x(S) = \sum_{i \in S} x_i$.[44]

The classical LP relaxation of weighted Max-SAT [138, §16.3, 21, §19.2] is the left-hand problem of the primal-dual pair

$$\begin{aligned}
&\max w^\top z && && \min n^\top y + q(C) + p(V) && && (2.11a)\\
&z_c \le x(V_c^+) + n_c - x(V_c^-) && && y_c \ge 0 && \forall c \in C && (2.11b)\\
&x_i \ge 0 && p_i \quad && y(C_i^+) + y(C_i^-) \ge 0 && \forall i \in V && (2.11c)\\
&x_i \le 1 && && p_i \ge 0 && \forall i \in V && (2.11d)\\
&z_c \ge 0 && && q_c + y_c \ge w_c && \forall c \in C && (2.11e)\\
&z_c \le 1 && && q_c \ge 0 && \forall c \in C. && (2.11f)
\end{aligned}$$

As with (1.1), we will refer to the left-hand problem (2.11) as primal and to the right-hand problem (2.11) as dual. This LP relaxation was shown to be as hard to solve as any linear program [114].

Note that the primal variables x_i represent the (relaxed) original Boolean variables. To better understand the meaning of the primal constraint (2.11b), it is easy to verify that for any $c \in C$ and $x \in \{0,1\}^V$, the expression

$$x(V_c^+) + n_c - x(V_c^-) = \sum_{i \in V_c^+} x_i + \sum_{i \in V_c^-} (1 - x_i) \tag{2.12}$$

is zero if and only if clause c is not satisfied by the corresponding assignment of truth values and is at least 1 otherwise. Thus, for any fixed assignment $x \in \{0,1\}^V$, setting the auxiliary z variables as large as possible (while maintaining feasibility) results in $z_c = 0$ if clause c is not satisfied and $z_c = 1$ otherwise.

Let us now point our attention to the dual (2.11). Analogously to the notation in the primal, for any subset of clauses $S \subseteq C$, we use the shortcut $y(S) = \sum_{c \in S} y_c$ and similarly for the other dual variables. Clearly (cf. §1.1.2), at dual optimum we have

$$p_i = \max\{ y(C_i^+) - y(C_i^-), 0 \} \qquad \forall i \in V \tag{2.13a}$$

$$q_c = \max\{ w_c - y_c, 0 \} \qquad \forall c \in C. \tag{2.13b}$$

Substituting (2.13) into the dual objective together with $n^\top y = \sum_{i \in V} y(C_i^-)$ results in a simpler form of the dual, namely

$$\min \sum_{c \in C} \max\{ w_c - y_c, 0 \} + \sum_{i \in V} \max\{ y(C_i^+), y(C_i^-) \} \tag{2.14a}$$

$$y_c \ge 0 \qquad \forall c \in C \tag{2.14b}$$

which minimizes a convex piecewise-affine function of non-negative variables.

[44] This shortcut will be used only in sections related to Max-SAT and SAT.

2.4.1 Employing Constraint Propagation

We will now show how the iterative scheme with constraint propagation from §2.2 is applied.

We begin by stating the complementary slackness conditions in terms of the primal variables based on fixed dual variables: a vector $y \in \mathbb{R}^C_+$ is optimal for (2.14) if and only if there exists $x \in \mathbb{R}^V$ satisfying

$$x(V_c^+) + n_c \quad x(V_c) \quad 1 \qquad \forall c \in C_{\quad 1}(y) = \{ c \in C \mid y_c = 0 \} \tag{2.15a}$$
$$x(V_c^+) + n_c \quad x(V_c) = 1 \qquad \forall c \in C_{=1}(y) = \{ c \in C \mid 0 < y_c < w_c \} \tag{2.15b}$$
$$x(V_c^+) + n_c \quad x(V_c) \quad 1 \qquad \forall c \in C_{\quad 1}(y) = \{ c \in C \mid y_c = w_c \} \tag{2.15c}$$
$$x(V_c^+) + n_c \quad x(V_c) = 0 \qquad \forall c \in C_{=0}(y) = \{ c \in C \mid y_c > w_c \} \tag{2.15d}$$
$$x_i = 1 \qquad \forall i \in X_1(y) = \{ i \in V \mid y(C_i^+) > y(C_i) \} \tag{2.15e}$$
$$x_i = 0 \qquad \forall i \in X_0(y) = \{ i \in V \mid y(C_i^+) < y(C_i) \} \tag{2.15f}$$
$$0 \quad x_i \quad 1 \qquad \forall i \in X_U(y) = \{ i \in V \mid y(C_i^+) = y(C_i) \}. \tag{2.15g}$$

We note that the z variables were eliminated from system (2.15) to simplify it, which resulted in the 4 types of constraints (2.15a)-(2.15d). Also, notice that $\{C_{\quad 1}(y), C_{=1}(y), C_{\quad 1}(y), C_{=0}(y)\}$ is a partition of C and $\{X_1(y), X_0(y), X_U(y)\}$ is a partition of V.

We now define propagation rules for system (2.15). These rules set the values of some of the undecided variables x_i, $i \in X_U(y)$ to 0 or 1. Precisely, we iteratively visit each constraint (2.15a)-(2.15d) and look whether with the already decided variables it permits only a single value of some so-far undecided variable. If so, we fix the value of this variable (i.e., make it decided). If some constraint (2.15a)-(2.15d) cannot be satisfied by any assignment subject to the already decided variables, (2.15) is infeasible. During propagation, we store the cause vector for each decided variable, so that if infeasibility is detected, we are able to construct an improving direction y for (2.14).

The propagation rules are listed in Table 2.1, divided into 3 groups based on the types of constraints (2.15a)-(2.15d). For each rule, we also specify how to construct the cause vector y^i for each decided variable x_i. For $i \in X_1(y) \cup X_0(y)$, we define $y^i = 0$ so that it can be referred to in the definition of other cause vectors y^j or y. To simplify the explanation of the rules, for any $c \in C$, we define $V_c = V_c^+ \cup V_c^-$. Moreover, for any $c \in C$ and $i \in V_c$, we denote

$$x_i^c = \begin{cases} x_i; & \text{if } i \in V_c^+; \\ 1 - x_i; & \text{if } i \in V_c^- \end{cases} \tag{2.16}$$

so that, e.g., (2.12) can be written as $\sum_{i \in V_c} x_i^c$. Finally, we define $e^c \in \mathbb{R}^C$ to be the standard-basis vector of $\mathbb{R}^C$ with 1 in the place corresponding to clause c.

Example 2.1. To show how the rules outlined in Table 2.1 are applied, let $C = \{1, 2, 3, 4\}$, $V = \{1, 2, 3, 4, 5\}$, and system (2.15a)-(2.15d) be

$$x_1 + (1 - x_2) = 0 \tag{2.17a}$$
$$x_1 + x_3 \quad 1 \tag{2.17b}$$
$$x_2 + x_4 + (1 - x_5) \quad 1 \tag{2.17c}$$
$$x_3 + x_5 = 1 \tag{2.17d}$$

Applicable to		Rule
$C_{-1}(y)$ and $C_{=1}(y)$	A1	If there is $k \in V_c$ such that x_k is undecided and for all $i \in V_c \setminus \{k\}$, x_i is decided and satisfies $x_i^c = 0$, then we set $x_k = \{k \in V_c^+\}$ and $y^k = e^c + \sum_{i \in V_c \setminus \{k\}} y^i$.
	A2	If for all $i \in V_c$, x_i is decided and satisfies $x_i^c = 0$, then we obtain a contradiction and set $y = e^c + \sum_{i \in V_c} y^i$.
$C_{=1}(y)$ and $C_{-1}(y)$	B1	If there is exactly one $i \in V_c$ such that x_i is decided and $x_i^c = 1$, then we set $x_k = \{k \in V_c\}$ and $y^k = -e^c + y^i$ for all undecided variables x_k, $k \in V_c$.
	B2	If there are two (or more) decided variables x_i, x_j for $i, j \in V_c$ with $x_i^c = x_j^c = 1$, then we obtain a contradiction and set $y = -e^c + y^i + y^j$.
$C_{=0}(y)$	C1	If there is no $i \in V_c$ such that x_i is decided and $x_i^c = 1$, then we set $x_k = \{k \in V_c\}$ and $y^k = -e^c$ for all undecided variables x_k, $k \in V_c$.
	C2	If there is $i \in V_c$ such that x_i is decided and $x_i^c = 1$, then we obtain a contradiction and set $y = -e^c + y^i$.

Table 2.1: Propagation rules for system (2.15). The first column determines the types of constraints to which the rule applies.

where all x variables are initially undecided, i.e., $X_U(y) = V$.

First, it is clear that condition (2.17a) (together with $x_1, x_2 \in [0,1]$) implies $x_1 = 0$ and $x_2 = 1$ by rule C1 and we set $y^1 = y^2 = -e^1$. Second, since $x_1 = 0$, we can apply rule A1 to (2.17b) to infer that $x_3 = 1$ with $y^3 = e^2 + y^1$. Next, because we have $x_2 = 1$, applying rule B1 to (2.17c) results in $x_4 = 0$ and $x_5 = 1$ with $y^4 = y^5 = -e^3 + y^2$. Finally, since $x_3 = 1$ and $x_5 = 1$, rule B2 detects that condition (2.17d) is contradictory and sets $y = -e^4 + y^3 + y^5$. 4

The derivation of the inequalities corresponding to the cause vectors $y^i \in R^C$ is technical and must be done for each rule separately. This is more complicated when compared to the case presented in §2.3 because here we have more types of constraints (which also include inequalities in different directions) and we set the x variables not only to 0, but also to 1, resulting in a larger number of propagation rules.

Remark 2.6. For general weighted Max-SAT, the propagation rules listed in Table 2.1 need not always detect infeasibility of (2.15) and thus are refutation incomplete. As an example, let $V = \{1,2,3\} = X_U(y)$ and (2.15a)-(2.15d) be

$$x_1 + x_2 + x_3 = 1 \tag{2.18a}$$
$$x_1 + x_2 = 1 \tag{2.18b}$$
$$x_1 + x_3 = 1 \tag{2.18c}$$
$$x_2 + x_3 = 1. \tag{2.18d}$$

No rule from Table 2.1 is applicable, but (2.18) is infeasible.

However, for weighted Max-2SAT (i.e., instances where $|V_c| \leq 2$ for all $c \in C$), the rules are refutation complete. In particular, if no more propagation is possible and no

contradiction is detected, setting all undecided variables x_i to $\frac{1}{2}$ satisfies all constraints of (2.15). This is simply verified by case analysis while assuming that none of the listed propagation rules is applicable (cf. more general Lemma 5.7a given later).

Remark 2.7. If any of the propagation rules A1, B1, or C1 is applied to some constraint $c \in C$ from (2.15a)-(2.15d), then all variables x_i, $i \in V_c$ become decided and the corresponding constraint from (2.15a)-(2.15d) is satisfied. Thus, no rule from Table 2.1 is applicable to this constraint anymore. Consequently, after at most $|C|$ rules are applied, one either detects a contradiction or finds out that no more rules are applicable.

Moreover, see that if a single rule sets the values of multiple variables at once (i.e., rules B1 and C1), then the cause vectors for these newly decided variables are identical.

