1
Tractable Valued Constraints

Peter G. Jeavons[a] and Stanislav Živný[b]

Abstract

In this chapter, we will survey recent results on the broad family of optimisation problems that can be cast as valued constraint satisfaction problems (VCSPs). We discuss general methods for analysing the complexity of such problems, and give examples of tractable cases.

1.1 Introduction

Computational problems from many different areas involve finding values for variables that satisfy certain specified restrictions and optimise certain specified criteria.

In this chapter, we will show that it is useful to abstract the general form of such problems to obtain a single generic framework. Bringing all such problems into a common framework draws attention to common aspects that they all share, and allows very general analytical approaches to be developed. We will survey some of these approaches, and the results that have been obtained by using them.

The generic framework we shall use is the *valued constraint satisfaction problem* (VCSP), defined formally in Section 1.3. We will show that many combinatorial optimisation problems can be conveniently expressed in this framework, and we will focus on finding restrictions to the general problem which are sufficient to ensure tractability.

[a] Supported by EPSRC grant EP/G055114/1.
E-mail: Peter.Jeavons@cs.ox.ac.uk
[b] Supported by a Junior Research Fellowship at University College, Oxford.
E-mail: Standa.Zivny@cs.ox.ac.uk

An important and well-studied special case of the VCSP is the constraint satisfaction problem (CSP), which deals with combinatorial search problems which have no optimisation criteria. We give a brief introduction to the CSP in Section 1.2, before defining the more general VCSP framework in Section 1.3. Section 1.4 then presents a number of examples of problems that can be seen as special cases of the VCSP.

The remainder of the chapter discusses what happens to the complexity of the valued constraint satisfaction problem when we restrict it in various ways. Section 1.6 considers the important special case of valued constraint problems involving submodular functions. Motivated by this example, we introduce the notion of a *multimorphism*, which can be used to define many other tractable forms of valued constraint, and we use this notion to obtain a complexity classification of all valued constraints over a 2-element domain. In Section 1.7 we present a complexity classification of problems over arbitrary finite domains allowing all unary valued constraints.

Section 1.8 describes the basics of a recently developed general algebraic theory for studying the complexity of different forms of valued constraints. Finally, Section 1.9 concludes with some open problems.

This chapter is self-contained, but we refer the reader to one of the standard textbooks (Apt, 2003; Dechter, 2003; Rossi et al., 2006) for more detailed background information on the basics of constraint satisfaction. Earlier surveys on the complexity of various forms of restricted constraint satisfaction problems can be found in (Cohen and Jeavons, 2006; Chen, 2006; Kolaitis and Vardi, 2007; Creignou et al., 2008a; Hell and Nešetřil, 2008; Jeavons, 2009; Nešetřil et al., 2010).

1.2 Constraint Satisfaction Problems

In this section, we present the simplest form of constraint satisfaction problem, where there are no optimisation criteria, so all solutions satisfying the specified constraints are considered equally desirable. This form of problem has been widely-studied since the pioneering work of Montanari (1974), and there are now several textbooks covering this topic (Apt, 2003; Dechter, 2003; Rossi et al., 2006), as well as a regular international conference devoted to constraint programming.

The basic constraint satisfaction problem (CSP) can be defined in a number of equivalent ways. Here we will present three equivalent standard definitions, each emphasizing different aspects of the problem.

Our first definition is couched in the terminology of predicate logic. Let $\mathbf{B} = (D, R_1, R_2, \ldots)$ be a relational structure, where D is the universe and $R_1, R_2, \ldots$ are relations over D. A first-order formula is called *primitive positive* over $\mathbf{B}$ if it is of the form

$$\exists x_1 \exists x_2 \ldots \exists x_n \ \psi_1 \wedge \cdots \wedge \psi_m$$

where the ψ_i are atomic formulas, i.e., formulas of the form $R(x_{i_1}, \ldots, x_{i_k})$ where R is a relation symbol for a k-ary relation from $\mathbf{B}$.

Definition 1.1 An instance of the constraint satisfaction problem (CSP) is given by a primitive positive sentence, Φ, over a fixed relational structure, $\mathbf{B}$. The question is whether Φ is true in $\mathbf{B}$.

This logical formulation of constraint satisfaction allows some classical combinatorial problems to be formulated very naturally.

Example 1.2 (Satisfiability) The standard propositional SATISFIABILITY problem for ternary clauses, 3-SAT (Garey and Johnson, 1979) consists in determining whether it is possible to satisfy a Boolean formula given in CNF as a conjunction of ternary clauses.

This can be viewed as a constraint satisfaction problem by fixing the structure $\mathbf{B}_{3SAT}$ to be $(\{0,1\}, R_1, \ldots, R_8)$, where the R_i are the 8 relations definable by a single ternary clause. For example, the clause $x \vee \neg y \vee \neg z$ can be written as $R_1(x, y, z)$, where

$$R_1 = \{\langle 0,0,0\rangle, \langle 0,0,1\rangle, \langle 0,1,0\rangle, \langle 1,0,0\rangle, \langle 1,0,1\rangle, \langle 1,1,0\rangle, \langle 1,1,1\rangle\}.$$

An instance of 3-SAT corresponds to a primitive positive sentence over $\mathbf{B}_{3SAT}$ with a conjunct for each clause.

Example 1.3 (Graph Colouring) The standard k-COLOURABILITY problem (Garey and Johnson, 1979) consists in determining whether it is possible to assign k colours to the vertices of a given graph so that adjacent vertices are assigned different colours.

This can be viewed as a constraint satisfaction problem by fixing the structure $\mathbf{B}_{kCOL}$ to be $(\{1, \ldots, k\}, \neq)$, where $\neq$ is the binary disequality relation on $\{1, \ldots, k\}$ given by $\{\langle i, j\rangle \mid i, j \in \{1, \ldots, k\}, i \neq j\}$.

An instance of GRAPH k-COLOURING corresponds to a primitive positive sentence over $\mathbf{B}_{kCOL}$ with a conjunct for each edge of the graph.

An important line of research in dealing with constraint satisfaction problems has been the development of programming languages to facilitate the expression of practical problems, and software tools to solve such

problems. This approach is known as *constraint programming* (Rossi et al., 2006).

In the field of constraint programming, a constraint satisfaction problem is usually defined in a more operational way, as follows.

Definition 1.4 An instance of the constraint satisfaction problem (CSP) is a triple $\mathcal{P} = \langle V, D, \mathcal{C} \rangle$, where: V is a finite set of *variables*; D is a set of possible *values* that may be assigned to the variables (called the *domain*); $\mathcal{C}$ is a multi-set of *constraints*. Each element of $\mathcal{C}$ is a pair $c = \langle \sigma, R \rangle$ where σ is a tuple of variables called the *scope* of c, and R is a $|\sigma|$-ary *relation* over D that defines the combinations of assignments to the variables in σ that are allowed by c. An *assignment* for $\mathcal{P}$ is a mapping $s : V \to D$. A *solution* to $\mathcal{P}$ is an assignment which is consistent with all of the constraints.

This formulation focuses attention on the *variables*, the *domain* and the *constraints*; these are the key data structures in a software system for solving constraint satisfaction problems.[1] In this formulation the constraints are often represented by oracles - black-box algorithms for a particular constraint type, known as "propagators" (Rossi et al., 2006), which communicate information with each other during the search for a solution by modifying the domains of the individual variables.

Many real-world problems such as timetabling and scheduling, are captured very naturally in this formulation, as well as classic combinatorial search problems and puzzles (Rossi et al., 2006).

It is easy to translate from this second formulation of the constraint satisfaction problem (Definition 1.4) to our original logical formulation (Definition 1.1). To do this, simply collect together the relations over D that occur in the constraints of $\mathcal{C}$, to give a relational structure $\mathbf{B}$ with universe D. The instance can then be written as a primitive positive sentence over $\mathbf{B}$ with a conjunct $R(\sigma[1], \ldots, \sigma[m])$ for each constraint $\langle \sigma, R \rangle$ of arity m that occurs in $\mathcal{C}$.

In a given CSP instance there may be several constraints with the same constraint relation, but different scopes. If we collect together the scopes associated with a particular constraint relation we get a set of tuples which is itself a relation, but a relation over the set of variables, V. If we do this for each distinct constraint relation occurring in our problem, we obtain a collection of such relations over V, which can

[1] In Definition 1.4, we have assumed that all the variables have the same domain. If this is not the case, we could simply collect together all the possible values occurring in the domains of the individual variables into a single set D.

be viewed as a relational structure $\mathbf{A}$ with universe V. Note that each relation E in $\mathbf{A}$ corresponds to a relation R in $\mathbf{B}$ of the same arity, and vice versa. This is captured in standard algebraic terminology by saying that the two relational structures, $\mathbf{A}$ and $\mathbf{B}$ are *similar*. Note also that a solution to the original CSP instance is a mapping from V to D that maps any tuple of variables related by a relation E in $\mathbf{A}$ to a tuple of values which are related by the corresponding relation R in $\mathbf{B}$. This is captured in standard algebraic terminology by saying that a solution is a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$.

These observations gives rise to our third alternative formulation of the CSP, this time in the terminology of algebra.

Definition 1.5 An instance of the constraint satisfaction problem (CSP) is given by a pair of similar relational structures $\mathbf{A}$ and $\mathbf{B}$. The question is whether there exists a homomorphism from $\mathbf{A}$ to $\mathbf{B}$.

This clean algebraic formulation of constraint satisfaction was introduced by Feder and Vardi (1998) (and independently by Jeavons (1998)) and has turned out to be very useful for the analysis of the complexity of different forms of the problem.

Example 1.6 (Graph Homomorphism) The standard Graph Homomorphism problem (Hell and Nešetřil, 2004) consists in determining whether it is possible to map the vertices of a given graph G to the vertices of another given graph H so that adjacent vertices of G are mapped to adjacent vertices of H.

This can be viewed as a constraint satisfaction problem by viewing G and H as similar relational structures, each with a single binary relation. A homomorphism between these structures is precisely a mapping with the desired properties.

Example 1.7 (Graph Colouring) The standard k-Colourability problem described in Example 1.3 can be viewed as the constraint satisfaction problem which asks whether there is a homomorphism from the given graph G to the structure $\mathbf{B}_{kCOL}$, defined in Example 1.3, which corresponds to a complete graph on k vertices.

This formulation makes it easy to see that the Graph k-Colourability problem is a special case of Graph Homomorphism.

Example 1.8 (Clique) The standard k-Clique problem (Garey and Johnson, 1979) consists in determining whether a given graph G has a clique of size k, that is, a set of k vertices which are fully connected. This

can be viewed as a constraint satisfaction problem which asks whether there is a homomorphism from the complete graph on k vertices to the given graph G.

This formulation of the problem makes it easy to see that k-Clique is a special case of Graph Homomorphism.

It is clear from the examples above that the general CSP is at least NP-hard. This has prompted many researchers to investigate ways in which restricting the problem can reduce its complexity. We will call a restricted version of the CSP *tractable* if there is a polynomial-time algorithm to determine whether any instance of the restricted problem has a solution.

The algebraic formulation of the CSP, given in Definition 1.5, clearly identifies two separate aspects of the specification of an instance: the *source* structure, $\mathbf{A}$, and the *target* structure, $\mathbf{B}$.

The source structure, $\mathbf{A}$, specifies the tuples referred to in Definition 1.4 as the *scopes* of the constraints. If we restrict the possible source structures that we allow in an instance, then we are restricting the set of variables and the ways in which the constraints may be imposed on those variables. Such restrictions are known as *structural restrictions*.

The target structure, $\mathbf{B}$, specifies the relations referred to in Definition 1.4 as *constraint relations*. If we restrict the possible target structures that we allow in an instance, then we are restricting the set of possible values and the types of constraints that may be imposed on those values. Such restrictions are known as *language restrictions*.

Definition 1.9 Given classes of structures, $\mathcal{A}$ and $\mathcal{B}$, we define the problem $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$ to be the class of CSP instances $(\mathbf{A}, \mathbf{B})$, where $\mathbf{A} \in \mathcal{A}$ and $\mathbf{B} \in \mathcal{B}$.

Structural restrictions If $\mathcal{B}$ is the class of all structures, we write $\mathrm{CSP}(\mathcal{A}, -)$ in place of $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$. In this case we impose no restriction on the type of constraint, but some restriction on how the constraints may overlap.

Structural restrictions of this kind have been studied since the pioneering work by Montanari (1974), who observed that CSPs on trees are solvable in polynomial time. This observation has since been generalised in many different ways (Freuder, 1985; Dechter and Pearl, 1989; Freuder, 1990; Gyssens et al., 1994; Gottlob et al., 2000; Kolaitis and Vardi, 2000; Dalmau et al., 2002; Gottlob et al., 2002; Chen and Dal-

mau, 2005; Grohe and Marx, 2006; Adler et al., 2007; Cohen et al., 2008a; Gottlob et al., 2009; Marx, 2010b,a, 2011).

In general, the structural restrictions that ensure tractability are those that enforce a bound on some measure of width in the class of source structures allowed (Gottlob et al., 2000). Complete classifications, identifying all tractable cases, have been obtained for bounded-arity CSPs by Grohe (2007), and for unbounded-arity CSPs by Marx (2010c).

One example of a constraint satisfaction problem with restricted structure is the k-Clique problem (see Example 1.8), which is tractable for any bounded k, but NP-complete if k is unbounded.

Language restrictions If $\mathcal{A}$ is the class of all structures, we write $\mathrm{CSP}(-, \mathcal{B})$ in place of $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$. In this case we impose no restriction on the way the constraints are placed, but some restriction on the forms of constraints that may be imposed.

Such language restrictions have also been widely studied (Jeavons and Cooper, 1995; Jeavons et al., 1997; Feder and Vardi, 1998; Bulatov et al., 2005; Bulatov, 2006; Bulatov and Dalmau, 2006; Dalmau, 2006; Bulatov and Valeriote, 2008; Barto et al., 2009b; Barto and Kozik, 2009; Barto et al., 2009a; Kun and Szegedy, 2009; Berman et al., 2010; Idziak et al., 2010; Bulatov, 2011b; Bodirsky and Pinsker, 2011; Bulatov, 2011a; Barto, 2011).

One example of a constraint satisfaction problem with a restricted constraint language is the Graph k-Colourability problem, described in Example 1.3, which is tractable when $k \leq 2$, but NP-complete for $k \geq 3$.

Hybrid restrictions Of course it is possible to impose other kinds of restrictions on the CSP, by restricting the possible pairs $(\mathbf{A}, \mathbf{B})$ that are allowed in instances in some other way. Such restrictions are sometimes referred to as *hybrid* restrictions (Pearson and Jeavons, 1997), because they involve simultaneous restrictions on both the source structure and the target structure. Hybrid restrictions have been much less widely studied, although some interesting cases have been identified (Cohen, 2003; Salamon and Jeavons, 2008; Cooper et al., 2010a; Cohen et al., 2011b; Fellows et al., 2011).

Related work In this chapter we focus on constraint satisfaction problems over a finite domain, but there has also been considerable work on such problems over infinite domains (Bodirsky, 2008; Bodirsky and Kára, 2010). The complexity of various forms of the CSP has recently

also been studied with respect to fixed-parameter tractability (Gottlob and Szeider, 2008; Samer and Szeider, 2010; Fellows et al., 2011).

1.3 Valued Constraint Satisfaction Problems

The standard constraint satisfaction problem, or CSP, described in the previous section, captures only the feasibility aspects of a given problem. Since in practice many problems involve seeking a solution that optimises certain criteria, as well as satisfying certain restrictions, various more general frameworks for so-called *soft* constraint satisfaction have been studied, which allow measures of desirability to be associated with different assignments to the variables (Dechter, 2003).

Several very general soft CSP frameworks have been proposed in the literature (Schiex et al., 1995; Bistarelli et al., 1997; Rossi et al., 2006), the two most general being the *semi-ring* CSP and the *valued* CSP.

The main difference between semi-ring CSPs and valued CSPs is that the measures of desirability used in valued CSPs represent costs and have to be totally ordered, whereas the measures used in semi-ring CSPs represent preferences and might be ordered only partially. Hence the semi-ring CSP framework is slightly more general than the valued CSP framework, but the valued CSP framework is sufficiently powerful to model a wide range of optimisation problems (Cohen et al., 2006b). Hence we will simply focus here on the valued CSP framework, which we will now define formally.

For a tuple t, we shall denote by $t[i]$ its i-th component. We shall denote by $\mathbb{Q}_+$ the set of all non-negative[2] rational numbers.[3] We define $\overline{\mathbb{Q}}_+ = \mathbb{Q}_+ \cup \{\infty\}$, with the standard addition operation, $+$, extended so that for all $a \in \mathbb{Q}_+$, $a + \infty = \infty$. Members of $\overline{\mathbb{Q}}_+$ are called *costs*.

A function ϕ from D^m to $\overline{\mathbb{Q}}_+$ will be called a *cost function* on D of *arity* m. If the range of ϕ is $\{\alpha, \infty\}$, for some finite $\alpha \in \mathbb{Q}_+$ then ϕ is called *essentially crisp*. If $\alpha = 0$, i.e. the range of ϕ is $\{0, \infty\}$, then ϕ is called *crisp*. If the range of ϕ lies entirely within $\mathbb{Q}_+$, then ϕ is called

[2] It is standard to define the range of cost functions as non-negative rationals with infinity. However, it is easy to observe that one could define the range to be *all* rationals with infinity. Indeed, any VCSP instance with such cost functions is equivalent to an instance with non-negative costs by adding a suitable constant.

[3] To avoid computational and representational issues, we work with rational numbers rather than arbitrary real numbers. We note that some of the algorithmic results stated in this chapter have been proved for the reals, but we will state the results only for the special case of the rationals.

finite-valued. If the range of ϕ includes both nonzero finite costs and infinity, we sometimes emphasise this fact by calling ϕ *general-valued*.

Definition 1.10 An instance of the *valued constraint satisfaction problem*, (VCSP), is a triple $\mathcal{P} = \langle V, D, \mathcal{C} \rangle$ where: V is a finite set of *variables*; D is a set of possible *values* that may be assigned to the variables (called the *domain*); $\mathcal{C}$ is a multi-set of *valued constraints*. Each element of $\mathcal{C}$ is a pair $c = \langle \sigma, \phi \rangle$ where σ is a tuple of variables called the *scope* of c, and ϕ is a $|\sigma|$-ary cost function on D taking values in $\overline{\mathbb{Q}}_+$. An *assignment* for $\mathcal{P}$ is a mapping $s : V \to D$. The cost of an assignment s, denoted $\mathrm{Cost}_{\mathcal{P}}(s)$, is given by the sum of the costs for the restrictions of s onto each constraint scope, that is,

$$\mathrm{Cost}_{\mathcal{P}}(s) \stackrel{\text{def}}{=} \sum_{\langle \langle v_1, v_2, \ldots, v_m \rangle, \phi \rangle \in \mathcal{C}} \phi(s(v_1), s(v_2), \ldots, s(v_m)) .$$

A *solution* to $\mathcal{P}$ is an assignment with minimum cost, and the question is to find a solution.

Remark In the original, more general, definition of the VCSP, as given by Bistarelli et al. (1999), costs were allowed to lie in any positive totally-ordered monoid, called a valuation structure. However, using costs from $\overline{\mathbb{Q}}_+$ and combining them using standard addition is sufficient for our purposes and standard in operational research. We refer the reader to Cooper (2005) for details on why the restriction to $\overline{\mathbb{Q}}_+$ is not severe.

There are many alternative frameworks for optimisation problems which can easily be shown to be equivalent to the valued constraint satisfaction problem. These include: Min-Sum Problems, Gibbs energy minimisation, Markov Random Fields (MRF), Conditional Random Fields (CRF) and others (Lauritzen, 1996; Boros and Hammer, 2002; Werner, 2007; Wainwright and Jordan, 2008; Crama and Hammer, 2011). Hence, all of the tractability and intractability results that we derive for the VCSP framework immediately apply to all of these other frameworks as well.

As with the CSP defined in Section 1.2, one can study structural restrictions[4] for the VCSP, and hybrid restrictions for the VCSP (see Cooper and Živný (2011a,b,c) for recent results on hybrid restrictions).

[4] The study of structural restrictions for the VCSP has not led to essentially new results as hardness results for the CSP immediately apply to the (more general) VCSP, and all known tractable structural classes for the CSP extend easily to the VCSP, see (Bertelé and Brioshi, 1972; Dechter, 2003).

However, the main focus in this chapter will be on language restrictions for the VCSP. For the remainder of the chapter we will assume that D denotes some fixed finite set D; that is, the size of D is not part of the input. A *valued constraint language* (or simply a *language*) is a set of possible cost functions mapping D^m to $\overline{\mathbb{Q}}_+$. A valued constraint language Γ is called *crisp* (*essentially crisp*, *finite-valued*, *general-valued*, respectively) if all cost functions from Γ are crisp (essentially crisp, finite-valued, general-valued, respectively). A language Γ over a two-element domain is called a *Boolean* language.

Definition 1.11 We will denote by VCSP(Γ) the class of all VCSP instances where the cost functions of the valued constraints are all contained in Γ.

A language Γ is called *tractable* if VCSP(Γ') can be solved in polynomial time for every finite subset $\Gamma' \subseteq \Gamma$, and Γ is called *intractable* if VCSP(Γ) is NP-hard for some finite $\Gamma' \subseteq \Gamma$.

Since we are interested in the complexity of languages over fixed finite domains, we can always assume an explicit representation (such as a table of values) for all cost functions from the language.

1.4 Examples of Valued Constraint Languages

We now give some examples of tractable and intractable valued constraint languages that can be used to model a wide variety of discrete combinatorial search and optimisation problems.

Example 1.12 (CSP) The standard constraint satisfaction problem defined in Section 1.2 can be seen as the special case of the VCSP where all cost functions are (essentially) crisp. In other words, the CSP can be seen as VCSP(Γ_{ecrisp}), where Γ_{ecrisp} is the language consisting of all cost functions with range $\{c, \infty\}$, for some fixed finite $c \in \mathbb{Q}_+$.

By the examples given in Section 1.2, the language Γ_{ecrisp} is clearly intractable.

Example 1.13 ((s,t)-Min-Cut) Let $G = \langle V, E \rangle$ be a directed weighted graph such that for every $(u,v) \in E$ there is a weight $w(u,v) \in \overline{\mathbb{Q}}_+$ and let $s, t \in V$ be the source and target nodes. Recall that an (s,t)-*cut* C is a subset of vertices V such that $s \in C$ but $t \notin C$. The weight, or the size, of an (s,t)-cut C is defined as $\sum_{(u,v)\in E, u\in C, v\notin C} w(u,v)$. The (s,t)-Min-Cut problem consists in finding a minimum-weight (s,t)-cut

in G. We can formulate the search for a minimum-weight (s,t)-cut in G as a VCSP instance as follows.

Let $D = \{0,1\}$. For any cost $w \in \overline{\mathbb{Q}}_+$, we define

$$\lambda_w(x,y) \stackrel{\text{def}}{=} \begin{cases} w & \text{if } x = 0 \text{ and } y = 1 \\ 0 & \text{if } x = 1 \text{ or } y = 0 \end{cases}.$$

For any value $d \in D$ and cost $c \in \overline{\mathbb{Q}}_+$, we define

$$\mu_c^d(x) \stackrel{\text{def}}{=} \begin{cases} c & \text{if } x = d \\ 0 & \text{if } x \neq d \end{cases}.$$

We denote by Γ_{cut} the set of all cost functions of the form λ_w or μ_c^d, for all $w, c \in \overline{\mathbb{Q}}_+$ and $d \in D$. Any instance of (s,t)-Min-Cut can be formulated in VCSP(Γ_{cut}) as follows:

$$\mathcal{P} = \left\langle V, D, \{\left\langle \langle u,v\rangle, \lambda_{w(u,v)}\right\rangle \mid (u,v) \in E\} \cup \{\left\langle s, \mu_\infty^1\right\rangle, \left\langle t, \mu_\infty^0\right\rangle\}\right\rangle.$$

The unary constraints ensure that the source and target nodes take the values 0 and 1, respectively, in any solution. Therefore, a minimum-weight (s,t)-cut in G corresponds to the set of variables assigned the value 0 in some solution to $\mathcal{P}$.