Remark 2.8. One can ask whether it is possible to infer other values of undecided variables than 0 or 1, such as $\frac{1}{2}$. Assuming that inference is done only from a single constraint from (2.15a)-(2.15d), this is impossible because the polyhedron defined by a single (in)equality from (2.15a)-(2.15d) subject to $0 \leq x_i \leq 1$ (where some of the variables may be already set to 0 or 1) has integral vertices. This was proved for constraint in the form (2.15a) in [76, Theorem 45] and the other cases (2.15b)-(2.15d) could be shown analogously (see Lemma 5.5 given later).

2.4.2 Finding Step Size by Approximate Line Search

If a contradiction is detected in (2.15) and improving direction y from the current point y is constructed, we need to find a step size $\alpha > 0$ in order to update $y := y + \alpha y$ as in §2.2. The optimal way (exact line search) would be to minimize the univariate function $g(\alpha) = f(y + \alpha y)$ over $\alpha > 0$ subject to $y + \alpha y \geq 0$ where $f(y)$ is the objective (2.14a). Since this is too costly for large instances, we do only approximate line search: we find the first breakpoint $\alpha > 0$ (i.e., non-differentiable point) of the univariate convex piecewise-affine function g, i.e., the smallest $\alpha > 0$ at which at least one previously inactive affine function becomes active.[45] This value of α may be further reduced to ensure feasibility, i.e., $y + \alpha y \geq 0$. More formally, such α is the maximum number satisfying the following constraints[46]:

- To stay within the feasible set, we need $y_c + \alpha y_c \geq 0$, therefore $\alpha \leq -y_c/y_c$ for all $c \in C$ with $y_c < 0$.
- For terms $\max\{w_c - y_c; 0\}$, if $w_c - y_c > 0$ and $y_c > 0$ (or with both inequalities inverted), then we need $\alpha \leq (w_c - y_c)/y_c$ where the bound is the point where $w_c - (y_c + \alpha y_c) = 0$. So, this is for all $c \in C$ such that $(w_c - y_c)y_c > 0$.
- For terms $\max\{y(C_i^+); y(C_i^-)\}$, if $y(C_i^+) > y(C_i^-)$ and $y(C_i^+) < y(C_i^-)$ (or with both inequalities inverted), we need $\alpha \leq (y(C_i^+) - y(C_i^-))/(y(C_i^-) - y(C_i^+))$ where the bound is the point where the terms equal, i.e., $y(C_i^+) + \alpha y(C_i^+) = y(C_i^-) + \alpha y(C_i^-)$. So, this is for all $i \in V$ with $(y(C_i^+) - y(C_i^-))(y(C_i^-) - y(C_i^+)) > 0$.

By formulating the Farkas alternative system to the complementary slackness conditions and noting the substitution (2.13), it can be shown analogously to Proposition 2.1 that there always exists $\alpha > 0$ satisfying these bounds.

[45] In formalism of §1.1.2, for a convex piecewise-affine function $\sum_{k \in K} \max_{l \in L_k} c_{kl}^\top x + d_{kl}$, an affine function $c_{kl}^\top x + d_{kl}$ is active (for some x) if $c_{kl}^\top x + d_{kl} = \max_{l \in L_k} c_{kl}^\top x + d_{kl}$, cf. Footnote 25.

[46] This choice of α is analogous to the first-hit strategy in [48a, §3.1.4].

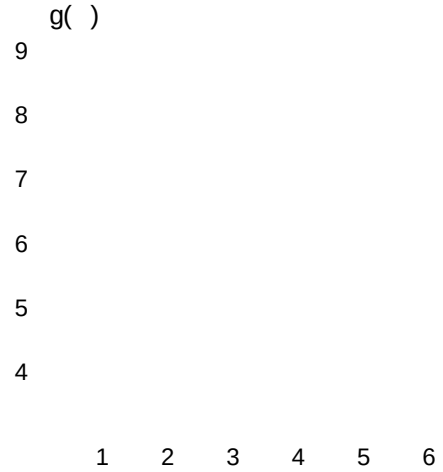


Figure 2.2: Graph of function g from Example 2.2.

Example 2.2. Let $C = f1;2;3g$, $V = f1;2g$, $C_1^+ = f1;2g$, $C_1^- = ;$, $C_2^+ = f1g$, $C_2^- = f2;3g$, and $w = (1;1;1)$. For this setting, the objective (2.14a) is given by

$$\begin{aligned} f(y) = &\max f1 \quad y_1;0g + \max f1 \quad y_2;0g + \max f1 \quad y_3;0g \\ &+ \max f y_1 + y_2;0g + \max f y_1;y_2 + y_3g: \end{aligned} \tag{2.19}$$

Next, let the current point be $y = (0;3;2)$. So, system (2.15a)-(2.15d) reads

$$x_1 + x_2 \quad 1 \tag{2.20a}$$
$$x_1 + (1 \quad x_2) = 0 \tag{2.20b}$$
$$(1 \quad x_2) = 0 \tag{2.20c}$$

and (2.15e)-(2.15f) sets $x_1 = 1$ and $x_2 = 0$. Rule C2 applied to (2.20b) detects contradiction and constructs the improving direction $y = (0; \ 1;0)$. We will now compute step size according to the principle outlined above.

By the first point, we have to ensure $y_2=y_2 = 3$ to satisfy $y + y \quad 0$ because $y_2 < 0$. Since $y_1;y_3 \quad 0$, there are no other bounds given by the first point. Following the second point, we need $(w_2 \quad y_2)=y_2 = 2$ due to $w_2 \quad y_2 < 0$ and $y_2 < 0$. Since $(w_1 \quad y_1)y_1 = (w_3 \quad y_3)y_3 \quad 0$, no other bounds are enforced by the second point. The third point yields conditions 3 and 5.

The maximum number satisfying the above derived upper bounds is $= 2$ and y is therefore updated to $y + y = (0;1;2)$.

For clarity, we show the graph of

$$\begin{aligned} g( ) &= f(y + y) \\ &= 1 + \max f \quad 2;0g + 0 + \max f 3 \quad ;0g + \max f 0;5 \quad g \end{aligned} \tag{2.21}$$

in Figure 2.2. Notice that the bounds given by the second and third point, i.e. $f2;3;5g$, are precisely the breakpoints of function g with > 0. Moreover, see that the chosen step size $= 2$ is the first (in increasing order) breakpoint of g with > 0 and also the point where the affine function 2 from (2.21) becomes active (while it is inactive for < 2).

Based on Figure 2.2, the unique optimal step size is $\operatorname{argmin}_{0<} \ _3 g( ) = f3g$, so the computed step size $= 2$ is not optimal in this case. 4

2.4.3 Algorithm Overview and Implementation Details

Let us summarize the algorithm that follows the general iterative scheme previously shown in Algorithm 2.1. We start with $y = 0$ (which is dual-feasible) and repeat the following iteration: For the current y, construct system (2.15). Apply rules listed in Table 2.1 to x values of undecided variables. During that, construct the DAG defining each y^i (recall §2.1) until no rule is applicable or contradiction is detected. If no contradiction is detected, stop. If contradiction is detected, compute y from the DAG, similarly as in §2.1.1. Calculate step size ε as in §2.4.2 and update $y := y + \varepsilon \Delta y$.

Remark 2.9. System (2.15) can be interpreted as an LP relaxation of a CSP with Boolean variables $x_i \in \{0,1\}$, $i \in V$ and constraints (2.15a)-(2.15f). The propagation corresponds to enforcing AC in this CSP (in the sense of (1.30)). The whole algorithm seeks to find a feasible dual solution of the LP relaxation of weighted Max-SAT that enforces this CSP to have a non-empty AC closure. Compare this with the WCSP case, where (2.8) is an LP relaxation of the CSP formed by the active tuples and the VAC algorithm seeks to find a reparametrization that makes this CSP have a non-empty AC closure. Note that, in contrast to WCSP, there is no obvious analogy of reparametrizations (or equivalent transformations) for weighted Max-SAT.

To speed up the algorithm and ensure finiteness, we use the trick similar to capacity scaling that was introduced in §2.2.1. In particular, we redefine the sets in (2.15) up to a tolerance $\varepsilon > 0$: we replace $y_c > 0$ by $y_c > \varepsilon$, $y(C_i^+) < y(C_i^-)$ by $y(C_i^+) + \varepsilon < y(C_i^-)$, $y_c = w_c$ by $w_c - \varepsilon \le y_c \le w_c + \varepsilon$ etc. We follow the general scheme outlined in Algorithm 2.2 where we initialize $\varepsilon = w(C) = \sum_{c \in C} w_c$ and whenever the algorithm cannot detect infeasibility with the current ε, we keep the current y and update $\varepsilon := \varepsilon/10$. We continue until ε is not very small (10^{-6}).

All data structures used by the algorithm need space that is linear in the input size, i.e., in the number $\sum_{c \in C} |V_c|$ of non-zeros in linear program (2.11). In particular, it can be shown that the DAG (used to calculate Δy) can be conveniently stored as an oriented subgraph of the clause-variable incidence graph.[47] Following Remark 2.7, it is only necessary to store for each $c \in C$ what rule was applied to this clause and a partition of V_c that encodes which variables were decided before the rule was applied and which variables were decided by the rule.

Remark 2.10. We argue that this algorithm terminates after a finite number of iterations (recall Theorem 2.1). First, the primal (2.11) is always feasible and bounded by $w(C)$, so the dual (2.11) is also feasible and bounded. Second, based on Remark 2.7, there are only finitely many options in which the rules from Table 2.1 can be applied to system (2.15) and, for each order in which the rules were applied, the improving direction Δy is defined deterministically. Consequently, there exists a finite set $\mathcal{Y}$ of improving directions used by the algorithm (for each instance). Finally, although the computed step size need not be optimal (for the formulation in terms of the convex piecewise-affine function (2.14)), it can be shown that there exists a positive lower bound on the step size by analyzing the bounds listed in §2.4.2 and noting that $\varepsilon > 0$ (this is analogous to the proof of Theorem 2.2).

[47] The clause-variable incidence graph is the bipartite graph whose nodes correspond to variables V and clauses C. The graph contains an edge between nodes $i \in V$ and $c \in C$ if $i \in V_c$. If V_c is unique for each $c \in C$, then the clause-variable incidence graph is isomorphic to the factor graph of $(V; \{V_c \mid c \in C\})$.

Alternatively, one can apply Theorem 2.2 directly: if we used the dual formulation (2.11) where variables p and q are set as in (2.13), then one could show that the computed step size α is in fact optimal for updating $(p, q, y) := (p, q, y) + \alpha(p, q, y)$ where the improving direction (p, q, y) for the dual (2.11) is obtained in a natural way from the already existing improving direction y.

2.4.4 Experimental Results

We compared the upper bound on the optimal value of (2.11) obtained by our algorithm with the exact optimal value of (2.11) computed by an off-the-shelf LP solver (we used Gurobi [72] with default parameters) on the Max-SAT Evaluations 2018 benchmark [9]. This benchmark contains 2591 instances of weighted Max-SAT. Gurobi was able to optimize (without memory overflow) the smallest 2100 instances, the largest of which had up to 600 thousand clauses, 300 thousand variables and 1.6 million non-zeros[48]. The largest instances in the benchmark have up to 27 million clauses, 19 million variables and 77 million non-zeros and were still manageable by our algorithm.

From the smallest 2100 instances, 154 instances were Max-2SAT and 91 instances did not contain any unit clause. As discussed in Remark 2.6, our algorithm attained the exact optimum of the LP relaxation on instances of Max-2SAT. Similarly, if an instance does not contain any unit clause, then setting $x_i = \frac{1}{2}$, $i \in V$ and $z_c = 1$, $c \in C$ yields an optimal solution of the primal (2.11) with objective value $w(C)$ [76, §13.1.1]. Our algorithm also attains optimality on these instances because $y = 0$ is already optimal for the dual. We exclude these instances from further evaluation.