Furthermore, we claim that any instance $\mathcal{P}$ of VCSP(Γ_{cut}) on variables $x_1, \ldots, x_n$ can be solved in $O(n^3)$ time by a reduction to (s,t)-Min-Cut, and then using the standard algorithm (Goldberg and Tarjan, 1988). The reduction works as follows: any unary constraint μ_c^0 (respectively μ_c^1) on x_i can be modelled by an edge of weight c from x_i to the target node (respectively, from the source node to the node x_i). Any constraint with cost function $\lambda_w(x_i, x_j)$ is modelled by an edge of weight w from x_i to x_j.

Hence Γ_{cut} is tractable.

Example 1.14 (Max-Cut) Let Γ_{xor} be the language that contains just the single binary cost function $\phi_{\mathsf{xor}} : D^2 \to \overline{\mathbb{Q}}_+$ defined by

$$\phi_{\mathsf{xor}}(x,y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}.$$

If $D = \{0,1\}$, then the problem VCSP(Γ_{xor}) corresponds to the Max-SAT problem for the exclusive-or predicate, which is known to be NP-hard (Creignou et al., 2001). Therefore, Γ_{xor} is intractable. Moreover, VCSP(Γ_{xor}) is also equivalent to the Max-Cut problem, a well-known NP-complete problem (Garey and Johnson, 1979).

For $|D| > 2$, VCSP(Γ_{xor}) includes the $|D|$-Colouring problem, which is also NP-complete.

Example 1.15 (Max-CSP) An instance of the maximum constraint satisfaction problem (Max-CSP) is an instance of the CSP with the goal to maximise the number of satisfied constraints. In the weighted version, each constraint has a non-negative weight and the goal is to maximise the weighted number of satisfied constraints.

When seeking the optimal solution, maximising the weighted number of satisfied constraints is the same as minimising the weighted number of unsatisfied constraints.[5] Hence for any instance $\mathcal{P}$ of the Max-CSP, we can define a corresponding VCSP instance $\mathcal{P}'$ in which each constraint c of $\mathcal{P}$ with weight w is associated with a cost function over the same scope in $\mathcal{P}'$ which assigns cost 0 to tuples allowed by c, and cost w to tuples disallowed by c.

The complexity of Boolean languages for Max-CSP has been completely classified in (Khanna et al., 2001), see also (Creignou et al., 2001). First results on languages over arbitrary finite domains appeared in (Cohen et al., 2005). A complexity classification (with respect to approximability) of languages over three-element domains appeared in (Jonsson et al., 2006). Let Γ_{fix} be the language containing unary cost functions u_d for all $d \in D$, where $u_d(x) = 0$ if $x = d$ and $u_d(x) = 1$ if $x \neq d$. A complexity classification with respect to approximability of all languages including Γ_{fix}, (so-called languages with fixed-value constraints), was obtained in (Deineko et al., 2008).

The last two mentioned results rely heavily on computer search. However, the main results of these papers follow easily from a recent result on conservative VCSPs (Kolmogorov and Živný, 2012), which we will discuss in Section 1.7.

For recent results on approximability and inapproximability of the Max-CSP, see (Raghavendra, 2008)

A generalisation of the Max-CSP allowing both positive and negative weights has also been considered, and the complexity of all Boolean languages in this framework was classified by Jonsson (2000), later generalised to all languages by Jonsson and Krokhin (2007).

Example 1.16 (Max-Ones) An instance of the Boolean Max-Ones problem is an instance of the CSP with the goal to maximise the number of variables assigned the value 1. A classification of the complexity

[5] This statement is not true with respect to approximability, even over Boolean domains (Creignou et al., 2001).

of Boolean languages for this problem was obtained by Creignou et al. (2001). This result was later generalised to a classification of maximal languages over domains of size up to 4, and to a classification of languages containing all permutation relations by Jonsson et al. (2008).

Example 1.17 (Min-Cost-Hom) Given two graphs (directed or undirected) G and H, we denote by $V(G)$ and $V(H)$ the set of vertices of G and H respectively. We denote by $E(G)$ and $E(H)$ the set of edges of G and H respectively. A mapping $f : V(G) \to V(H)$ is a *homomorphism* of G to H if f preserves edges, that is, $(u, v) \in E(G)$ implies $(f(u), f(v)) \in E(H)$.

The homomorphism problem for graphs was described in Example 1.6 as a special case of the CSP. It asks whether an input graph G admits a homomorphism to a fixed graph H. This problem is also known as H-Colouring (Hell and Nešetřil, 1990, 2004).

For two graphs G and H, define nonnegative rational costs $c_v(u)$, for all $u \in V(G)$ and $v \in V(H)$. The cost of a homomorphism f from G to H can then be defined to be $\sum_{u \in V(G)} c_{f(u)}(u)$. For a fixed H, the Minimum-Cost Homomorphism problem (Min-Cost-Hom) asks for a homomorphism from G to H with minimum cost.

The Min-Cost-Hom problem can be seen as a special case of the VCSP, where all cost functions are either binary crisp functions, or unary finite-valued functions, and each instance has a unary cost function on each variable, and exactly one binary cost function imposed on some pairs of variables.

A complexity classification of Min-Cost-Hom for undirected graphs was obtained in (Gutin et al., 2008). A complexity classification of Min-Cost-Hom for digraphs follows from results in (Takhanov, 2010a), with generalisations in (Takhanov, 2010b).

Example 1.18 (Max-Sol) The Maximum Solution problem (Max-Sol) (Jonsson and Nordh, 2008) can be seen as a valued constraint satisfaction problem over a domain with positive integer values. It is equivalent to the VCSP over the language consisting of crisp cost functions together with all unary cost functions of the following form: $\mu(d) = wd$ for any domain value d and some fixed $w \in \mathbb{N}$. Jonsson et al. (2007) studied the Max-Sol problem over undirected graphs, that is, for languages containing only a single symmetric binary crisp cost function, and unary

functions of the form specified above.[6] Max-Sol has been also studied by Jonsson and Thapper (2009).

1.5 Expressive power

In this section, we define and study a notion of *expressibility* for valued constraint languages. This notion has played a key role in the analysis of complexity for the CSP and VCSP. Expressibility allows a particular form of problem reduction: if a constraint can be expressed in a given constraint language, then it can be added to the language without changing the computational complexity of the associated class of problems. To indicate its central role we note that the same basic idea of expressibility has been studied under many different names in different fields: implementation (Creignou et al., 2001), pp-definability (Chen, 2006), existential inverse satisfiability (Creignou et al., 2008b), structure identification (Dechter and Pearl, 1992) or join and projection operations in relational databases (Ullman, 1989). In the context of the CSP, both upper bounds (Jeavons et al., 1996, 1999) and lower bounds (Willard, 2010) on the complexity of deciding expressibility have been studied.

Expressibility is also a significant notion in practical constraint programming. As with all computing paradigms, it is desirable for many purposes to have a small language which can be used to describe a large collection of problems. Determining which additional constraints can be expressed by a given constraint language is therefore a central issue in assessing the flexibility and usefulness of a constraint system.

Definition 1.19 For any VCSP instance $\mathcal{P} = \langle V, D, C\rangle$, and any list $L = \langle v_1, \ldots, v_m\rangle$ of variables of $\mathcal{P}$, the *projection* of $\mathcal{P}$ onto L, denoted $\pi_L(\mathcal{P})$, is the m-ary cost function defined as follows:

$$\pi_L(\mathcal{P})(x_1, \ldots, x_m) \stackrel{\text{def}}{=} \min_{\{s:V\to D \mid \langle s(v_1),\ldots,s(v_m)\rangle = \langle x_1,\ldots,x_m\rangle\}} \text{Cost}_{\mathcal{P}}(s)\,.$$

We say that a cost function ϕ is *expressible* over a constraint language Γ if there exists a VCSP instance $\mathcal{P} \in \text{VCSP}(\Gamma)$ and a list L of variables of $\mathcal{P}$ such that $\pi_L(\mathcal{P}) = \phi$. We call the pair $\langle \mathcal{P}, L\rangle$ a *gadget* for expressing ϕ over Γ. We define $\langle\Gamma\rangle$ to be the *expressive power* of Γ; that is, the set of all cost functions expressible over Γ.

[6] This problem is a restriction of Min-Cost-Hom for graphs, but a generalisation of both List-Hom for graphs and Max-Ones.

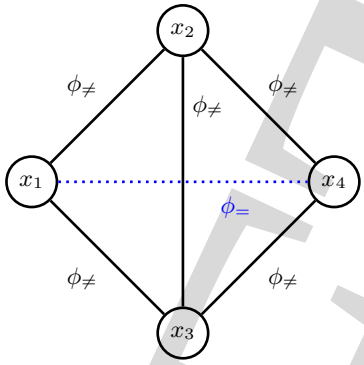

Figure 1.1 Expressing $\phi_{=}$ over $\Gamma = \{\phi_{\neq}\}$ for $|D| = 3$ (Example 1.20).

Example 1.20 Let $\Gamma = \{\phi_{\neq}\}$ be a language over D which consists of a binary crisp cost function, $\phi_{\neq}$, given by

$$\phi_{\neq}(x, y) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}.$$

Consider an instance $\mathcal{P} = \langle V, D, \mathcal{C} \rangle$ of VCSP(Γ), where $n = |D|$, $V = \{x_1, \ldots, x_{n+1}\}$, and

$$\mathcal{C} = \{\langle \langle x_i, x_j \rangle, \phi_{\neq} \rangle \mid i \neq j \in \{1, \ldots, n\}\} \cup \{\langle \langle x_i, x_{n+1} \rangle, \phi_{\neq} \rangle \mid i \in \{2, \ldots, n\}\}.$$

An example of this construction for $|D| = 3$ is shown in Figure 1.1.

In order to avoid infinite cost, variables $x_1, \ldots, x_n$ have to be assigned different values. Moreover, the value of the variable x_{n+1} has to be different from the values of the variables $x_2, \ldots, x_n$. Hence, the only remaining value that can be assigned to the variable x_{n+1} is the value which is assigned to the variable x_1. Therefore, every solution s to $\mathcal{P}$ with minimum total cost (in this case zero) satisfies $s(x_1) = s(x_{n+1})$. Therefore, $\langle \mathcal{P}, \langle x_1, x_{n+1} \rangle \rangle$ is a gadget for expressing the equality cost function, $\phi_{=}$, given by

$$\phi_{=}(x, y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = y \\ \infty & \text{if } x \neq y \end{cases}.$$

In other words, the equality cost function, $\phi_{=}$, can be expressed using the disequality cost function, $\phi_{\neq}$, that is, $\phi_{=} \in \langle \{\phi_{\neq}\} \rangle$.

The importance of Definition 1.19 is in the following:

Theorem 1.21 (Cohen et al. (2006b)) *For any valued constraint language Γ, Γ is tractable if and only if $\langle\Gamma\rangle$ is tractable.*

Consequently, in order to classify the computational complexity of all valued constraint languages, it is sufficient to consider only those languages which are closed under expressibility.

Example 1.22 By Theorem 1.21 and Example 1.14, in order to show that Γ is an intractable language it is sufficient to show that ϕ_{xor} is expressible over Γ.

Questions related to the expressive power of various valued constraint languages have been studied in (Cohen et al., 2008c; Zanuttini and Živný, 2009; Živný, 2009).

1.6 Submodular Functions and Multimorphisms

In this section, we define submodular functions and multimorphisms. We show how the minimisation problem for submodular functions can be expressed as a particular case of the VCSP, over a language characterised by having a certain binary multimorphism. Moreover, we show several generalisations which give rise to other tractable languages characterised by other kinds of multimorphisms.

Submodular functions For any finite set V, a rational-valued function f defined on subsets of V is called a *set function*.

Definition 1.23 (Submodularity) A set function $f : 2^V \to \mathbb{Q}_+$ is called *submodular* if for all subsets S and T of V,

$$f(S \cap T) + f(S \cup T) \;\leq\; f(S) + f(T). \tag{1.1}$$

Submodular functions are a key concept in operational research and combinatorial optimisation (Nemhauser and Wolsey, 1988; Narayanan, 1997; Topkis, 1998; Schrijver, 2003; Fujishige, 2005; Korte and Vygen, 2007). Examples include cuts in graphs (Goemans and Williamson, 1995; Queyranne, 1998), matroid rank functions (Edmonds, 1970), set covering problems (Feige, 1998) and entropy functions. Submodular functions are often considered to be a discrete analogue of convex functions (Lovász, 1983).

Both minimising and maximising submodular functions, possibly under some additional conditions, have been considered extensively in the

literature. Most scenarios use the so-called *oracle value model*: for any set $S \subseteq V$, an algorithm can query an oracle to find the value of $f(S)$.

Submodular function maximisation is easily shown to be NP-hard (Schrijver, 2003), but good approximation algorithms exist (Feige et al., 2011). In contrast, the submodular function minimisation problem can be solved efficiently with only polynomially many oracle calls. Since the first combinatorial[7] polynomial-time algorithms (Schrijver, 2000; Iwata et al., 2001) for minimising submodular functions, there have been several improvements in the running times, see Iwata (2008) for a survey. The time complexity of the fastest known strongly[8] polynomial time algorithm for minimising a submodular function in n variables is $O(n^6 + n^5 L)$, where L is the time required to evaluate the function in n variables (Orlin, 2009).

Binary multimorphisms We now define the concept of a binary multimorphism and give several examples.

Definition 1.24 (Binary multimorphisms) Let $\langle f, g \rangle$ be a pair of operations $f, g : D^2 \to D$. We say that an m-ary cost function $\phi : D^m \to \overline{\mathbb{Q}}_+$ *admits* $\langle f, g \rangle$ as a *multimorphism* if for all $x_1, y_1, \ldots, x_m, y_m \in D$ it holds that

$$\phi(f(x_1, y_1), \ldots, f(x_m, y_m)) + \phi(g(x_1, y_1), \ldots, g(x_m, y_m)) \leq \phi(x_1, \ldots, x_m) + \phi(y_1, \ldots, y_m). \quad (1.2)$$

If a cost function ϕ admits $\langle f, g \rangle$ as a multimorphism, we also say that ϕ is *improved* by $\langle f, g \rangle$. We say that a language Γ admits $\langle f, g \rangle$ as a multimorphism (or equivalently, that Γ is improved by $\langle f, g \rangle$), if every cost function ϕ from Γ admits $\langle f, g \rangle$ as a multimorphism.

Remark Using standard vector notation, an m-ary cost function ϕ admits $\langle f, g \rangle$ as a multimorphism if

$$\phi(f(\mathbf{x}, \mathbf{y}) + \phi(g(\mathbf{x}, \mathbf{y})) \;\leq\; \phi(\mathbf{x}) + \phi(\mathbf{y})$$

for all $\mathbf{x}, \mathbf{y} \in D^m$, where both f and g are applied coordinate-wise.

Example 1.25 (Submodular cost functions) Any set function f defined on subsets of $V = \{v_1, \ldots, v_m\}$ can be associated with a cost

[7] An algorithm is called *combinatorial* if it does not employ the ellipsoid method.
[8] An algorithm is called *strongly polynomial* if the running time does not depend on the maximum value of the function to be minimised.

function $\phi : \{0,1\}^m \to \overline{\mathbb{Q}}_+$ defined as follows: for each tuple $t \in D^m$, set $\phi(t) = f(T)$, where $T = \{v_i \mid t[i] = 1\}$.

For any tuples s, t over $\{0,1\}$, if we set $S = \{v_i \mid s[i] = 1\}$ and $T = \{v_i \mid t[i] = 1\}$, then $S \cap T = \{v_i \mid \mathrm{Min}(s[i], t[i]) = 1\}$ and $S \cup T = \{v_i | \mathrm{Max}(s[i], t[i]) = 1\}$. Here Min is the function returning the minimum of its two arguments and Max is the function returning the maximum of its two arguments. By comparing Definitions 1.24 and 1.23, it can be seen that f is submodular if and only if ϕ admits $\langle \mathrm{Min}, \mathrm{Max} \rangle$ as a multimorphism.

Example 1.26 (Submodular languages) We denote by Γ_{sub} the set of all cost functions (over some fixed finite totally ordered set D) that admit $\langle \mathrm{Min}, \mathrm{Max} \rangle$ as a multimorphism. Using a polynomial-time strongly combinatorial algorithm for minimising submodular functions, it was shown in (Cohen et al., 2006b) that Γ_{sub} is tractable.

However, an instance $\mathcal{P} \in \mathrm{VCSP}(\Gamma_{\mathsf{sub}})$ is a special kind of submodular function minimisation problem, because the objective function is given explicitly by the sum of the cost functions for each individual constraint of $\mathcal{P}$. (Note that submodularity is preserved under addition.) Such functions are also known as locally-defined functions (Cooper, 2008) or succinct functions (Feige et al., 2011). Hence, one might hope that there is an algorithm for $\mathrm{VCSP}(\Gamma_{\mathsf{sub}})$ with better running time than the general submodular function minimisation algorithms, which work in the oracle-value model and do not assume anything about the structure of the objective function.

In some special cases more efficient algorithms are known. For example, in the case when $D = \{0,1\}$, the cost functions defined in Example 1.13 are all submodular, so the language Γ_{cut} defined in Example 1.13 is strictly contained in Γ_{sub} for this set D. (Indeed, it is well known that the cut function for any graph is submodular.) Since cut functions can be minimised more efficiently than general submodular functions (see Example 1.13), classes of submodular functions from Γ_{sub} that are expressible over Γ_{cut} have been studied (Živný, 2009; Živný and Jeavons, 2010). However, it has been shown in (Živný, 2009; Živný et al., 2009) that not all functions from Γ_{sub} are expressible over Γ_{cut}, so this approach cannot be used to obtain a more efficient algorithm for the whole of $\mathrm{VCSP}(\Gamma_{\mathsf{sub}})$.

Other approaches for solving instances from $\mathrm{VCSP}(\Gamma_{\mathsf{sub}})$ include linear programming (Cooper, 2008; Cooper et al., 2010b) or submodular flows (Kolmogorov, 2010).

Example 1.27 (Max) Let Γ_{max} be a language, over some totally ordered domain D, which is improved by the pair $\langle \mathrm{Max}, \mathrm{Max} \rangle$, where $\mathrm{Max} : D^2 \to D$ is the binary operation returning the larger of its two arguments. For crisp languages, the tractability of Γ_{max} has been shown in (Jeavons and Cooper, 1995). Using this result, it was shown in (Cohen et al., 2006b) that after establishing arc consistency (Rossi et al., 2006) (which might restrict the domains of individual variables), assigning the smallest remaining domain value to all variables will yield an optimal solution. Thus Γ_{max} is tractable.

Example 1.28 (Min) Let Γ_{min} be a language, over some totally ordered domain D, which is improved by the pair $\langle \mathrm{Min}, \mathrm{Min} \rangle$, where $\mathrm{Min} : D^2 \to D$ is the binary operation returning the smaller of its two arguments. The tractability of Γ_{min} has been shown in (Cohen et al., 2006b), using a similar argument to the one in Example 1.27.

Example 1.29 (Bisubmodularity) For a given finite set V, bisubmodular functions are functions defined on pairs of disjoint subsets of V with a requirement similar to Inequality 1.1 (see (Fujishige and Iwata, 2005) for the precise definition). Examples of bisubmodular functions include rank functions of delta-matroids (Bouchet, 1987; Chandrasekaran and Kabadi, 1988). Bisubmodularity also arises in bicooperative games (Bilbao et al., 2008).

A property equivalent to bisubmodularity can be defined on cost functions on the set $D = \{0, 1, 2\}$. We define two binary operations Min_0 and Max_0 as follows:

$$\mathrm{Min}_0(x, y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } 0 \neq x \neq y \neq 0 \\ \mathrm{Min}(x, y) & \text{otherwise} \end{cases},$$

and

$$\mathrm{Max}_0(x, y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } 0 \neq x \neq y \neq 0 \\ \mathrm{Max}(x, y) & \text{otherwise} \end{cases}.$$

We denote by Γ_{bis} the set of finite-valued cost functions (i.e. with range $\mathbb{Q}_+$) that admit $\langle \mathrm{Min}_0, \mathrm{Max}_0 \rangle$ as a multimorphism. Γ_{bis} can be shown to be tractable using the results of (Fujishige and Iwata, 2005; McCormick and Fujishige, 2010).

We remark that the tractability of general-valued languages defined on $D = \{0, 1, 2\}$ and admitting $\langle \mathrm{Min}_0, \mathrm{Max}_0 \rangle$ as a multimorphism remains open. Another open question is the tractability of (even finite-valued) languages defined over D, where $|D| > 3$, and improved by $\langle \mathrm{Min}_0, \mathrm{Max}_0 \rangle$.

Example 1.30 ((Symmetric) tournament pair) A binary operation $f : D^2 \to D$ is called a *tournament* operation if (i) f is commutative, i.e., $f(x, y) = f(y, x)$ for all $x, y \in D$; and (ii) f is conservative, i.e., $f(x, y) \in \{x, y\}$ for all $x, y \in D$. The *dual* of a tournament operation is the unique tournament operation g satisfying $x \neq y \Rightarrow g(x, y) \neq f(x, y)$.

A *tournament pair* is a pair $\langle f, g\rangle$, where both f and g are tournament operations. A tournament pair $\langle f, g\rangle$ is called *symmetric* if g is the dual of f.

Let Γ be an arbitrary language that admits a symmetric tournament pair multimorphism. It has been shown in (Cohen et al., 2008b), by a reduction to the minimisation problem for submodular functions (cf. Example 1.26), that any such Γ is tractable.

Now let Γ be an arbitrary language that admits any tournament pair multimorphism. It has been shown in (Cohen et al., 2008b), by a reduction to the symmetric tournament pair case, that any such Γ is also tractable.

Example 1.31 (Tree-submodularity) Now assume that the domain values from D can be arranged into a binary tree T; i.e., a tree where each node has at most two children. Given $a, b \in T$, let P_{ab} denote the unique path in T between a and b of length (=number of edges) $d(a, b)$, and let $P_{ab}[i]$ denote the i-th vertex on P_{ab}, where $0 \leq i \leq d(a, b)$ and $P_{ab}[0] = a$. Let $\langle g_\sqcap, g_\sqcup\rangle$ be two binary operations satisfying $\{g_\sqcap(a, b), g_\sqcup(a, b)\} = \{P_{ab}[\lfloor d/2\rfloor], P_{ab}[\lceil d/2\rceil]\}$.

A language admitting $\langle g_\sqcup, g_\sqcap\rangle$ as a multimorphism has been called *strongly tree-submodular*. The tractability of strongly tree-submodular languages was shown in (Kolmogorov, 2011).

For $a, b \in T$, let $g_\wedge(a, b)$ be defined as the highest common ancestor of a and b in T; i.e. the unique node on the path P_{ab} that is ancestor of both a and b. We define $g_\vee(a, b)$ as the unique node on the path P_{ab} such that the distance between a and $g_\vee(a, b)$ is the same as the distance between b and $g_\wedge(a, b)$.

A language admitting $\langle g_\wedge, g_\vee\rangle$ as a multimorphism has been called *weakly tree-submodular*, since it has been shown that the property of strong tree-submodularity implies weak tree-submodularity (Kolmogorov, 2011). The tractability of weakly tree-submodular languages on chains[9] and forks[10] has also been shown in (Kolmogorov, 2011).

[9] A chain is a binary tree in which all nodes except leaves have exactly one child.
[10] A fork is a binary tree in which all nodes except leaves and one special node have exactly one child. The special node has exactly two children.

We remark that the tractability of strongly tree-submodular languages defined similarly on general (not necessarily binary) trees remains open. (A special case is a tree on $k+1$ vertices, where $k > 2$, consisting of a root node with k children. This corresponds precisely to bisubmodular languages on domains of size $k+1$, cf. Example 1.29.) Moreover, the tractability of weakly tree-submodular languages defined on any trees other than chains and forks remains open.