Each of the remaining 1855 instances contains a clause of length at least 3 and also contains a unit clause, thus the bound provided by our algorithm is not guaranteed to coincide with the optimum of the LP relaxation.

We measure the quality of the bound by two criteria, namely

$$R_1 = \frac{U - \overline{U}}{U} \quad \text{and} \quad R_2 = \frac{U - \overline{U}}{w(C) - \overline{U}} \tag{2.22}$$

where U is the optimal value of (2.11) and $\overline{U}$ is the upper bound computed by our algorithm. Both criteria are invariant to scaling the weights. Criterion R_1 is the relative difference between the optimal value and the provided upper bound whereas criterion R_2 shows how tight the bound is relative to the trivial bound $w(C)$.

The sorted numbers R_1 and R_2 for the 1855 instances are plotted in Figure 2.3. For 802 instances, we obtained $\overline{U} = U$. Due to this, the vertical logarithmic axes in Figures 2.3a and 2.3b are trimmed, starting from 10^{-20}. Based on the linear plots, the obtained upper bound $\overline{U}$ is comparable to U in at least 1000-1100 cases. In detail, R_1 was lower than 10^{-6} and 10^{-8} on 1644 and 1308 from the 1855 instances, respectively, and was always lower than 0.0029. Also, R_2 was higher than 0.6 only in 35 instances.

For 152 out of the 2100 considered instances, the LP relaxation is known to be tight (i.e., its optimal value coincides with the optimal value of the weighted Max-SAT problem). In 133 of them, our algorithm attained this optimum. Only 2 of these were Max-2SAT and each contained a unit clause, so optimality was not guaranteed trivially.

[48] By the number of non-zeros, we mean $\sum_{c \in C} |V_c|$.

(a) R_1 (log scale).

(b) R_2 (log scale).

(c) R_1 (linear scale).

(d) R_2 (linear scale).

Figure 2.3: Sorted values of R_1 (left) and R_2 (right) with linear (down) and logarithmic (up) scale.

An unoptimized implementation of our algorithm was on average 3.3 times faster[49] than Gurobi. We believe a significant speed-up could be achieved by warm-starting. The part of the DAG needed to explain the found contradiction (see §2.1) is usually very small. If the DAG is built in every iteration from scratch, most of it is therefore thrown away. Since the system (2.15) changes only slightly between consecutive updates, it makes sense to reuse a part of the DAG in the next iteration and thus avoid repeatedly applying many rules in the same way. Such a warm-starting was presented for the VAC algorithm in [108] and for the Augmenting DAG algorithm in [145] with significant speed-ups.

2.4.5 Tightness of the Bound on Tractable Max-SAT Classes

We show that our constraint propagation rules in system (2.15) are refutation complete for tractable subclasses of Max-SAT that either use tractable clause types (language) or have acyclic structure (clause-variable incidence graph). For these instances, the LP relaxation (2.11) is tight and any point returned by our algorithm is an optimizer of the LP relaxation.

It was shown in [85, 41] that a subclass of generalized Max-SAT (i.e., Max-CSP with Boolean variables) defined by restricting constraint types (language) is tractable if and only if one of the following holds:

- All constraints are 0-valid or all are 1-valid. In this case, the optimal value is $w(C)$, which coincides with the optimum of the linear program and our algorithm attains this optimum already at $y = 0$.
- All constraints are 2-monotone. Restricting these constraints to clauses results in clauses with at most two literals where at most one is positive and at most one is negative. In this case, Max-SAT can be reduced to minimum st-cut problem [85, Lemma 3, 41] and the optimum of its LP formulation equals (up to a trivial recalculation) the optimum of the LP relaxation of Max-SAT which is thus tight. Since this is an instance of Max-2SAT, any point returned by our algorithm is optimal by Remark 2.6.

Following Remark 2.9, if we view (2.15) as the LP relaxation of a CSP with Boolean variables, then the propagation rules in Table 2.1 enforce AC of this CSP (in the sense of (1.30)). If the factor graph of this CSP is acyclic, AC solves this CSP exactly (recall Example 1.11). Equivalently, if the clause-variable incidence graph (i.e., the factor graph of this CSP) is acyclic, our constraint propagation rules are refutation complete and the points returned by our algorithm are optimal. Additionally, if no contradiction is detected, an integral solution to (2.15) can be constructed, so the LP relaxation is tight.

2.5 Discussion

In this chapter, we reviewed a technique that we originally proposed in [52a] to bound the optimal value of large-scale linear programs. To summarize, given a dual-feasible solution, infeasibility of the complementary slackness conditions (a system of linear inequalities and equalities in the primal variables) is detected by constraint propagation. If the system

[49] The average speed-up of 3.3 is the overall runtime of the LP solver divided by overall runtime of our algorithm. The geometric and arithmetic mean of speed-ups for the individual instances are 4.4 and 19.0, respectively.

is proved to be infeasible, a certificate of infeasibility of this system turns out to be a dual-improving direction that can be used to improve the current solution. In general, the constraint propagation method may be refutation incomplete, hence the feasible solutions returned by the algorithm may not be global optima of the linear program. We do not solve the system given by complementary slackness exactly because this is not practical for large instances.

Although constraint propagation for systems of constraints (here, linear inequalities and equalities) with continuous variables has been studied [13], our novelty lies in constructing the infeasibility certificate which constitutes a dual-improving direction.

This technique can be seen as a generalization of the VAC / Augmenting DAG algorithm [33, 95, 146] for WCSP. Newly, we applied it to the LP relaxation of weighted Max-SAT. Recall from §1.5.4 that the main purpose of (soft) local consistencies in WCSP, such as EDAC [43], FDAC, DAC [37, 98], or OSAC [38], is to bound the optimal value of WCSP during search. Each local consistency has a different trade-off point between bound tightness and computational complexity. In this view, our approach can be seen as a (soft) local consistency technique for other problems than WCSP.

Though in principle our approach can be also applied to other linear programs (if an initial dual-feasible solution is available), the quality of the obtained bounds depends on the possibility to design suitable (possibly problem-dependent) propagation rules.

The propagation rules that we used in §2.3 and §2.4 can be interpreted as examples of a single general rule that infers whether some inequalities in the system given by complementary slackness are always active (recall Definition 1.2). Indeed, in (2.8), we inferred whether some of the constraints (2.8c) are always active (i.e., if some of the primal variables are implied to be zero). Similarly, in case of (2.15), we inferred whether some variables x_i are implied to be 0 or 1, which corresponds to one of the inequalities in (2.15g) being always active. We will precisely define and thoroughly analyze this general propagation rule, which is applicable to any linear program, later in §4. In §4, we also show that the fixed points of BCD are related to the stopping points of Algorithm 2.1 with this rule. Before we do that, we show another application of our approach from §2.2 in §3.

Chapter 3

Bounds on Weighted CSP Using Constraint Propagation and Super-Reparametrizations

As we reviewed in §1.5, a popular approach for obtaining bounds on the optimal value of a WCSP is to compute a feasible (ideally optimal) solution of its dual LP relaxation. For large instances, such solutions are obtained by methods based on BCD (§1.5.4.1) or constraint propagation (§1.5.4.2) whose stopping points are typically characterized by local consistencies of the active-tuple CSP of the reparametrized WCSP. This approach is limited in that it cannot enforce an arbitrary level of local consistency, unless new weight functions are introduced. Moreover, the methods for improving the bound by enforcing local consistencies in the active-tuple CSP needed to be specifically designed based on the chosen kind of local consistency.

In contrast, in this chapter, we propose a method that is able to improve the bound on the WCSP optimal value using any kind of constraint propagation without introducing new weight functions. To this end, we recall from §1.5.5 the problem of minimizing an upper bound on the optimal value of a WCSP over its super-reparametrizations (1.45) and show that it can be approximately optimized using any method that can (at least sometimes) detect unsatisfiability of a CSP. On the other hand, a super-reparametrization of a WCSP need not preserve the objective values of the individual assignments or even the set of optimal assignments, but, as we will show, it is capable of providing a (possibly tighter) bound on the optimal value.

Super-reparametrizations were not utilized nor analyzed in the literature except for [92] and [125]. Yet, they are mentioned only briefly in [125] where the main focus is on reparametrizations. Thus, to fill in this gap, we also provide additional theoretical results connected to the optimization problem (1.45) and to super-reparametrizations.

Optimization problem (1.45) has an exponential number of constraints (1.45b), equal to the number of assignments, i.e. $|D^V|$. However, the suitable structure of this problem allows us to apply the approach that we outlined in §2.2. Indeed, the method that we present in this chapter can be interpreted as an instantiation of the previously shown Algorithm 2.1. However, we chose not to include this method as another example in §2 because it is more involved and its description is extensive.

The structure of this chapter is as follows. We begin in §3.1 by extending the notation that was previously defined in §1.5 and use it to state the necessary and sufficient conditions of optimality for the problem (1.45). Next, we present our approach for approximate minimization of the upper bound using constraint propagation in §3.2. After that, we theoretically analyze the optimization problem and also identify further properties of the active-tuple CSPs and the sets of optimal (and also non-optimal) super-reparametrizations in §3.3. Unsurprisingly, we prove in §3.4 that some decision problems connected to our approach and super-reparametrizations are NP-complete.

This chapter contains (in some places reformulated or rewritten) text and figures from

the submitted journal version [56a] of our earlier conference paper [55a].

3.1 Notation and Optimality Conditions

Throughout this chapter, we will use the notation for CSPs and WCSPs that was defined in §1.4 and §1.5. Nevertheless, to concisely explain our approach, we additionally define the set

$$M^{\infty} = \{ d \in \mathbb{R}^{\mathcal{T}} \mid F(x \mid d) = 0 \ \forall x \in D^{V} \}, \tag{3.1}$$

i.e., M^{∞} is the set of WCSPs whose objective values are zero for all assignments (called zero problems in [146]). See that the set M^{∞} is defined only by the structure (V, D, C). An example of a WCSP belonging to M^{∞} is in Figure 1.6b.

By linearity of $F(x \mid \cdot)$ (see §1.5.1), set M^{∞} is defined as the solution set of a system of homogeneous linear equalities and thus constitutes a linear subspace of $\mathbb{R}^{\mathcal{T}}$. Moreover, it is clear that WCSP f is a reparametrization of WCSP g if and only if $f - g \in M^{\infty}$ due to $F(x \mid f) = F(x \mid g) \Leftrightarrow F(x \mid f - g) = 0$ for all $x \in D^{V}$. The affine subspace of all reparametrizations of g is thus[50] $g + M^{\infty} = \{ g + d \mid d \in M^{\infty} \}$ and the optimization problem of minimizing the upper bound over reparametrizations (1.40) can be stated as

$$\min \{ B(f) \mid f \text{ is a reparametrization of } g \} = \min \{ B(f) \mid f \in g + M^{\infty} \}. \tag{3.2}$$

Analogously, we define the set of all WCSPs (with the fixed structure) whose objective values are non-negative for all assignments, i.e.,

$$M = \{ d \in \mathbb{R}^{\mathcal{T}} \mid F(x \mid d) \geq 0 \ \forall x \in D^{V} \}. \tag{3.3}$$

Set M is a polyhedral convex cone which is however not pointed (i.e., it contains a line [24, §2.4]) because $M^{\infty} \subseteq M$ and the subspace M^{∞} is non-trivial (assuming $|V| > 1$). For a given $d \in \mathbb{R}^{\mathcal{T}}$, deciding whether $d \in M$ is NP-complete, as we show later in Corollary 3.2.

Again, WCSP f is a super-reparametrization of WCSP g if and only if $f - g \in M$. Therefore, the set of all super-reparametrizations of g is the translated cone $g + M = \{ g + d \mid d \in M \}$. The binary relation `is a super-reparametrization of' (on the set of WCSPs with a fixed structure) is reflexive and transitive, hence a preorder. It is not anti-symmetric: $f - g \in M$ and $g - f \in M$ does not imply $f = g$ but merely $f - g \in M^{\infty}$, i.e., that f is a reparametrization of g. This is because the cone M is not pointed (see [79, §2] and [24, §2.4]).

Remark 3.1. To explain our notation for the sets M^{∞} and M, recall the mapping from (1.34) and the set M from (1.37). By expressing the set M^{∞} as

$$M^{\infty} = \{ d \in \mathbb{R}^{\mathcal{T}} \mid \langle d, \varphi \rangle (x) = 0 \ \forall x \in D^{V} \} = \{ d \in \mathbb{R}^{\mathcal{T}} \mid \langle d, \varphi \rangle = 0 \ \forall \varphi \in M \}, \tag{3.4}$$

it becomes clear that $M^{\infty} \subseteq \mathbb{R}^{\mathcal{T}}$ is the orthogonal space [156, §1.1] of the set $M \subseteq \mathbb{R}^{\mathcal{T}}$.

Similarly, we have that

$$M = \{ d \in \mathbb{R}^{\mathcal{T}} \mid \langle d, \varphi \rangle (x) \geq 0 \ \forall x \in D^{V} \} = \{ d \in \mathbb{R}^{\mathcal{T}} \mid \langle d, \varphi \rangle \geq 0 \ \forall \varphi \in M \} \tag{3.5}$$

is the dual cone [156, §1.1] to M. It is easy to show that

$$M = (\operatorname{conv} M)^{*} = (\operatorname{cone} M)^{*} \tag{3.6}$$

[50] Note, the symbol `+' in the expression $g + M^{\infty}$ denotes the sum of a vector and a set of vectors.

where cone denotes the conic hull operator and * the dual cone operator [24, §2.1.5 and §2.6.1, 10, §1]. Thus, M^* can be also seen as the dual cone to the marginal polytope. We made this observation in [56a, Remark 1] and, to the best of our knowledge, this has not been mentioned before.

Using the set M^*, one can rewrite the optimization problem (1.45) as

$$\min\{B(f) \mid f \text{ is a super-reparametrization of } g\} = \min\{B(f) \mid f \in g + M^*\}. \quad (3.7)$$

It is important to note that every f feasible for (3.7) (i.e., every super-reparametrization of g) satisfies

$$B(f) \geq F(x|f) \geq F(x|g) \qquad \forall x \in D^V; \quad (3.8)$$

and thus provides an upper bound on the optimal value of g. This is also an immediate consequence of the previously given Theorem 1.16.

The next theorem characterizes optimal solutions of (3.7).

Theorem 3.1. Let f be feasible for (3.7). The following are equivalent:

(a) f is optimal for (3.7),

(b) $B(f) = \max_{x \in D^V} F(x|f) = \max_{x \in D^V} F(x|g)$,

(c) CSP $A(f)$ has a solution x satisfying $F(x|f) = F(x|g)$.

Proof. (a) $\Leftrightarrow$ (b): This is a corollary of Theorem 1.16 together with (3.8).

(b) $\Rightarrow$ (c): Since every feasible f satisfies (3.8), (b) implies $B(f) = F(x|f) = F(x|g)$ for some x. By Theorem 1.15b, this implies (c).

(c) $\Rightarrow$ (b): By Theorem 1.15b together with (3.8), (c) implies $B(f) = F(x|f) = F(x|g)$ for some x. Statement (b) now follows from (3.8). □

We remark that for $f \in g + M^*$, deciding whether f is optimal for (3.7) is NP-complete, as we discuss later in Corollary 3.3. Although one part of Theorem 3.1 was already proved in [92, Theorem 1] (reviewed in §1.5.5, see Theorem 1.16), it is crucial to identify statement (c) in Theorem 3.1 as it has a simple but useful consequence:

Theorem 3.2. Let $g \in R^T$. CSP $A(g)$ is satisfiable if and only if $B(g) \geq B(f)$ for every $f \in g + M^*$.

Proof. By Theorem 3.1, $A(g)$ is satisfiable if and only if (3.7) attains its optimum at the point $f = g$, i.e., $B(g) \geq B(f)$ for every $f \in g + M^*$. □

3.2 Iterative Method to Improve the Bound

In this section, we present an iterative method for approximately solving (3.7). Starting from a feasible solution to (3.7), every iteration finds a new feasible solution with a lower objective, which by (3.8) corresponds to decreasing the upper bound on the optimal value of the initial WCSP.

```
   input:  WCSP g ∈ R^T.
1  Initialize f^0 := g, k := 0.
2  while CSP A(f^k) is unsatisfiable do
3  |   Find f^{k+1} ∈ f^k + M such that B(f^{k+1}) < B(f^k).
4  |   k := k + 1
5  return B(f^k)
```

Algorithm 3.1: Iterative scheme for upper bounding the optimal value of WCSP g assuming the ability to decide satisfiability of a CSP.

3.2.1 Outline of the Method

Consider a WCSP f feasible for (3.7), i.e., $f \in g + M$. By Theorem 3.1, a necessary (but not sufficient) condition for f to be optimal for (3.7) is that CSP $A(f)$ is satisfiable. By Theorem 3.2, $A(f)$ is satisfiable if and only if $B(f) \not\geq B(f')$ for all $f' \in f + M$. In summary, we have the following implications and equivalences:

$$\begin{array}{ccc} f \text{ is optimal for (3.7)} & \Rightarrow & \mathrm{CSP}\ A(f) \text{ is satisfiable} \\ \tilde{w} & & \tilde{w} \\ B(f) \not\geq B(f') \ \forall f' \in g + M & \Leftarrow & B(f) \not\geq B(f') \ \forall f' \in f + M \end{array} \tag{3.9}$$

The left-hand equivalence is just the definition of the optimum of (3.7), the right-hand equivalence is Theorem 3.2, and the top implication follows from Theorem 3.1. The bottom implication independently follows from transitivity of super-reparametrizations, which means 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 $f \in R^T$ feasible for (3.7) (i.e., $f \in g + M$), 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 3.2). By transitivity of super-reparametrizations, such f' is also feasible for (3.7). This suggests an iterative scheme to improve feasible solutions to (3.7) that we outline in Algorithm 3.1 and analyze next. Note that transitivity of super-reparametrizations implies $f^k \in f^0 + M$ for every k, so every f^k is feasible for (3.7) as expected. An example of a single iteration of Algorithm 3.1 is shown in Figure 3.1a and 3.1b.

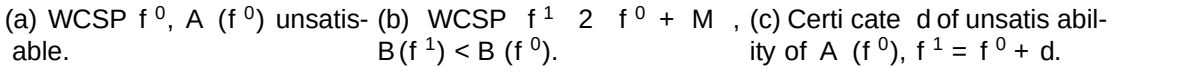
3.2.1.1 Properties of the Method

This iterative scheme can be interpreted as a local search method to (approximately) solve (3.7): 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 problem (3.7) with respect to this method to be super-reparametrizations f of g such that $A(f)$ is satisfiable. [51]

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

$$f^{k+1} + M \subseteq f^k + M \tag{3.10}$$

[51] For clarity, we of course do not apply local search to improve some assignment x w.r.t. the objective of the WCSP. Instead, our variables are the components of the weight vector f and we try to improve $B(f)$ by local search. Furthermore, the local optima mentioned here are different from the ones considered in Definition 1.3.



(a) WCSP f^0, $A(f^0)$ unsatisfiable. (b) WCSP $f^1 \in f^0 + M$, $B(f^1) < B(f^0)$. (c) Certificate of unsatisability of $A(f^0)$, $f^1 = f^0 + d$.

Figure 3.1: Example of one iteration on a pairwise WCSP whose graph $(V; C_2)$ is a cycle of length 4.

which holds with equality if and only if $f^{k+1} \in f^k + M^?$ (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 (3.7) is cut off and becomes forever inaccessible. If, for some k, all global optima of (3.7) happen to lie in the cut-off part, the method has lost any chance to find a global optimum. This is illustrated in Figure 3.2.

This has the following consequence. By Theorems 1.15a and 3.1, every f^k satisfies

$$B(f^k) \le \min_{f \in f^k + M} B(f) = \max_{x \in D^V} F(x \mid f^k). \tag{3.11}$$

In every iteration, the left-hand side of inequality (3.11) decreases and the right-hand side increases or stays the same due to (3.10). If both sides meet for some k, the CSP $A(f^k)$ becomes satisfiable by Theorem 1.15b 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 Figures 3.3 and 3.4. Unlike the case of reparametrizations, note that not every optimal assignment for WCSP $f \in g + M$ is optimal for WCSP g. We will return to this in §3.3.

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 (3.11). Inclusion (3.10) holds with equality if and only if $f^{k+1} \in f^k + M^?$, 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 must recourse to heuristics. One natural heuristic is to choose f^{k+1} such that the vector $f^{k+1} - f^k$ is sparse and its positive components are small. Unfortunately, this can sometimes be too restrictive because, e.g., vectors from $M^?$ can be dense and their components have unbounded magnitudes.[52]

[52] To see this, recall (1.41) where one can set $g = 0$ and choose ε' arbitrarily. Then, g' may be dense and have arbitrarily large components g'_t, yet $g' \in M^?$.

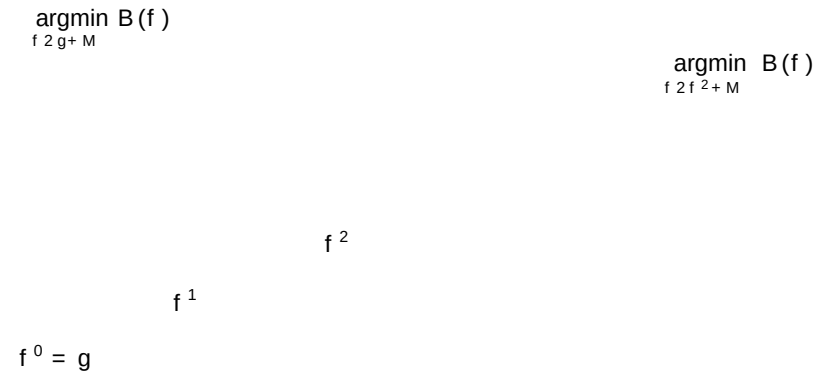


Figure 3.2: The shrinking of the search space of the iterative method. The gure illustrates the translated conesf i + M and several contours of the objectiveB(f). After the second iteration, all global minima of the original problem (marked in gray) become inaccessible as the right-hand side of (3.11) increases. Note that the picture is only illustrative.