Example 1.32 (1-defect) Let b and c be two distinct elements of D and let $(D; <)$ be a partial order which relates all pairs of elements except for b and c. We call $\langle f, g \rangle$, where $f, g : D^2 \to D$ are two binary operations, a *1-defect* if f and g are both commutative and satisfy the following conditions:

- If $\{x, y\} \neq \{b, c\}$, then $f(x, y) = \mathrm{Min}(x, y)$ and $g(x, y) = \mathrm{Max}(x, y)$.
- If $\{x, y\} = \{b, c\}$, then $\{f(x, y), g(x, y)\} \cap \{x, y\} = \emptyset$, and $f(x, y) < g(x, y)$.

The tractability of languages that admit a 1-defect multimorphism has recently been shown in (Jonsson et al., 2011). This result generalises the tractability result for weakly tree-submodular languages on chains and forks described in Example 1.31, but is incomparable with the tractability result for strongly tree-submodular languages on binary trees.

General multimorphisms Having seen several examples of binary multimorphisms, we are now ready to define general (not necessarily binary) multimorphisms.

Definition 1.33 (General multimorphisms) Let $\mathbf{g} = \langle g_1, \ldots, g_k \rangle$ be a k-tuple of k-ary operations $g_i : D^k \to D$, $1 \leq i \leq k$. An m-ary cost function $\phi : D^m \to \overline{\mathbb{Q}}_+$ admits $\mathbf{g}$ as a multimorphism if

$$\sum_{i=1}^{k} \phi(g_i(\mathbf{x_1}, \ldots, \mathbf{x_k})) \;\leq\; \sum_{i=1}^{k} \phi(\mathbf{x_i}) \tag{1.3}$$

for all $\mathbf{x}_i \in D^m$, $1 \leq i \leq k$, where all functions g_i, $1 \leq i \leq k$ are applied coordinate-wise.

Definition 1.33 is illustrated in Figure 1.2, which should be read from left to right. Starting with the m-tuples $\mathbf{x}_1, \ldots, \mathbf{x}_k$, we first apply functions $g_1, \ldots, g_k$ on these tuples coordinate-wise, thus obtaining the m-

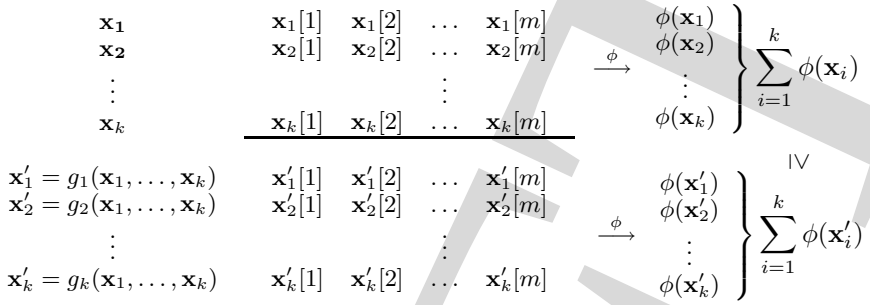

Figure 1.2 Definition of a multimorphism $\mathbf{g} = \langle g_1, \ldots, g_k \rangle$.

tuples $\mathbf{x}'_1, \ldots, \mathbf{x}'_k$. Inequality 1.3 amounts to comparing the sum of ϕ applied to tuples $\mathbf{x}_1, \ldots, \mathbf{x}_k$ and ϕ applied to tuples $\mathbf{x}'_1, \ldots, \mathbf{x}'_k$.

We now give a simple example of a unary multimorphism.

Example 1.34 (c-validity) Given a fixed $c \in D$, we say that language Γ is c-valid if for all $\phi \in \Gamma$, $\phi(c, \ldots, c) \leq \phi(\mathbf{x})$ holds for all $\mathbf{x} \in D^m$, where m is the arity of ϕ. It is clear that c-valued languages are tractable as assigning the value c to all variables yields an optimal solution.

It is easy to see that Γ is c-valid if and only if Γ admits $\langle g_c \rangle$ as a multimorphism, where $g_c : D \to D$ is defined as $g_c(x) = c$ for all $x \in D$ (i.e., g_c is the constant function returning the value c).

Example 1.35 (Majority) A ternary operation $f : D^3 \to D$ is called a majority operation if $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$ for all $x, y \in D$.

Let $\mathbf{g} = \langle g_1, g_2, g_3 \rangle$ be a triple of ternary operations such that g_1, g_2 and g_3 are all majority operations. Let $\phi : D^m \to \overline{\mathbb{Q}}_+$ be an m-ary cost function that admits $\mathbf{g}$ as a multimorphism. It follows from Definition 1.33 that for all $\mathbf{x}, \mathbf{y} \in D^m$, $3\phi(\mathbf{x}) \leq \phi(\mathbf{x}) + \phi(\mathbf{x}) + \phi(\mathbf{y})$ and $3\phi(\mathbf{y}) \leq \phi(\mathbf{y}) + \phi(\mathbf{y}) + \phi(\mathbf{x})$. Therefore, if both $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ are finite, then we have $\phi(\mathbf{x}) \leq \phi(\mathbf{y})$ and $\phi(\mathbf{y}) \leq \phi(\mathbf{x})$, and hence $\phi(\mathbf{x}) = \phi(\mathbf{y})$. In other words, the range of ϕ is $\{c, \infty\}$, for some finite $c \in \mathbb{Q}_+$, and hence ϕ is essentially crisp.

Let Γ_{Mjty} be the set of cost functions improved by a triple $\mathbf{g} = \langle g_1, g_2, g_3 \rangle$ of ternary majority operations $g_i : D^3 \to D$, $1 \leq i \leq 3$. The tractability of Γ_{Mjty} has been shown by Cohen et al. (2006b), using the earlier result that CSPs closed under a majority polymorphism are tractable (Jeavons et al., 1997).

Example 1.36 (Minority) A ternary operation $f : D^3 \to D$ is called a minority operation if $f(x, x, y) = f(x, y, x) = f(y, x, x) = y$ for all $x, y \in D$. Let Γ_{Mnty} be the set of cost functions improved by a triple $\mathbf{g} = \langle g_1, g_2, g_3 \rangle$ of ternary minority operations $g_i : D^3 \to D$, $1 \leq i \leq 3$. A similar argument to the one in Example 1.35 shows that the cost functions in Γ_{Mnty} are essentially crisp. The tractability of Γ_{Mnty} has been shown in (Cohen et al., 2006b), using the result that CSPs closed under a Mal'tsev polymorphism[11] are tractable (Dalmau, 2006).

Example 1.37 (Majority & Minority) Let $\mathbf{g} = \langle g_1, g_2, g_3 \rangle$ be three ternary operations such that g_1 and g_2 are majority operations, and g_3 is a minority operation. Let Γ_{MJN} be an arbitrary language improved by $\mathbf{g}$. The tractability of Γ_{MJN} has been shown in (Kolmogorov and Živný, 2012), generalising an earlier tractability result for a specific $\mathbf{g}$ of this form from (Cohen et al., 2006b).

Having seen several tractable languages characterised by multimorphisms, we are now able to state a dichotomy classification for Boolean languages given in (Cohen et al., 2006b).

Theorem 1.38 (Classification of Boolean languages) *An arbitrary valued constraint language on $D = \{0, 1\}$ is tractable if it admits at least one of the following eight multimorphisms. Otherwise it is intractable.*

1. $\langle g_0 \rangle$ *(i.e.,* Γ *is 0-valid),*
2. $\langle g_1 \rangle$ *(i.e.,* Γ *is 1-valid),*
3. $\langle \mathrm{Min}, \mathrm{Min} \rangle$,
4. $\langle \mathrm{Max}, \mathrm{Max} \rangle$,
5. $\langle \mathrm{Min}, \mathrm{Max} \rangle$,
6. $\langle \mathrm{Mjrty}, \mathrm{Mjrty}, \mathrm{Mjrty} \rangle$,
7. $\langle \mathrm{Mnrty}, \mathrm{Mnrty}, \mathrm{Mnrty} \rangle$,
8. $\langle \mathrm{Mjrty}, \mathrm{Mjrty}, \mathrm{Mnrty} \rangle$.

Using results from (Kolmogorov and Živný, 2012) on conservative languages, as described in the next section, a complete classification of $\{0, 1\}$-valued languages over a domain with at most 4 elements has recently appeared in (Jonsson et al., 2011). It turns out that the only two tractable classes over these domain sizes are those characterised by $\langle \mathrm{Min}, \mathrm{Max} \rangle$ multimorphisms (cf. Example 1.26) and those characterised by 1-defect multimorphisms (cf. Example 1.32).

[11] A ternary operation $f : D^3 \to D$ is called Mal'tsev if $f(x, y, y) = f(y, y, x) = x$ for all $x, y \in D$.

1.7 Conservative Valued Constraint Languages

A language Γ is called *conservative* if Γ includes all unary cost functions.[12] In this section, we will survey recent results on the complexity of conservative languages (Kolmogorov and Živný, 2012).

Let Γ be a fixed language on D. Let $P = \{(a, b) \mid a, b \in D, a \neq b\}$. Let $M \subseteq P$ be arbitrary. We denote by $\overline{M}$ the complement of M in P, i.e., $\overline{M} = P \setminus M$.

Let $\langle g_{\sqcap}, g_{\sqcup} \rangle$ be two binary operations. We call $\langle g_{\sqcap}, g_{\sqcup} \rangle$ a symmetric tournament pair (STP) on M (cf. Example 1.30) if $g_{\sqcap}$ and $g_{\sqcup}$ are conservative on D and commutative on M. Let $\langle \mathrm{Mjrty}_1, \mathrm{Mjrty}_2, \mathrm{Mnrty}_3 \rangle$ be three ternary operations. We call $\langle \mathrm{Mjrty}_1, \mathrm{Mjrty}_2, \mathrm{Mnrty}_3 \rangle$ an MJN on $\overline{M}$ if operations $\mathrm{Mjrty}_1, \mathrm{Mjrty}_2, \mathrm{Mnrty}_3$ are conservative and for each triple $\langle a, b, c \rangle \in D^3$ with $\{a, b, c\} = \{x, y\} \in \overline{M}$ operations $\mathrm{Mjrty}_1(a, b, c)$, $\mathrm{Mjrty}_2(a, b, c)$ return the unique majority element among a, b, c (that occurs twice) and $\mathrm{Mnrty}_3(a, b, c)$ returns the remaining minority element.

Given a language Γ, we say that Γ admits *complementary STP and MJN* multimorphisms if there is a set $M \subseteq P$, binary operations $\langle g_{\sqcap}, g_{\sqcup} \rangle$ and ternary operations $\langle \mathrm{Mjrty}_1, \mathrm{Mjrty}_2, \mathrm{Mnrty}_3 \rangle$ such that $\langle g_{\sqcap}, g_{\sqcup} \rangle$ is an STP on M and $\langle \mathrm{Mjrty}_1, \mathrm{Mjrty}_2, \mathrm{Mnrty}_3 \rangle$ is an MJN on $\overline{M}$, where $\overline{M} = P \setminus M$.

Generalising the tractability results described in Examples 1.30 and 1.37, Kolmogorov and Živný (2012) have shown that languages admitting complementary STP and MJN multimorphisms are tractable. Moreover, for general-valued conservative languages, it has been shown that this class is the *only* tractable case, as the following result from (Kolmogorov and Živný, 2012) indicates.

Theorem 1.39 (General-valued conservative languages) *A general-valued conservative valued constraint language is tractable if it admits complementary STP and MJN multimorphisms. Otherwise it is intractable.*

In the finite-valued case, the more restricted class from Example 1.30 is the only tractable case, as the following result from (Kolmogorov and Živný, 2012) indicates.

Theorem 1.40 (Finite-valued conservative languages) *A finite-valued conservative valued constraint language is tractable if it admits a symmetric-tournament pair multimorphism. Otherwise it is intractable.*

[12] Kolmogorov and Živný (2012) have shown that this condition is polynomial-time equivalent to the (weaker) requirement that Γ includes all $\{0, 1\}$-valued unary cost functions.

These results for conservative languages over arbitrary finite domains are obtained by considering restrictions to all 2-element subdomains, using unary constraints. The possible tractable languages for 2-element domains are already known (see Theorem 1.38) and highly restricted.