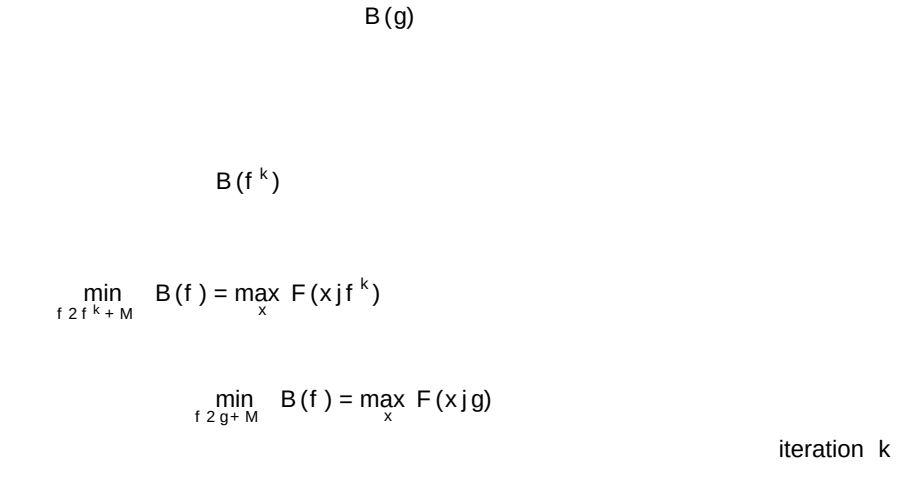


Figure 3.3: Illustration to the iterative scheme: B(g) and B(f k) are shown by the full lines, maxx F(xjg) and maxx F(xjf k) are represented by the dashed lines.

(a) WCSP g (b) WCSP f

Figure 3.4: WCSP f is a super-reparametrization of WCSP g and this pair of WCSPs satisfies $B(f) = 11 < B(g) = 12$ and $\max_{x \in D^V} F(x|f) = 11 > \max_{x \in D^V} F(x|g) = 8$. Assignment $x = (b; b)$ (where b is the lower value, as in Figure 1.6) is not optimal for g even though $B(f) = F(x|f)$.

```
   input: WCSP g ∈ R^T.
1  Initialize f^0 := g, k := 0.
2  repeat
3  |  Try to prove that CSP A(f^k) is unsatisfiable (e.g., using constraint propagation).
4  |  if CSP A(f^k) is proved to be unsatisfiable then
5  |  |  Find f^{k+1} ∈ f^k + M such that B(f^{k+1}) < B(f^k).
6  |  |  k := k + 1
7  |  else
8  |  |  return B(f^k)
```

Algorithm 3.2: Iterative scheme for upper bounding the optimal value of WCSP g using constraint propagation.

3.2.1.2 Employing Constraint Propagation

So far, we assumed that 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 detect unsatisfiability of $A(f)$ only sometimes, e.g., using constraint propagation. We outline this procedure in Algorithm 3.2. In this case, stopping points of the method will be even weaker local minima of (3.7), 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 §3.2.2), 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 (3.7). Second (in §3.2.3), given d and f^k, we find a step size > 0 such that $f^{k+1} = f^k + d$ and $B(f^{k+1}) < B(f^k)$. An example of such a certificate of unsatisfiability is shown in Figure 3.1c.

3.2.1.3 Relation to Existing Approaches

The Augmenting DAG algorithm [95, 146] and the VAC algorithm [33] are (up to the precise way of computing certificates d and step sizes) an example of the described approach, which uses arc consistency to prove unsatisfiability of $A(f^k)$. In this favorable case, there exist certificates $d \in M^?$, so we are, in fact, approximately solving (3.2) rather than (3.7). Such certificates do not generally exist for stronger local consistencies (i.e., inevitably $F(x|d) > 0$ for some x).

The algorithm proposed in [92] 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 (1.33) and minimizing an upper bound over reparametrizations, as in [127, 11, 149, 147, 107]. In our particular case, after each update $f^{k+1} = f^k + d$ we could introduce[53] a new weight function with scope

$$S' = \bigcup \{ S \mid (S,k) \in T,\ d_S(k) < 0 \} \tag{3.12}$$

and weights

$$f_{S'}(k) = \sum_{S \in C,\ S \subseteq S'} d_S\langle k \rangle_S \tag{3.13}$$

where $k \in D^{S'}$. 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 is an example of the iterative scheme to optimize linear programs using constraint propagation from §2.2. In this particular case, if (3.7) is formulated as a linear program, then the complementary slackness conditions (expressed in terms of the dual variables) can be interpreted as the optimality condition (c) stated in Theorem 3.1 expressed as a set of linear equalities with an exponential number of non-negative variables (we will discuss this in detail later in §5.2). Applying constraint propagation to this system is in correspondence with constraint propagation in a CSP.

3.2.2 Certificates of Unsatisfiability of CSP

Recall from §1.4.1 that constraint propagation[54] is an iterative algorithm, which in each iteration (executed by a propagator) infers that some allowed tuples $R \in A$ of a current CSP $A \in T$ 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 or when it becomes explicit that the current CSP is unsatisfiable. The former usually happens when the CSP satisfies some desired local consistency. The latter happens if $A \setminus T_S = \emptyset$ for some $S \in C$, which implies unsatisfiability of A.[55]

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 (3.7), as was mentioned in §3.2.1.2. First, in §3.2.2.1, we introduce a more general concept, deactivating directions. One iteration of constraint propagation constructs an R-deactivating direction for the current CSP A,

[53] Notice that such an added weight function would not increase the bound (1.39) since its weights are non-positive due to the fact that it needs to decrease the objective value of some assignments.

[54] 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).

[55] This is because, as stated in §1.4, each assignment uses exactly one tuple from each scope. In addition, if $A \setminus T_S = \emptyset$ for some $S \in C$ with $|S| = 1$, this is usually called domain wipeout [33, 31].

which certifies that SOL($\mathcal{A}$) = SOL($\mathcal{A}$ ∪ $\mathcal{R}$). Then, in §3.2.2.2, we show how to compose the deactivating directions obtained from individual iterations of constraint propagation into 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.

Remark 3.2. Deactivating directions correspond to cause vectors from §1.1.3 and §2. Indeed, cause vectors certify that a linear inequality is implied by a system of linear inequalities, whereas deactivating directions certify that some tuples can be forbidden in a CSP while its solution set is preserved. As in §2, where cause vectors were combined to compute an improving direction, deactivating directions will be composed to obtain an improving direction here. Although it is also possible to interpret deactivating directions as cause vectors of certain equalities (implied by an LP formulation of a CSP), using CSP terminology without referring to systems of linear (in)equalities simplifies the explanation given here.

3.2.2.1 Deactivating Directions

Definition 3.1. Let $\mathcal{A} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{T})$ and $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{T}')$, $\mathcal{R} \neq \mathcal{A}$, $\mathcal{T}' \subseteq \mathcal{T}$. An $\mathcal{R}$-deactivating direction for CSP $\mathcal{A}$ is a vector $d \in \mathbb{R}^{\mathcal{M}}$ satisfying

(a) $d_t < 0$ for all $t \in \mathcal{R}$,

(b) $d_t = 0$ for all $t \in \mathcal{A} \setminus \mathcal{R}$.

For fixed $\mathcal{A}$ and $\mathcal{R}$, all $\mathcal{R}$-deactivating directions for $\mathcal{A}$ form a convex cone. Moreover, note that if $\mathcal{A} \subseteq \mathcal{A}' \subseteq \mathcal{T}$ and d is an $\mathcal{R}$-deactivating direction for $\mathcal{A}'$, then d is an $\mathcal{R}$-deactivating direction also for $\mathcal{A}$. Taking this observation into account, the following result shows a way how to obtain a particular class of $\mathcal{R}$-deactivating directions for $\mathcal{A}$:

Theorem 3.3. Let $\mathcal{R} \subseteq \mathcal{A} \subseteq \mathcal{A}' \subseteq \mathcal{T}$ be such that SOL($\mathcal{A}'$) = SOL($\mathcal{A}' \setminus \mathcal{R}$) and $\mathcal{R} \neq \emptyset$. Denote[56] $\gamma = |\{ S \in \mathcal{C} \mid \mathcal{T}_S \cap \mathcal{R} \neq \emptyset \}|$. Then, vector $d \in \mathbb{R}^{\mathcal{T}}$ defined by

$$d_t = \begin{cases} -1 & \text{if } t \in \mathcal{R} \\ -\gamma & \text{if } t \in \mathcal{T} \setminus \mathcal{A}' \\ 0 & \text{otherwise (i.e., } t \in \mathcal{A}' \setminus \mathcal{R}) \end{cases} \qquad \forall t \in \mathcal{T} \tag{3.14}$$

is an $\mathcal{R}$-deactivating direction for $\mathcal{A}$.

Proof. Conditions (a) and (b) of Definition 3.1 are clearly satisfied, so it only remains to show that $d \in \mathcal{M}$. We have

$$F(x|d) = \sum_{S \in \mathcal{C}} d_{S[x|_S]} = \sum_{\substack{S \in \mathcal{C} \\ (S,x|_S) \in \mathcal{R}}} -1 + \sum_{\substack{S \in \mathcal{C} \\ (S,x|_S) \in \mathcal{T} \setminus \mathcal{A}'}} -\gamma = -n_1(x) + \gamma\, n_2(x) \tag{3.15}$$

where $n_1(x) = |\{ S \in \mathcal{C} \mid (S, x|_S) \in \mathcal{R} \}|$ and $n_2(x) = |\{ S \in \mathcal{C} \mid (S, x|_S) \in \mathcal{T} \setminus \mathcal{A}' \}|$.

[56] The quantity γ is the number of scopes S such that $\mathcal{T}_S$ contains at least one tuple from $\mathcal{R}$. In other words, for every assignment $x \in \mathcal{D}^V$, $(S, x|_S) \in \mathcal{R}$ holds for at most γ scopes. We remark that the value of γ could be in some cases decreased while (3.14) remains an $\mathcal{R}$-deactivating direction, thus decreasing also the objective values $F(x|d)$. However, deciding whether (3.14) is not an $\mathcal{R}$-deactivating direction for $\mathcal{A}$ for a given value γ is an NP-complete problem (see Theorem 3.13).

For contradiction, let $x \in D^V$ satisfy $F(x|d) < 0$. This implies $n_1(x) > 0$ and $n_2(x) = 0$, where the latter is because $n_1(x)$ by the definition of . That is, we have $\langle S, x_{|S} \rangle \in R$ for some $S \in C$ and $\langle S, x_{|S} \rangle \in A'$ for all $S \in C$. But the latter means $x \in SOL(A')$ and the former implies $x \notin SOL(A' \cap R)$, a contradiction. □

Theorem 3.4. Let $A = (V, D, C)$ and $R \subseteq A$. If there exists an R-deactivating direction for A, then $SOL(A) = SOL(A \cap R)$.

Proof. Recall from §1.4 that SOL is an isotone mapping, so $SOL(A) = SOL(A \cap R)$ is equivalent to $SOL(A) \subseteq SOL(A \cap R)$ because forbidding tuples may only remove solutions.

By contradiction: let d be an R-deactivating direction for A and let $x \in SOL(A) \setminus SOL(A \cap R)$, so $\langle S, x_{|S} \rangle \notin R$ for some $S \in C$. By (1.33), we have $F(x|d) < 0$ because $d_S(x_{|S}) = 0$ for all $\langle S, x_{|S} \rangle \in A \cap R$ by condition (b) in Definition 3.1 and $d_S(x_{|S}) < 0$ for all $\langle S, x_{|S} \rangle \notin R$ by condition (a). This contradicts $d \in M$. □

Combining Theorem 3.3 (with $A' = A$) and Theorem 3.4 yields the following result: for any $R \subseteq A$ with $R \neq \emptyset$, an R-deactivating direction for A exists if and only if $SOL(A) = SOL(A \cap R)$. Thus, any R-deactivating direction for A is a certificate of $SOL(A) = SOL(A \cap R)$.