1.8 A General Algebraic Theory of Complexity

One of the three equivalent definitions of the CSP presented in Section 1.2 uses the terminology of algebra (Definition 1.5). This formulation led to the development of an algebraic approach for classifying the complexity of crisp constraint languages, started by Jeavons et al. (1997), which has so far been the most successful technique for precisely identifying the tractable cases (Bulatov et al., 2005; Bulatov and Valeriote, 2008; Bulatov, 2011b).

Cohen et al. (2011a) have recently presented an extension of this algebraic approach to the more general setting of valued constraint languages.

The basic idea of this extended algebraic theory is to associate with each valued constraint language a set of functions, called *weighted polymorphisms*. These objects are similar to the multimorphisms defined in Section 1.6 (see Example 1.46), but are defined in a slightly more general way. The theory then establishes that there is a one-to-one correspondence between certain sets of weighted polymorphisms and valued constraint languages which are closed under expressibility and scaling. This correspondence simplifies the search for tractable languages to a search for suitable sets of weighted polymorphisms. For example, Creed and Živný (2011) have used this result to obtain the classification of Boolean valued languages presented in Theorem 1.38 using simple algebraic arguments.

In this section, we will give a brief overview of the main results of this new algebraic theory. We refer the reader to (Cohen et al., 2011a) for full details and proofs.

We first recall some basic terminology from universal algebra (Denecke and Wismath, 2002; Börner, 2008). A function $f : D^k \to D$ is called a k-ary *operation* on D. We denote by $\mathbf{O}_D$ the set of all finitary operations on D and by $\mathbf{O}_D^{(k)}$ the k-ary operations in $\mathbf{O}_D$. The k-ary *projections* on D, defined for $i = 1, \ldots, k$, are the operations $\mathrm{e}_i^{(k)}$ such that $\mathrm{e}_i^{(k)}(a_1, \ldots, a_k) = a_i$. Let $f \in \mathbf{O}_D^{(k)}$ and $g_1, \ldots, g_k \in \mathbf{O}_D^{(l)}$. The *su-*

perposition of f and $g_1, \ldots, g_k$ is the l-ary operation $f[g_1, \ldots, g_k]$ such that $f[g_1, \ldots, g_k](x_1, \ldots, x_l) = f(g_1(x_1, \ldots, x_l), \ldots, g_k(x_1 \ldots, x_l))$.

A set $F \subseteq \mathbf{O}_D$ is called a *clone* of operations if it contains all the projections on D and is closed under superposition. For each $F \subseteq \mathbf{O}_D$ we define Clone(F) to be the smallest clone containing F. For any clone C, we use $C^{(k)}$ to denote the k-ary terms in C.

Definition 1.41 A *Galois connection* between two sets A and B is a pair $\langle F, G\rangle$ of mappings between the power sets $\mathcal{P}(A)$ and $\mathcal{P}(B)$, $F : \mathcal{P}(A) \to \mathcal{P}(B)$ and $G : \mathcal{P}(B) \to \mathcal{P}(B)$, such that for all $X, X' \subseteq A$ and all $Y, Y' \subseteq B$ the following conditions are satisfied:

1. $X \subseteq G(F(X))$,
2. $Y \subseteq F(G(Y))$,
3. $X \subseteq X' \;\Rightarrow\; F(X) \supseteq F(X')$,
4. $Y \subseteq Y' \;\Rightarrow\; G(Y) \supseteq G(Y')$.

The algebraic theory of complexity for crisp constraint languages (Bulatov et al., 2005; Bulatov and Valeriote, 2008) makes use of a standard Galois connection between the set of all relations on a fixed set D and the set of all operations on D (Denecke and Wismath, 2002). Using this result, Jeavons (1998) showed that there was a one-to-one correspondence between crisp constraint languages over a finite set D which are closed under expressibility and clones of operations on D.

To extend this approach to valued constraint languages, we make the following definitions. We denote by $\mathbf{\Phi}_D$ the set of all cost functions on D taking values in $\overline{\mathbb{Q}}_+$.

Definition 1.42 A valued constraint language $\Gamma \subseteq \mathbf{\Phi}_D$ is called a *weighted relational clone* if it is closed under expressibility, scaling by non-negative rational constants, and addition of rational constants.

We define wRelClone(Γ) to be the smallest weighted relational clone containing Γ.

It follows from Theorem 1.21, and the results of (Cohen et al., 2006a), that Γ is tractable if and only if wRelClone(Γ) is tractable.

Definition 1.43 We define a k-ary *weighted operation* on a set D to be a (partial) function $\omega : \mathbf{O}_D^{(k)} \to \mathbb{Q}$ such that $\omega(f) < 0$ only if f is a projection and

$$\sum_{f \in \mathbf{dom}(\omega)} \omega(f) = 0\,.$$

The *domain* of ω, denoted $\mathbf{dom}(\omega)$, is the subset of $\mathbf{O}_D^{(k)}$ on which ω is defined.

We denote by $\mathbf{W}_D$ the set of all finitary weighted operations on D.

Definition 1.44 Let C be a clone of operations on D. A *weighted clone supported* by C is a set of weighted operations with domain $C^{(k)}$ for some k, which is closed under:

1. **proper translation** Given a k-ary weighted operation $\omega : C^{(k)} \to \mathbb{Q}$ and a list of operations $g_1, \ldots, g_k \in C^{(\ell)}$, we define the *translation* of ω by $g_1, \ldots, g_k$, denoted $\omega[g_1, \ldots, g_k]$, to be the function $\omega' : C^{(\ell)} \to \mathbb{Q}$ satisfying
$$\omega'(f') = \sum_{f \in C^{(k)} : f' = f[g_1, \ldots, g_k]} \omega(f)\,,$$
for each $f' \in C^{(\ell)}$. A translation is called a *proper translation* if ω' is a weighted operation.
2. **addition** Given a pair of k-ary weighted operations $\omega_1, \omega_2 : C^{(k)} \to \mathbb{Q}$, we define the *addition* $\omega_1 + \omega_2$ to be the weighted operation ω' satisfying $\omega'(f) = \omega_1(f) + \omega_2(f)$, for each $f \in C^{(k)}$.
3. **scaling** Given a k-ary weighted operation $\omega : C^{(k)} \to \mathbb{Q}$ and a rational $\alpha \geq 0$, we define the α-scaling of ω, $\alpha\omega$, to be the weighted operation ω' satisfying $\omega'(f) = \alpha\omega(f)$, for each $f \in C^{(k)}$.

For each $W \subseteq \mathbf{W}_D$ we define $\mathrm{wClone}(W)$ to be the smallest weighted clone containing W.

Definition 1.45 We say that a weighted operation ω of arity k is a *weighted polymorphism* of the cost function ϕ of arity r if, for any $x_1, x_2, \ldots, x_k \in D^r$ such that $\phi(x_i) < \infty$ for $i = 1, \ldots, k$, we have
$$\sum_{f \in \mathbf{dom}(\omega)} \omega(f)\phi(f(x_1, x_2, \ldots, x_k)) \leq 0\,. \tag{1.4}$$
If ω is a weighted polymorphism of ϕ we also say ϕ is *improved* by ω.

Example 1.46 Consider the class of submodular cost functions from Example 1.25. These are precisely the cost functions satisfying
$$\phi(\mathrm{Min}(x_1, x_2)) + \phi(\mathrm{Max}(x_1, x_2)) - \phi(x) - \phi(y) \;\leq\; 0\,.$$
In other words, the set of submodular functions are defined as the set of

cost functions with the 2-ary weighted polymorphism

$$\omega(f) \stackrel{\text{def}}{=} \begin{cases} -1 & \text{if } f \in \{e_1^{(2)}, e_2^{(2)}\} \\ +1 & \text{if } f \in \{\text{Min}, \text{Max}\} \end{cases} .$$

Definition 1.47 For any $\Gamma \subseteq \mathbf{\Phi}_D$, we denote by $\text{wPol}(\Gamma)$ the set of all finitary weighted operations on D which are weighted polymorphisms of all cost functions $\phi \in \Gamma$.

Definition 1.48 For any $W \subseteq \mathbf{W}_D$, we denote by $\text{Imp}(W)$ the set of all cost functions in $\mathbf{\Phi}_D$ that are improved by all weighted operations $\omega \in W$.

It follows immediately from Definition 1.41 that, for any set D, the mappings wPol and Imp form a Galois connection between $\mathbf{W}_D$ and $\mathbf{\Phi}_D$. A characterisation of this Galois connection for finite sets D is given by the following theorem from Cohen et al. (2011a):

Theorem 1.49 (Galois Connection for Valued Constraint Languages)
1. For any finite sets D and $\Gamma \subseteq \mathbf{\Phi}_D$, $\text{Imp}(\text{wPol}(\Gamma)) = \text{wRelClone}(\Gamma)$.
2. For any finite sets D and $W \subseteq \mathbf{W}_D$, $\text{wPol}(\text{Imp}(W)) = \text{wClone}(W)$.

Theorem 1.49 is depicted in Figure 1.3. Using the standard properties of Galois connections, it follows that there is a one-to-one correspondence between weighted relational clones on a finite set D and weighted clones on D. Hence, to identify all tractable valued constraint languages on a finite set D it is sufficient to study the possible weighted clones on D. This provides a new approach to the identification of tractable cases, which we hope will prove to be as successful as the algebraic approach has been in the study of crisp constraint languages.

1.9 Conclusions and Open Problems

We have seen in this chapter that the valued constraint satisfaction problem is a powerful general framework that can be used to express many standard combinatorial optimisation problems. The general problem is NP-hard, but there are many special cases that have been shown to be tractable. In particular, by considering restrictions on the cost functions we allow in problem instances, we have identified a range of different sets of cost functions that ensure tractability.

These restricted sets of cost functions are referred to as valued constraint languages, and we have seen that such languages can sometimes

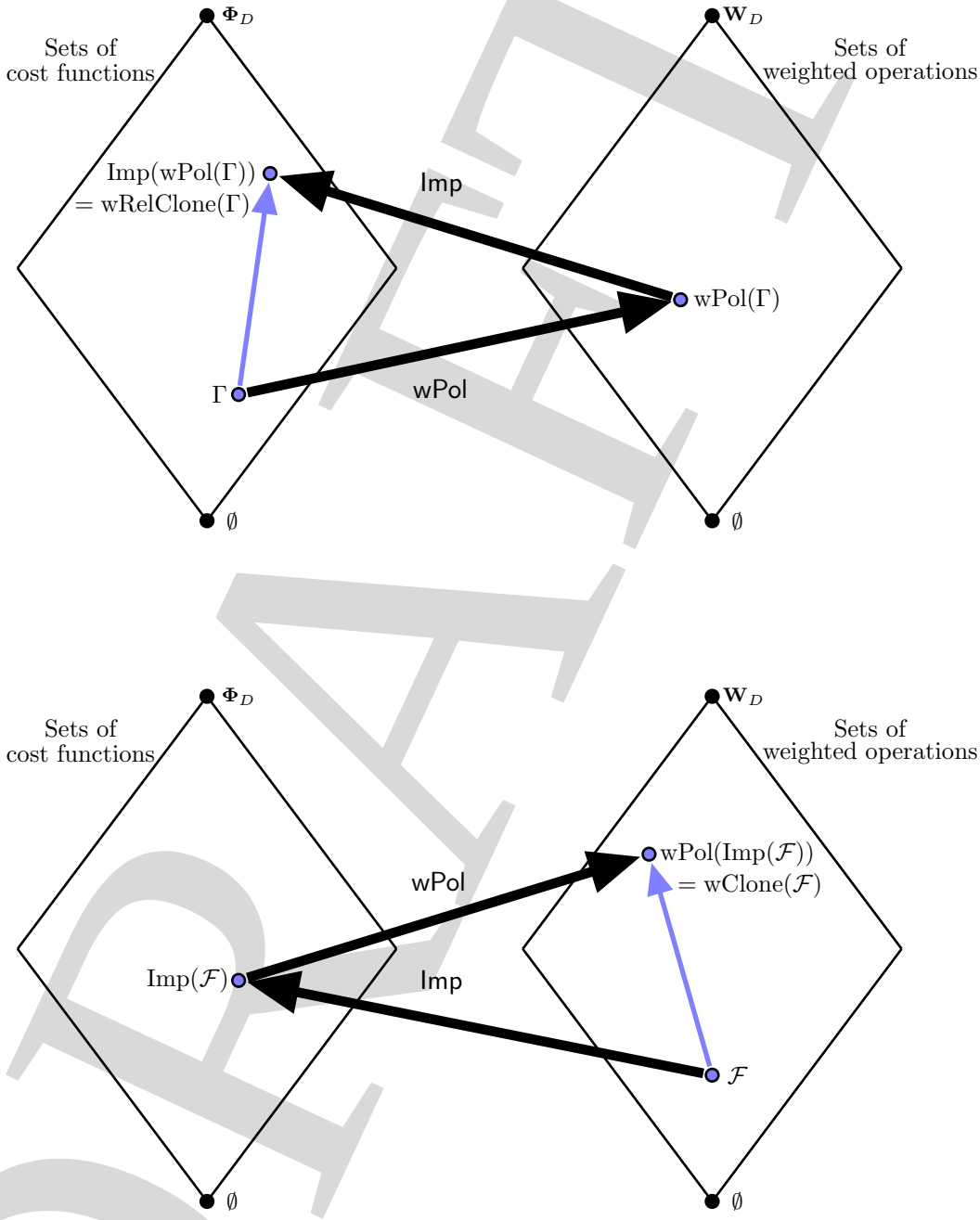

Figure 1.3 Galois connection between $\mathbf{\Phi}_D$ and $\mathbf{W}_D$.

be shown to be tractable, and sometimes be shown to be NP-hard. Some powerful algebraic techniques are now being developed to carry out this classification, as described in Section 1.8, but it is still far from complete.