Unfortunately, vectors d calculated naively from (3.14) with $A' = A$ can have many positive components, which is undesirable as we explained earlier in §3.2.1.1. However, we are allowed to have $A' \supseteq A$ in Theorem 3.3, which gives us some freedom in choosing the deactivating direction. In particular, (3.14) shows that larger sets A' give rise to sparser vectors d – more precisely, vectors d with fewer positive components. This offers us a possibility to obtain a sparser deactivating direction if we can provide a superset $A' \supseteq A$ of the allowed tuples satisfying $SOL(A') = SOL(A' \cap R)$.

Given $A = (V, D, C)$ and $R \subseteq A$, finding a maximal (w.r.t. the partial ordering by set inclusion) superset $A' \supseteq A$ such that $SOL(A') = SOL(A' \cap R)$ is closely related to finding a minimal unsatisfiable core and minimally unsatisfiable set of tuples[57] of an unsatisfiable CSP. While finding a maximal such subset is likely intractable (see [70, 71]), 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 $A' = T \setminus P$ 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 $SOL(A) = SOL(A \cap R)$. Note that P need not be the same for each CSP instance, even for a fixed level of local consistency: for example, if the AC closure of A is empty, then A is unsatisfiable but a domain wipeout 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 (3.14), any other method can be used as long as it satisfies Definition 3.1. We will now give examples of deactivating directions corresponding to some popular constraint propagation rules.

[57] Given an unsatisfiable CSP $A = (V, D, C)$, finding a maximal set $A' \subseteq A$ such that A' is still unsatisfiable corresponds to finding a minimally unsatisfiable set of tuples [70]. This is a finer-grained (tuple-based rather than constraint-based) version of finding a minimal unsatisfiable core of a CSP [71]. Note that we are looking here for a maximal superset A' in contrast to a minimal unsatisfiable core/set of tuples because we define CSP instances by allowed tuples while cores are CSP instances defined by forbidden tuples.

Example 3.1. As already stated in §1.4.1.2, CSP A is arc consistent if for all $S \in C_2$, $i \in S$, and $k \in D$, we have the equivalence

$$(\{i\},k) \in A \iff \exists\, (S,\ell) \in A : \ell_i = k. \tag{3.16}$$

If, for some $S \in C_2$, $i \in S$, and $k \in D$, the left-hand statement in (3.16) is true and the right-hand statement is false, AC propagator (1.32) infers $\mathrm{SOL}(A) = \mathrm{SOL}(A \wedge R)$ where $R = \{(\{i\},k)\}$. To infer this, it suffices to know that the tuples $P = \{(S,\ell) \mid \ell \in D^S,\ \ell_i = k\}$ are all forbidden. An R-deactivating direction d for A can be chosen as in (3.14), where $\gamma = |\{S' \in C \mid T_{S'} \cap R \neq \{\}\}| = 1$ and $A' = T_A \wedge P$. Note that then we have $d \in M^?$.

If the left-hand statement in (3.16) is false and the right-hand statement is true, AC propagator (1.32) infers $\mathrm{SOL}(A) = \mathrm{SOL}(A \wedge R)$ where $R = \{(S,\ell) \mid \ell \in D^S,\ \ell_i = k\} \cap A$. To infer this, it suffices to know that the tuple $P = \{(\{i\},k)\}$ is forbidden. In this particular case, rather than using (3.14) it is better to choose d as

$$d_t = \begin{cases} 1 & \text{if } t \in \{(S,\ell) \mid \ell \in D^S,\ \ell_i = k\} \\ 1 & \text{if } t = (\{i\},k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall t \in T. \tag{3.17}$$

Vector (3.17) satisfies $d \in M^?$, in contrast to vector (3.14) which satisfies only $d \in M$. Thus, the update $f^{k+1} = f^k + d$ is a mere reparametrization[58], which is desirable as explained in §3.2.1.1. ◆

Example 3.2. We now consider cycle consistency as defined in [92].[59] As this local consistency was defined only for pairwise CSPs, we assume that $|S| = 2$ for each $S \in C$. Let L be a (polynomially sized) set of cycles in the graph (V, C_2). A CSP A is cycle consistent if for each tuple $(\{i\},k) \in A$ (where $i \in V$ and $k \in D$) and each cycle $L \in L$ 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 [92] constructs a deactivating direction whenever an inconsistent cycle is found. Moreover, the constructed direction in this case coincides with (3.14) for a suitable set P (i.e., $A' = T_A \wedge P$) which contains a subset of the forbidden tuples within the cycle. ◆

Example 3.3. Recall that a CSP A is singleton arc consistent if for every tuple $t = (\{i\},k) \in A$ (where $i \in V$ and $k \in D$), the CSP[60] $A|_{x_i = k} = A \wedge (T_{\{i\}} \setminus \{t\})$ has non-empty AC closure. Good (i.e., sparse) deactivating directions for singleton arc consistency (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 trace back the AC operations and identify only those tuples that were necessary to prove the empty AC closure. These tuples form the set P. The deactivating direction is then constructed as in Theorem 3.3 with $R = \{(\{i\},k)\}$ and $A' = T_A \wedge P$. Note that SAC does not have bounded support [19] as many other local consistencies do, so the size of P can significantly vary for different CSP instances. ◆

[58] Such reparametrizations correspond to soft arc consistency operations extend and project that we mentioned in Remark 1.12.

[59] 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.

[60] This can be also stated as $A|_{x_i = k} = A \wedge (\{(\{i\},k')\} \mid k' \in D \setminus \{k\})$. In other words, the solutions of CSP $A|_{x_i = k}$ are the solutions x of CSP A satisfying $x_i = k$. This notation is used, e.g., in [18, 99].

```
1 procedure (S, (R_i)_{i=0}^n, (d^i)_{i=0}^n) = propagate(A)
2 Initialize n := 0, A_0 := A.
3 while A_n is not -consistent do
4 |   Find a set R_n ⊆ A_n and an R_n-deactivating direction d^n for A_n.
5 |   A_{n+1} := A_n ⊖ R_n
6 |   if ∃S ∈ C : A_{n+1} ∩ T_S = ∅ then
7 |   |   return (S, (R_i)_{i=0}^n, (d^i)_{i=0}^n)
8 |   n := n + 1
9 return (∅, (R_i)_{i=0}^{n-1}, (d^i)_{i=0}^{n-1}))
```

Algorithm 3.3: The procedure propagate applies constraint propagation to CSP $A \in T$ 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 $S = \emptyset$.

3.2.2.2 Composing Deactivating Directions

Consider now a propagator which, for a current CSP $A \in T$, returns a set $R \subseteq A$ such that SOL(A) = SOL($A \ominus R$) 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 or it becomes explicit that the CSP is unsatisfiable (due to $A \cap T_S = \emptyset$ for some $S \in C$). This is outlined in Algorithm 3.3, 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 $A_i = A \ominus \bigcup_{j=0}^{i-1} R_j$ for every $i \in \{0, \ldots, n+1\}$. Therefore, by Theorem 3.4, we have SOL(A) = SOL(A_1) = SOL(A_2) = $\cdots$ = SOL(A_{n+1}), which implies that if A_{n+1} is unsatisfiable, then so is A. Note that Algorithm 3.3 is an extension of the previously shown Algorithm 1.2 that did not use deactivating directions.

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

Proposition 3.1. Let $A \in T$ 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 $A \ominus R$. Let

$$\gamma = \begin{cases} 0 & \text{if } d'_t \geq -1 \text{ for all } t \in R; \\ \max\{(-1 - d'_t)/d_t \mid t \in R,\ d'_t > -1\} & \text{otherwise.} \end{cases} \tag{3.18}$$

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

Proof. First, if $d'_t \geq -1$ for all $t \in R$, then $d'' = d'$ satisfies the required condition immediately. Otherwise, $\gamma > 0$ 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 γ. We will show that d'' satisfies the conditions in Definition 3.1.

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

```
1 procedure (R, d) = 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 ∈ I or ∃t ∈ R_i : d_t ≠ 0 then
6 |   |   d := d + d^i (where ⊕ is given by (3.18) with d, d^0, R replaced by d^i, d, R_i)
7 |   |   R := R ∪ R_i
8 return (R, d)
```

Algorithm 3.4: The procedure compose takes the sequences $(R_i)_{i=0}^n$ and $(d^i)_{i=0}^n$ (generated by the procedure propagate in Algorithm 3.3) and a non-empty index set $I \subseteq \{0, \dots, n\}$ and composes them into an R-deactivating direction d for A.

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

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

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

Proposition 3.1 allows us to combine R_i-deactivating direction d^i for $A_i = A_{i-1} \setminus R_{i-1}$ 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 $\bigcup_{i=0}^n R_i$-deactivating direction for A, which certifies unsatisfiability of A whenever Algorithm 3.3 detects on line 6 that A_{n+1} (and thus also A) is unsatisfiable.

However, it is not always necessary to construct a full $\bigcup_{i=0}^n R_i$-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 $A_{n+1} = A \setminus \bigcup_{i=0}^n R_i$, as mentioned above) returned by Algorithm 3.3 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 $(A \setminus R) \cap T_S = \emptyset$. Such a direction d still certifies unsatisfiability of A and can be sparser and/or may have lower objective values $F(x \mid d)$ than a $\bigcup_{i=0}^n R_i$-deactivating direction, which is desirable as explained in §3.2.1.1.

This is outlined in Algorithm 3.4, 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 $R \subseteq \bigcup_{i \in I} R_i$. Although Algorithm 3.4 is applicable to 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

$$I = \{ i \in \{0, \dots, n\} \mid R_i \cap T_S \neq \emptyset \} \tag{3.19}$$

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

Proposition 3.2. Let $S \in C$ be such that $(A \setminus \bigcup_{i=0}^n R_i) \cap T_S = \emptyset$. Let I be given by (3.19). Then, $(A \setminus \bigcup_{i \in I} R_i) \cap T_S = \emptyset$.

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

Correctness of Algorithm 3.4 is given by the following theorem:

Theorem 3.5. Algorithm 3.4 returns an R-deactivating direction d for A such that $\bigcup_{i \in I} R_i \subseteq R \subseteq \bigcup_{i=0}^{n} R_i$.

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 an R-deactivating direction for A, which will be done by induction. We claim that vector d is always an R-deactivating direction for A_i on line 3 and an 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 3.1.

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, Proposition 3.1 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 3.3. □

Remark 3.3. This is similar to what the VAC [33] or Augmenting DAG algorithm [95, 146] 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 empty AC closure of $A(f)$ is detected (which corresponds to $T_S \setminus A_{n+1} = \emptyset$ 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).

3.2.3 Line Search

In §3.2.2, we showed how to construct an R-deactivating direction d for CSP A, which certifies unsatisfiability of A whenever $(A - R) \setminus T_S = \emptyset$ for some $S \in C$. Given a WCSP $f \in R^T$ with $A(f) = A$, to obtain $f' \in f + M$ with $B(f') < B(f)$ (as in Theorem 3.2), we need to find a step size > 0 so that $f' = f + d$, as discussed in §3.2.1.2. That means, we need to find > 0 such that $B(f + d) < B(f)$.