In fact, even in the special case of the CSP, discussed in Section 1.2, there is still no complete classification of complexity for the corresponding crisp constraint languages, although many partial results have been obtained over the past 15 years. In particular, it has been shown that the tractable cases fall into two broad groups. The first of these are the problems that can be solved by some form of local consistency (Barto and Kozik, 2009), and the second are the problems that have a polynomial-sized generating set (Idziak et al., 2010). These two cases are both characterised by specific algebraic conditions.

Using the same argument as in Example 1.35, it can be shown that the second condition gives only essentially crisp languages. Hence only the first condition gives rise to interesting tractable valued constraint languages. Local consistency techniques have been generalised to the VCSP, but their power is not fully understood (Cooper et al., 2010b).

Even the classic tractable problem of submodular function minimisation, discussed in Section 1.6, is not fully understood. As discussed in Example 1.26, it is currently unknown whether VCSP instances with submodular cost functions can be solved more efficiently than general submodular function minimisation. Moreover, the complexity of submodular function minimisation over a domain D which is partially ordered by a lattice ordering is still unknown for non-distributive lattices. However, it has been shown that there is a pseudo-polynomial-time algorithm for diamonds (Kuivinen, 2011), and several constructions on lattices preserving tractability have been identified (Krokhin and Larose, 2008).

The complexity of VCSP instances with bisubmodular cost functions on domains of size greater than 3 is open (cf. Example 1.29). The complexity of strongly tree-submodular languages on general (i.e. non-binary) trees as well as the complexity of weakly tree-submodular languages on binary (let alone general) trees also remain open (cf. Example 1.31). In fact, even the complexity of finite-valued (let alone general-valued) languages on domains of size 3 is open, whereas crisp languages on domains of size 3 have been classified in Bulatov (2006).

References

Adler, Isolde, Gottlob, Georg, and Grohe, Martin. 2007. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, **28**(8), 2167–2181.

Apt, Krzysztof. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Barto, Libor. 2011. The dichotomy for conservative constraint satisfaction problems revisited. In: *Proceedings of the 26th IEEE Symposium on Logic in Computer Science (LICS'11)*.

Barto, Libor, and Kozik, Marcin. 2009. Constraint Satisfaction Problems of Bounded Width. Pages 461–471 of: *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS'09)*.

Barto, Libor, Kozik, Marcin, Maróti, Miklós, and Niven, Todd. 2009a. CSP dichotomy for special triads. *Proceedings of the American Mathematical Society*, **137**(9), 2921–2934.

Barto, Libor, Kozik, Marcin, and Niven, Todd. 2009b. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). *SIAM Journal on Computing*, **38**(5), 1782–1802.

Berman, Joel, Idziak, Pawel, Marković, Petar, McKenzie, Ralph, Valeriote, Matthew, and Willard, Ross. 2010. Varieties with few subalgebras of powers. *Transactions of the American Mathematical Society*, **362**(3), 1445–1473.

Bertelé, Umberto, and Brioshi, Francesco. 1972. *Nonserial dynamic programming*. Academic Press.

Bilbao, Jesús M., Fernández, Julio R., Jiménez, Nieves, and López, Jorge J. 2008. Survey of Bicooperative Games. In: Chinchuluun, Altannar, Pardalos, Panos M., Migdalas, Athanasios, and Pitsoulis, Leonidas (eds), *Pareto Optimality, Game Theory And Equilibria*. Springer.

Bistarelli, Stefano, Montanari, Ugo, and Rossi, Francesca. 1997. Semiring-based Constraint Satisfaction and Optimisation. *Journal of the ACM*, **44**(2), 201–236.

Bistarelli, Stefano, Montanari, Ugo, Rossi, Francesca, Schiex, Thomas, Verfaillie, Gérard, and Fargier, Hélène. 1999. Semiring-based CSPs and Valued

CSPs: Frameworks, Properties, and Comparison. *Constraints*, **4**(3), 199–240.

Bodirsky, Manuel. 2008. Constraint Satisfaction Problems with Infinite Templates. Pages 196–228 of: *Complexity of Constraints*. Lecture Notes in Computer Science, vol. 5250. Springer.

Bodirsky, Manuel, and Kára, Jan. 2010. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, **57**(2).

Bodirsky, Manuel, and Pinsker, Michael. 2011. Schaefer's theorem for graphs. Pages 655–664 of: *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC'11)*.

Börner, Ferdinand. 2008. Basics of Galois Connections. Pages 38–67 of: *Complexity of Constraints*. Lecture Notes in Computer Science, vol. 5250. Springer.

Boros, Endre, and Hammer, Peter L. 2002. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, **123**(1-3), 155–225.

Bouchet, André. 1987. Greedy algorithm and symmetric matroids. *Mathematical Programming*, **38**(1), 147–159.

Bulatov, Andrei. 2006. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, **53**(1), 66–120.

Bulatov, Andrei, and Dalmau, Víctor. 2006. A Simple Algorithm for Mal'tsev Constraints. *SIAM Journal on Computing*, **36**(1), 16–27.

Bulatov, Andrei, Krokhin, Andrei, and Jeavons, Peter. 2005. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, **34**(3), 720–742.

Bulatov, Andrei A. 2011a. Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, **12**(4), 24.

Bulatov, Andrei A. 2011b. On the CSP Dichotomy Conjecture. Pages 331–344 of: *Proceedings of the 6th International Computer Science Symposium in Russia (CSR'11)*. Lecture Notes in Computer Science, vol. 6651. Springer.

Bulatov, Andrei A., and Valeriote, Matthew. 2008. Recent Results on the Algebraic Approach to the CSP. Pages 68–92 of: *Complexity of Constraints*. Lecture Notes in Computer Science, vol. 5250. Springer.

Chandrasekaran, Ramaswamy, and Kabadi, Santosh N. 1988. Pseudomatroids. *Discrete Mathematics*, **71**(3), 205–217.

Chen, Hubie. 2006. A rendezvous of logic, complexity, and algebra. *SIGACT News*, **37**(4), 85–114.

Chen, Hubie, and Dalmau, Víctor. 2005. Beyond Hypertree Width: Decomposition Methods Without Decompositions. Pages 167–181 of: *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*. Lecture Notes in Computer Science, vol. 3709. Springer.

Cohen, David, and Jeavons, Peter. 2006. The complexity of constraint languages. In: Rossi, F., van Beek, P., and Walsh, T. (eds), *The Handbook of Constraint Programming*. Elsevier.

Cohen, David, Cooper, Martin, Jeavons, Peter, and Krokhin, Andrei. 2005. Supermodular Functions and the Complexity of MAX-CSP. *Discrete Applied Mathematics*, **149**(1-3), 53–72.

Cohen, David, Jeavons, Peter, and Gyssens, Marc. 2008a. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, **74**(5), 721–743.

Cohen, David A. 2003. A New Class of Binary CSPs for which Arc-Constistency Is a Decision Procedure. Pages 807–811 of: *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)*. Lecture Notes in Computer Science, vol. 2833. Springer.

Cohen, David A., Cooper, Martin C., and Jeavons, Peter G. 2006a. An Algebraic Characterisation of Complexity for Valued Constraints. Pages 107–121 of: *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP'06)*. Lecture Notes in Computer Science, vol. 4204. Springer.

Cohen, David A., Cooper, Martin C., Jeavons, Peter G., and Krokhin, Andrei A. 2006b. The Complexity of Soft Constraint Satisfaction. *Artificial Intelligence*, **170**(11), 983–1016.

Cohen, David A., Cooper, Martin C., and Jeavons, Peter G. 2008b. Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theoretical Computer Science*, **401**(1-3), 36–51.

Cohen, David A., Jeavons, Peter G., and Živný, Stanislav. 2008c. The expressive power of valued constraints: Hierarchies and collapses. *Theoretical Computer Science*, **409**(1), 137–153.

Cohen, David A., Creed, Páidí, Jeavons, Peter G., and Živný, Stanislav. 2011a. An algebraic theory of complexity for valued constraints: Establishing a Galois connection. Pages 231–242 of: *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*. Lecture Notes in Computer Science, vol. 6907. Springer.

Cohen, David A., Cooper, Martin C., Green, Martin, and Marx, Dániel. 2011b. On guaranteeing polynomially-bounded search tree size. Pages 160–171 of: *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*. Lecture Notes in Computer Science, vol. 6876. Springer.

Cooper, Martin C. 2005. High-order Consistency in Valued Constraint Satisfaction. *Constraints*, **10**(3), 283–305.

Cooper, Martin C. 2008. Minimization of Locally Defined Submodular Functions by Optimal Soft Arc Consistency. *Constraints*, **13**(4), 437–458.

Cooper, Martin C., and Živný, Stanislav. 2011a. Hierarchically nested convex VCSP. Pages 187–194 of: *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*. Lecture Notes in Computer Science, vol. 6876. Springer.

Cooper, Martin C., and Živný, Stanislav. 2011b. Hybrid tractability of valued constraint problems. *Artificial Intelligence*, **175**(9-10), 1555–1569.

Cooper, Martin C., and Živný, Stanislav. 2011c. Tractable triangles. Pages 195–209 of: *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*. Lecture Notes in Computer Science, vol. 6876. Springer.

Cooper, Martin C., Jeavons, Peter G., and Salamon, András Z. 2010a. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artificial Intelligence*, **174**(9–10), 570–584.

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

Crama, Yves, and Hammer, Peter L. 2011. *Boolean Functions - Theory, Algorithms, and Applications*. Cambridge University Press.

Creed, Páidí, and Živný, Stanislav. 2011. On minimal weighted clones. Pages 210–224 of: *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*. Lecture Notes in Computer Science, vol. 6876. Springer.

Creignou, Nadaia, Kolaitis, Phokion G., and Vollmer, Heribert (eds). 2008a. *Complexity of Constraints: An Overview of Current Research Themes*. Lecture Notes in Computer Science, vol. 5250. Springer.

Creignou, Nadia, Khanna, Sanjeev, and Sudan, Madhu. 2001. *Complexity Classification of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications, vol. 7. SIAM.

Creignou, Nadia, Kolaitis, Phokion G., and Zanuttini, Bruno. 2008b. Structure identification of Boolean relations and plain bases for co-clones. *Journal of Computer and System Sciences*, **74**(7), 1103–1115.

Dalmau, Víctor. 2006. Generalized Majority-Minority Operations are Tractable. *Logical Methods in Computer Science*, **2**(4).

Dalmau, Víctor, Kolaitis, Phokion G., and Vardi, Moshe Y. 2002. Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. Pages 310–326 of: *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*. Lecture Notes in Computer Science, vol. 2470. Springer.