Finding the best step size (i.e., exact line search) would require finding a global minimum of the univariate convex piecewise-affine function $\mapsto B(f + d)$. As this would be too expensive for large WCSP instances, we perform only approximate line search, i.e., find some non-zero step size by the following theorem.[61]

[61] In detail, the step size $\min\{\ ,\ \}$ computed in Theorem 3.6c corresponds to the first breakpoint of the univariate function with a lower objective. Similarly as in §2.4.2, this is analogous to the first-hit strategy in [48a, §3.1.4].

Theorem 3.6. Let $f \in R^T$. Let d be an R-deactivating direction for $A(f)$. Denote[62]

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

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

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

(a) If $(A(f) - R) \cap T_S \neq \emptyset$ and $0 \leq \alpha \leq \gamma$, 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 $0 < \alpha < \gamma$, then $A(f') \cap T_S = (A(f) - R) \cap T_S$.

(c) If $(A(f) - R) \cap T_S = \emptyset$ and $0 < \alpha \leq \min\{\gamma, \gamma\}$, then $\max_{t \in T_S} f'_t < \max_{t \in T_S} f_t$.

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

To prove (a), let $t \in (A(f) - R) \cap T_S$. Hence, by Definition 3.1, $d_t = 0$ 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 $\alpha \leq \frac{\max_{t \in T_S} f_t - f_{t'}}{d_{t'}}$, so $f'_{t'} = f_{t'} + \alpha d_{t'} \leq \max_{t \in T_S} f_t$.

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 $f'_t = f_t + \alpha d_t < f_t = \max_{t' \in T_S} f_{t'}$ 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 $\alpha < \frac{\max_{t' \in T_S} f_{t'} - f_t}{d_t}$, $f'_t = f_t + \alpha d_t < \max_{t' \in T_S} f_{t'}$, 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 $d_{t'} > d_t$, $\alpha \leq \frac{f_t - f_{t'}}{d_{t'} - d_t}$ implies $f'_t = f_t + \alpha d_t \geq f_{t'} + \alpha d_{t'} = f'_{t'}$. If $d_{t'} \leq d_t$, then also $d_{t'} \leq d_t$ and $f'_{t'} = f_{t'} + \alpha d_{t'} \leq f_t + \alpha d_t = f'_t$ holds for any $\alpha \geq 0$ since $f_{t'} \leq f_t$. 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 $(A(f) - R) \cap T_S \neq \emptyset$ for all $S \in C$, then there is $\gamma > 0$ such that $f' = f + \alpha d$ satisfies $B(f') = B(f)$ and $A(f') = A(f) - R$ by Theorem 3.6a and 3.6b. This justifies why such a direction d is called R-deactivating: a suitable update of f along this direction makes tuples R inactive.

Remark 3.4. This might suggest that to improve the current bound $B(f)$, we need not use Algorithm 3.4 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 $0 < \alpha_i < \gamma_i$. We empirically found Algorithm 3.5 to be preferable.

[62] γ is always defined: by Definition 3.1 we have $F(x \mid d) \neq 0$ for all x, hence $\exists t : d_t < 0 \Rightarrow \exists t' : d_{t'} > 0$. 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 $\gamma = +\infty$ by convention and $\min\{\gamma, \gamma\} = \gamma$.

```
   input: WCSP g ∈ R^T.
1  Initialize f := g.
2  repeat
3  |  (S, (R_i)_{i=0}^n, (d^i)_{i=0}^n) := propagate(A(f))                (see Algorithm 3.3)
4  |  if S ≠ ∅ then
5  |  |  Define I as in (3.19).
6  |  |  (R, d) := compose((R_i)_{i=0}^n, (d^i)_{i=0}^n, I)               (see Algorithm 3.4)
7  |  |  Update f := f + min{f, ∞}d following Theorem 3.6.
8  |  else
9  |  |  return B(f)
```

Algorithm 3.5: The final algorithm to iteratively improve feasible solutions to (3.7).

If d is an R-deactivating direction for $A(f)$ and we have $(A(f) \cdot R) \setminus T_S = \emptyset$ for some $S \in C$, then there is $\alpha > 0$ such that $f' = f + \alpha d$ satisfies $B(f') < B(f)$ by Theorem 3.6a and 3.6c. The following corollary of Theorem 3.6 finally justifies why the certificate of unsatisfiability of CSP $A(f)$ is an improving direction for (3.7):

Corollary 3.1. CSP $A \cdot T$ is unsatisfiable if and only if there is $d \in M$ such that for every $f \in R^T$ 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 $A \setminus T_S = \emptyset$, A is unsatisfiable and no $f \in R^T$ satisfies $A = A(f)$, so the second condition is trivially satisfied by choosing any $d \in M$.

If $A \setminus T_S \neq \emptyset$ for all $S \in C$ but A is unsatisfiable, let d be any A-deactivating direction (which exists by Theorem 3.3). It follows from Theorem 3.6 that for any $f \in R^T$ with $A(f) = A$, we can compute a suitable step size $\alpha > 0$ such that $B(f + \alpha d) < B(f)$. The case when A is satisfiable follows from Theorem 3.2. □

3.2.4 Final Algorithm

Having certificates of unsatisfiability from §3.2.2 and step sizes from §3.2.3, we can now precisely formulate in Algorithm 3.5 the iterative method that was previously sketched in §3.2.1.2 (Algorithm 3.2). First, constraint propagation is applied to CSP $A(f)$ by Algorithm 3.3 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 $A_{n+1} \setminus T_S = \emptyset$ for some $S \in C$, define I as in (3.19) so that $(A(f) \cdot \bigcup_{i \in I} R_i) \setminus T_S = \emptyset$ and compute an R-deactivating direction d where $R = \bigcup_{i \in I} R_i$ using Theorem 3.5. Since $(A(f) \cdot R) \setminus T_S = \emptyset$, we can update WCSP f using Theorem 3.6. Consequently, the bound $B(f)$ strictly improves after each update on line 7.

Although our theoretical results are more general, our implementation is limited only to pairwise WCSPs. In our implementation, we again use the trick similar to capacity scaling, i.e., we replace the active tuples $A(f)$ with 'almost' active tuples $A_\theta(f)$ defined in (2.10) and proceed as in Algorithm 2.2. 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 ε becomes very small.[63] This heuristic 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.

We implemented two versions of Algorithm 3.5 (including 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) uses singleton arc consistency (SAC). Precisely, we alternate between AC and SAC propagators: whenever a single tuple $(\{i\}, k)$ (where $i \in V$ and $k \in D_i$) 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 cycle consistency (CC). Though our implementation is different from [92] (we compose deactivating directions rather than alternate between the cycle-repair procedure and the Augmenting DAG algorithm), it has the same stopping points.

The procedures for generating deactivating directions for AC, SAC and CC were implemented as described in Examples 3.1, 3.3, and 3.2, respectively. In SAC and CC, it is useful to step back to AC whenever possible because deactivating directions of AC correspond to reparametrizations (that are more favorable, recall §3.2.1.1) rather than super-reparametrizations.

Remark 3.5. In analogy to [33, 107], let us call a WCSP instance f virtual φ-consistent (e.g., virtual AC or virtual RPC) if $A(f)$ has 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.

Since we restricted ourselves to pairwise WCSPs, $(V; C_2)$ is an undirected graph. The cycles in VCC-SR were chosen as follows: if $|C_2| = |V| \cdot 5$ (i.e., the average degree of the nodes in $(V; C_2)$ is at most 5), then all cycles of length 3 and 4 present in the graph $(V; C_2)$ are used. If $2|C_2| = |V| \cdot 10$, then all cycles of length 3 present in the graph are used. If $2|C_2| = |V| > 10$ or the above method did not result in any cycles, we use all fundamental cycles w.r.t. a spanning tree of the graph $(V; C_2)$ [116, §9]. No additional edges are added to the graph. Note, [92] 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 $A(f)$ have non-empty AC closure by reparametrizations), before running these methods we used toulbar2 [1] to reparametrize the input WCSP instance to a VAC state (because a specialized algorithm is faster than the more general Algorithm 3.5). We employed specialized data structures for storing the sequences $(R_i)_{i=0}^n$ and $(d^i)_{i=0}^n$ from Algorithm 3.3, which utilize the property that the sets $(R_i)_{i=0}^n$ are disjoint and make easier sequential querying

[63] In detail, we initialized $\varepsilon = \max_{k_i;k_j} g_{\{i,j\}}(k_i; k_j) - \min_{k_i;k_j} g_{\{i,j\}}(k_i; k_j) + \max_k g_{i'}(k) - \min_k g_{i'}(k)$ where $\{i, j\} \in C_2$ and $i' \in V$ is the edge and variable with the lowest index (based on indexing in the input instance). The terminating condition was $\varepsilon < 10^{-6}$. In order to improve the efficiency of our method, we also decreased ε whenever the bound did not improve by more than 10^{-15} in 20 consecutive iterations (cf. VAC in [33, §11.1]).

of (sparse) vectors $(\delta^i)_{i=0}^n$ in Algorithm 3.4. 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 3.4. To avoid working with `structured' tuples (1.23), 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 ε or constants mentioned in Footnote 63) were not learned nor systematically optimized. SAC was checked on all active tuples without warm-starting or using any faster SAC algorithm than SAC1 [18, 45, Figure 2]. Perhaps most importantly, we did not implement inter-iteration warm-starting as in [145, 48a], i.e., after updating the weights on line 7 of Algorithm 3.5, some deactivating directions in the sequence that were not used in computing the improving direction may be preserved for the next iteration instead of being computed from scratch. Except for computing the deactivating directions, the code was the same for VSAC-SR and VCC-SR. We implemented everything in Java.

3.2.5 Experimental Results

We compared the bounds calculated by VSAC-SR and VCC-SR with the bounds provided by EDAC [43], VAC [33], pseudo-triangles (option -t=8000 in toulbar2, adds up to 8 GB of ternary weight functions), PIC, EDPIC, maxRPC, and EDmaxRPC [107], which are implemented in toulbar2 [1].

We did the comparison on the Cost Function Library benchmark [2]. 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). 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. Then, for each instance and method, we computed the normalized bound $\frac{B_w - B_m}{B_w - B_b}$ where B_w and B_b are the worst and the best bound for the instance among all the methods, respectively, and B_m is the bound computed by the method for the instance. Thus, the best bound[64] 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 5731 instances in total.

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

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 3.1: 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	0.11	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

Table 3.2: Results on instances from Cost Function Library: Average CPU time in seconds.

The results in Table 3.1 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 [3], planning [40] and qplib [64] 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 3.2. 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.

3.3 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. Although we did not need these properties for the previously described method, they may be valuable for future research. Here, we first revisit in §3.3.1 the notion of a minimal CSP for a set of assignments that was studied, e.g., in [105, §3, 46, §2.3.2]. The key result of §3.3 is presented in §3.3.2, 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 (3.7), showing that they need not coincide in general. In §3.3.3, we analyze the case of general (i.e., not necessarily optimal for (3.7)) super-reparametrizations.

3.3.1 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 $A \in \mathbb{T}$ such that $X = \mathrm{SOL}(A)$, i.e., when is X representable as the solution set of a CSP with a given structure (V, D, C). For that, denote

$$A_{\min}(X) = \bigcap A(X) \quad \text{where} \quad \mathbb{A}(X) = \{A \in \mathbb{T} \mid X \subseteq \mathrm{SOL}(A)\}. \tag{3.20}$$

Thus, $\mathbb{A}(X)$ 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 pairwise CSPs, this concept was studied in [105, §3] and [46, §2.3.2].

Proposition 3.3. The mapping SOL preserves intersections[65], i.e., for any $A_1, A_2 \in \mathbb{T}$ we have $\mathrm{SOL}(A_1 \wedge A_2) = \mathrm{SOL}(A_1) \cap \mathrm{SOL}(A_2)$.

[65] We remark that isotony of SOL (that we noted already in §1.4) does not in general imply preserved intersections. A weaker result than our Proposition 3.3 is [105, Theorem 3.2]: in our notation, it says that $\mathrm{SOL}(A_1) = \mathrm{SOL}(A_2)$ implies $\mathrm{SOL}(A_1 \wedge A_2) = \mathrm{SOL}(A_1)$.

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

$$\begin{aligned} x \in SOL(A_1) \cap SOL(A_2) &\iff x \in SOL(A_1),\ x \in SOL(A_2) \\ &\iff \forall\ S \in C:\ S, x_{|S} \in A_1,\ S, x_{|S} \in A_2 \\ &\iff \forall\ S \in C:\ S, x_{|S} \in A_1 \cap A_2 \\ &\iff x \in SOL(A_1 \cap A_2). \end{aligned}$$

□

Proposition 3.4. For any $X \subseteq D^V$, the set $\mathbb{A}(X)$ is closed under intersections, i.e., for any $A_1, A_2 \subseteq T$ we have $A_1, A_2 \in \mathbb{A}(X) \Rightarrow A_1 \cap A_2 \in \mathbb{A}(X)$.

Proof. Suppose that $X \subseteq SOL(A_1)$ and $X \subseteq SOL(A_2)$, then $X \subseteq SOL(A_1) \cap SOL(A_2) = SOL(A_1 \cap A_2)$, where the equality holds by Proposition 3.3. □

Proposition 3.4 implies $A_{min}(X) \in \mathbb{A}(X)$, i.e., $X \subseteq SOL(A_{min}(X))$. This shows that $A_{min}(X)$ is the smallest CSP whose solution set includes X. It follows that $X = SOL(A_{min}(X))$ if and only if $X = SOL(A)$ for some $A \subseteq T$.

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

Proposition 3.5 ([105, 46]). We have $A_{min}(X) = \{(S,k) \in T \mid \exists x \in X : x_{|S} = k\}$.

Proof. Denote $A^0 = \{(S,k) \in T \mid \exists x \in X : x_{|S} = k\}$. By definition of A^0, we have $SOL(A^0) \supseteq X$, so $A^0 \in \mathbb{A}(X)$ and $A_{min}(X) = \bigcap \mathbb{A}(X) \subseteq A^0$.

It remains to show that $A_{min}(X) \supseteq A^0$. For contradiction, suppose there is a tuple $(S,k) \in A^0 \setminus A_{min}(X)$. By definition of A^0, there exists $x \in X$ such that $x_{|S} = k$. However, since $(S, x_{|S}) = (S,k) \notin A_{min}(X)$, we have $x \notin SOL(A_{min}(X))$. By $x \in X$, this contradicts $X \subseteq SOL(A_{min}(X))$. □

Note that Proposition 3.5 shows that $A_{min}(X)$ is positively consistent (recall Definition 1.6) for every $X \subseteq D^V$.

Theorem 3.7. For any $X \subseteq D^V$ and $A \subseteq T$, we have

$$A_{min}(X) \subseteq A \iff X \subseteq SOL(A). \tag{3.21}$$

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

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

Comparing (3.21) with (3.20) shows that the set $\mathbb{A}(X)$ is just an interval (w.r.t. the partial ordering given by the set inclusion):

$$\mathbb{A}(X) = \{A \mid A_{min}(X) \subseteq A \subseteq T\} \tag{3.22}$$

Theorem 3.7 further reveals that the maps A_{min} and SOL form a Galois connection [42, §7] between sets $2^{(D^V)}$ and 2^T, partially ordered by the set inclusion[66]. Associated with the Galois connection are the closure operator $SOL \circ A_{min}$ and the dual closure operator $A_{min} \circ SOL$. We have already seen their meaning:

[66] To our knowledge, we were the first to notice this in [56a, §5.1].

- For any CSP $A \subseteq T$, the CSP $A_{\min}(\mathrm{SOL}(A))$ is the positive consistency (dual) closure $C_{\mathrm{pos}}(A)$ from §1.4.1.
- 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 3.6. Following [42, §7.27], it is easy to see in this case that the mappings $A_{\min}$ and SOL are mutually inverse bijections (even order-isomorphisms) if we restrict ourselves only to positively consistent CSPs, i.e., $\{A \subseteq T \mid A_{\min}(\mathrm{SOL}(A)) = A\}$ and sets of assignments representable as solution sets of some CSP $\subseteq T$, i.e., $\{\mathrm{SOL}(A) \mid A \subseteq T\}$.

3.3.2 Optimal Assignments of Optimal Super-Reparametrizations

Theorem 3.1 (or already Theorem 1.16) says that the optimal value of (3.7) equals the optimal value of WCSP g. We now focus on the optimal assignments (rather than value) of WCSP g. For brevity, we denote the set of all optimal assignments of WCSP g by

$$\mathrm{OPT}(g) = \operatorname*{argmax}_{x \in D^V} F(x \mid g) \subseteq D^V. \tag{3.23}$$

Theorem 3.8. If f is optimal for (3.7), then $\mathrm{OPT}(g) \subseteq \mathrm{OPT}(f) = \mathrm{SOL}(A(f))$.

Proof. To show $\mathrm{OPT}(g) \subseteq \mathrm{OPT}(f)$, let $x \in \mathrm{OPT}(g)$. By statement (b) in Theorem 3.1, $F(x \mid g) = B(f)$. Since $B(f) \le F(x \mid f) \le F(x \mid g)$, we have that $B(f) = F(x \mid f) = F(x \mid g)$, thus x is optimal for WCSP f.

The equality $\mathrm{OPT}(f) = \mathrm{SOL}(A(f))$ follows from Theorem 1.15b together with $B(f) = \max_{x \in D^V} F(x \mid f)$ which is given by optimality of f (see statement (b) in Theorem 3.1). □

Our main goal in §3.3 is to characterize when the inclusion in Theorem 3.8 holds with equality, which is given by a later stated Theorem 3.10.

Proposition 3.6. For every $g \in \mathbb{R}^T$ and $A \subseteq T$ such that $\mathrm{OPT}(g) \subseteq \mathrm{SOL}(A)$, there exists $f \in \mathbb{R}^T$ optimal for (3.7) such that $A = A(f)$.

Proof. Define the vector f by

$$f_t = \begin{cases} F_1 - |C| & \text{if } t \in A \\ F_2 - |C| & \text{if } t \notin A \end{cases} \qquad \forall t \in T \tag{3.24}$$

where

$$F_1 = \max_{x \in D^V} F(x \mid g) \quad \text{and} \quad F_2 = \max\{ F(x \mid g) \mid x \in D^V,\ F(x \mid g) < F_1 \} \tag{3.25}$$

are the best and the second-best objective value of WCSP g, respectively. Note, if $\mathrm{OPT}(g) = D^V$, then F_2 is undefined but it does not matter because it is never used in (3.24).

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

$$\max_{t \in T_S} f_t = F_1 - |C|. \tag{3.26}$$

Equality $A = A(f)$ now follows from (3.24).

To show that f is feasible for (3.7), we distinguish two cases:

(a) WCSP g (b) WCSP f^1 (c) WCSP f^2 (d) WCSP f^3

Figure 3.5: Examples of four WCSPs with the same structure. WCSPs f^1, f^2, and f^3 are optimal super-reparametrizations of WCSP g.

- If $x \in \mathrm{OPT}(g)$, i.e., $F(x|g) = F_1$, then $x \in \mathrm{SOL}(\mathbf{A}) = \mathrm{SOL}(\mathbf{A}(f))$. Therefore for all $S \in \mathbf{C}$ we have $S; x_S \in \mathbf{A}(f)$, hence $f_S(x_S) = F_1/|\mathbf{C}|$ by (3.26). Substituting into (1.33) yields $F(x|f) = F_1$ and $F(x|f) = F_1 = F(x|g)$.
- If $x \notin \mathrm{OPT}(g)$, we have $f_S(x_S) \ge F_2/|\mathbf{C}|$ for all $S \in \mathbf{C}$, hence $F(x|f) \ge F_2$ by (1.33). By (3.25) we also have $F(x|g) \ge F_2$, so $F(x|f) \ge F_2 \ge F(x|g)$.

To show that f is optimal for (3.7), we use (3.26) to obtain $B(f) = \sum_{S \in \mathbf{C}} F_1/|\mathbf{C}| = F_1 = \max_x F(x|g)$ and apply Theorem 3.1. □

Example 3.4. To exemplify Proposition 3.6, consider WCSP g from Figure 3.5a. We have that $|\mathbf{C}| = 3$, $F_1 = 3$ and $F_2 = 0$. Each of the WCSPs shown in Figures 3.5b, 3.5c, and 3.5d has a different set of active tuples, is a super-reparametrization of g, and is optimal for (3.7). 4

Theorem 3.9. For every $g \in \mathbf{R}^{\mathbf{T}}$, we have

$$\mathbf{A}(\mathrm{OPT}(g)) = \{\mathbf{A}(f) \mid f \text{ is optimal for (3.7) } g\}. \tag{3.27}$$

Proof. The inclusion $\subseteq$ says that for every optimal f we have $\mathrm{OPT}(g) \subseteq \mathrm{SOL}(\mathbf{A}(f))$, which was proved in Theorem 3.8. The inclusion $\supseteq$ was proved in Proposition 3.6. □

Now we combine the results of §3.3.1 and §3.3.2 to obtain the main result of §3.3. First observe that, by (3.22), the set (3.27) is just the interval $\{\mathbf{A} \mid \mathbf{A}_{\min}(\mathrm{OPT}(g)) \subseteq \mathbf{A} \subseteq \mathbf{T}\}$.

Theorem 3.10. For every $g \in \mathbf{R}^{\mathbf{T}}$, the following statements are equivalent:

(a) $\mathrm{OPT}(g) = \mathrm{SOL}(\mathbf{A})$ for some $\mathbf{A} \subseteq \mathbf{T}$,

(b) $\mathrm{OPT}(g) = \mathrm{OPT}(f)$ for some f optimal for (3.7).

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

Proof. Let $g \in \mathbf{R}^{\mathbf{T}}$. By Theorem 3.9, there exists f optimal for (3.7) satisfying $\mathbf{A}(f) = \mathbf{A}_{\min}(\mathrm{OPT}(g))$. By Theorem 3.8, this f satisfies $\mathrm{OPT}(f) = \mathrm{SOL}(\mathbf{A}(f))$.

By the results of §3.3.1, statement (a) is equivalent to $\mathrm{OPT}(g) = \mathrm{SOL}(\mathbf{A}_{\min}(\mathrm{OPT}(g)))$. 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 3.10 shows that the set inclusion in Theorem 3.8 holds with equality for some optimal f if and only if the set $\mathrm{OPT}(g)$ is representable as a solution set of some CSP with the same structure. If no such CSP exists, then $\mathrm{OPT}(g) \subset \mathrm{OPT}(f)$ for all optimal f. An example of WCSP g for which no such CSP exists is in Figure 3.6.

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.