Dechter, Rina. 2003. *Constraint Processing*. Morgan Kaufmann.

Dechter, Rina, and Pearl, Judea. 1989. Tree Clustering for Constraint Networks. *Artificial Intelligence*, **38**, 353–366.

Dechter, Rina, and Pearl, Judea. 1992. Structure Identification in Relational Data. *Artificial Intelligence*, **58**, 237–270.

Deineko, Vladimir, Jonsson, Peter, Klasson, Mikael, and Krokhin, Andrei. 2008. The approximability of Max CSP with fixed-value constraints. *Journal of the ACM*, **55**(4).

Denecke, Klaus, and Wismath, Shelly L. 2002. *Universal Algebra and Applications in Theoretical Computer Science*. Chapman and Hall/CRC Press.

Edmonds, Jack. 1970. Submodular Functions, Matroids, and Certain Polyhedra. *Combinatorial Structures and Their Applications*, 69–87.

Feder, Tomás, and Vardi, Moshe Y. 1998. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, **28**(1), 57–104.

Feige, Uriel. 1998. A Threshold of ln n for Approximating Set Cover. *Journal of the ACM*, **45**(4), 634–652.

Feige, Uriel, Mirrokni, Vahab S., and Vondrák, Jan. 2011. Maximizing Non-monotone Submodular Functions. *SIAM Journal on Computing*, **40**(4), 1133–1153.

Fellows, Michael R., Friedrich, Tobias, Hermelin, Danny, Narodytska, Nina, and Rosamond, Frances A. 2011. Constraint Satisfaction Problems: Convexity Makes AllDifferent Constraints Tractable. Pages 522–527 of: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*.

Freuder, Eugene C. 1985. A Sufficient Condition for Backtrack-bounded Search. *Journal of the ACM*, **32**, 755–761.

Freuder, Eugene C. 1990. Complexity of K-Tree Structured Constraint Satisfaction Problems. Pages 4–9 of: *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*.

Fujishige, Satoru. 2005. *Submodular Functions and Optimization*. 2nd edn. Annals of Discrete Mathematics, vol. 58. Amsterdam: North-Holland.

Fujishige, Satoru, and Iwata, Satoru. 2005. Bisubmodular Function Minimization. *SIAM Journal on Discrete Mathematics*, **19**(4), 1065–1073.

Garey, Michael R., and Johnson, David S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

Goemans, Michel X., and Williamson, David P. 1995. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, **42**(6), 1115–1145.

Goldberg, Andrew V., and Tarjan, Rober Endre. 1988. A New Approach to the Maximum Flow Problem. *Journal of the ACM*, **35**(4), 921–940.

Gottlob, Georg, and Szeider, Stefan. 2008. Fixed-parameter algorithms For Artificial Intelligence, Constraint Satisfaction and Database Problems. *The Computer Journal*, **51**(3), 303–325.

Gottlob, Georg, Leone, Nicola, and Scarcello, Francesco. 2000. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, **124**(2), 243–282.

Gottlob, Georg, Leone, Nicola, and Scarcello, Francesco. 2002. Hypertree decomposition and tractable queries. *Journal of Computer and System Sciences*, **64**(3), 579–627.

Gottlob, Georg, Miklós, Zoltán, and Schwentick, Thomas. 2009. Generalized Hypertree Decompositions: NP-Hardness and Tractable Variants. *Journal of the ACM*, **56**(6).

Grohe, Martin. 2007. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, **54**(1), 1–24.

Grohe, Martin, and Marx, Dániel. 2006. Constraint solving via fractional edge covers. Pages 289–298 of: *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*.

Gutin, Gregory, Hell, Pavol, Rafiey, Arash, and Yeo, Anders. 2008. A dichotomy for minimum cost graph homomorphisms. *European Journal of Combinatorics*, **29**(4), 900–911.

Gyssens, Marc, Jeavons, Peter G., and Cohen, David A. 1994. Decomposing Constraint Satisfaction Problems Using Database Techniques. *Artificial Intelligence*, **66**(1), 57–89.

Hell, Pavol, and Nešetřil, Jaroslav. 1990. On the Complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, **48**(1), 92–110.

Hell, Pavol, and Nešetřil, Jaroslav. 2004. *Graphs and Homomorphisms*. Oxford University Press.

Hell, Pavol, and Nešetřil, Jaroslav. 2008. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, **2**(3), 143–163.

Idziak, Pawel M., Markovic, Petar, McKenzie, Ralph, Valeriote, Matthew, and Willard, Ross. 2010. Tractability and Learnability Arising from Algebras with Few Subpowers. *SIAM Journal on Computing*, **39**(7), 3023–3037.

Iwata, Satoru. 2008. Submodular Function Minimization. *Mathematical Programming*, **112**(1), 45–64.

Iwata, Satoru, Fleischer, Lisa, and Fujishige, Satoru. 2001. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, **48**(4), 761–777.

Jeavons, Peter G. 1998. On the Algebraic Structure of Combinatorial Problems. *Theoretical Computer Science*, **200**(1-2), 185–204.

Jeavons, Peter G. 2009. Presenting Constraints. Pages 1–15 of: *Proceedings of the 18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'09)*. Lecture Notes in Artificial Intelligence, vol. 5607. Springer.

Jeavons, Peter G., and Cooper, Martin C. 1995. Tractable Constraints on Ordered Domains. *Artificial Intelligence*, **79**(2), 327–339.

Jeavons, Peter G., Cohen, David A., and Gyssens, Marc. 1996. A Test for Tractability. Pages 267–281 of: *Proceedings of the 2nd International Conference on Constraint Programming (CP'96), 1996*. Lecture Notes in Computer Science, vol. 1118. Springer.

Jeavons, Peter G., Cohen, David A., and Gyssens, Marc. 1997. Closure Properties of Constraints. *Journal of the ACM*, **44**(4), 527–548.

Jeavons, Peter G., Cohen, David A., and Gyssens, Marc. 1999. How to Determine the Expressive Power of Constraints. *Constraints*, **4**(2), 113–131.

Jonsson, Peter. 2000. Boolean constraint satisfaction: complexity results for optimization problems with arbitrary weights. *Theoretical Computer Science*, **244**(1-2), 189–203.

Jonsson, Peter, and Krokhin, Andrei. 2007. Maximum H-colourable subdigraphs and constraint optimization with arbitrary weights. *Journal of Computer and System Sciences*, **73**(5), 691–702.

Jonsson, Peter, and Nordh, Gustav. 2008. Introduction to the Maximum Solution Problem. Pages 255–282 of: *Complexity of Constraints*. Lecture Notes in Computer Science, vol. 5250. Springer.

Jonsson, Peter, and Thapper, Johan. 2009. Approximability of the Maximum Solution Problem for Certain Families of Algebras. Pages 215–226 of: *Proceedings of the 4th International Computer Science Symposium in Russia (CSR'09)*. Lecture Notes in Computer Science, vol. 5675. Springer.

Jonsson, Peter, Klasson, Mikael, and Krokhin, Andrei. 2006. The Approximability of Three-valued MAX CSP. *SIAM Journal on Computing*, **35**(6), 1329–1349.

Jonsson, Peter, Nordh, Gustav, and Thapper, Johan. 2007. The Maximum Solution Problem on Graphs. Pages 228–239 of: *Proceedings of the 32nd International Symposium on Mathematical Foundations of Computer Science (MFCS'07)*. Lecture Notes in Computer Science, vol. 4708. Springer.

Jonsson, Peter, Kuivinen, Fredrik, and Nordh, Gustav. 2008. MAX ONES Generalized to Larger Domains. *SIAM Journal on Computing*, **38**(1), 329–365.

Jonsson, Peter, Kuivinen, Fredrik, and Thapper, Johan. 2011. Min CSP on Four Elements: Moving Beyond Submodularity. Pages 438–453 of: *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*. Lecture Notes in Computer Science, vol. 6876. Springer.

Khanna, Sanjeev, Sudan, Madhu, Trevisan, Luca, and Williamson, David. 2001. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, **30**(6), 1863–1920.

Kolaitis, Phokion G., and Vardi, Moshe Y. 2000. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, **61**(2), 302–332.

Kolaitis, Phokion G., and Vardi, Moshe Y. 2007. A Logical Approach to Constraint Satisfaction. In: *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Kolmogorov, Vladimir. 2010 (June). *Minimizing a sum of submodular functions*. Tech. rept. arXiv:1006.1990.

Kolmogorov, Vladimir. 2011. Submodularity on a tree: Unifying $L^\natural$-convex and bisubmodular functions. Pages 400–411 of: *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*. Lecture Notes in Computer Science, vol. 6907. Springer.

Kolmogorov, Vladimir, and Živný, Stanislav. 2012. The complexity of conservative valued CSPs. Pages 750–759 of: *Proceedings of of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*. SIAM. full version available on arXiv:1110.2809.

Korte, Bernhard, and Vygen, Jens. 2007. *Combinatorial Optimization*. 4th edn. Algorithms and Combinatorics, vol. 21. Springer.

Krokhin, Andrei, and Larose, Benoit. 2008. Maximizing Supermodular Functions on Product Lattices, with Application to Maximum Constraint Satisfaction. *SIAM Journal on Discrete Mathematics*, **22**(1), 312–328.

Kuivinen, Fredrik. 2011. On the complexity of submodular function minimisation on diamonds. *Discrete Optimization*, **8**(3), 459–477.

Kun, Gabor, and Szegedy, Mario. 2009. A New Line of Attack on the Dichotomy Conjecture. Pages 725–734 of: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*.

Lauritzen, Steffen L. 1996. *Graphical Models*. Oxford University Press.

Lovász, László. 1983. Submodular Functions and Convexity. Pages 235–257 of: Bachem, A., Grötschel, M., and Korte, B. (eds), *Mathematical Programming – The State of the Art*. Berlin: Springer.

Marx, Dániel. 2010a. Approximating fractional hypertree width. *ACM Transactions on Algorithms*, **6**(2).

Marx, Dániel. 2010b. Can You Beat Treewidth? *Theory of Computing*, **6**(1), 85–112.

Marx, Dániel. 2010c. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. Pages 735–744 of: *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'10)*.

Marx, Dániel. 2011. Tractable Structures for Constraint Satisfaction with Truth Tables. *Theory of Computing Systems*, **48**(3), 444–464.

McCormick, S. Thomas, and Fujishige, Satoru. 2010. Strongly polynomial and fully combinatorial algorithms for bisubmodular function minimization. *Mathematical Programming*, **122**(1), 87–120.

Montanari, Ugo. 1974. Networks of Constraints: Fundamental properties and applications to picture processing. *Information Sciences*, **7**, 95–132.

Narayanan, H. 1997. *Submodular Functions and Electrical Networks*. Amsterdam: North-Holland.

Nemhauser, George L., and Wolsey, Laurence A. 1988. *Integer and Combinatorial Optimization*. John Wiley & Sons.

Nešetřil, Jaroslav, Siggers, Marh H., and Zádori, László. 2010. A combinatorial constraint satisfaction problem dichotomy classification conjecture. *European Journal of Combinatorics*, **31**(1), 280–296.

Orlin, James B. 2009. A Faster Strongly Polynomial Time Algorithm for Submodular Function Minimization. *Mathematical Programming*, **118**(2), 237–251.

Pearson, Justin K., and Jeavons, Peter G. 1997 (July). *A survey of tractable constraint satisfaction problems*. Tech. rept. CSD-TR-97-15. Royal Holloway, University of London.

Queyranne, Maurice. 1998. Minimising symmetric submodular functions. *Mathematical Programming*, **82**(1-2), 3–12.

Raghavendra, Prasad. 2008. Optimal algorithms and inapproximability results for every CSP? Pages 245–254 of: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC'08)*.

Rossi, Francesca, van Beek, Peter, and Walsh, Toby (eds). 2006. *The Handbook of Constraint Programming*. Elsevier.

Salamon, András Z., and Jeavons, Peter G. 2008. Perfect Constraints Are Tractable. Pages 524–528 of: *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming (CP'08)*. Lecture Notes in Computer Science, vol. 5202. Springer.

Samer, Marko, and Szeider, Stefan. 2010. Constraint satisfaction with bounded treewidth revisited. *Journal of Computer and System Sciences*, **76**(2), 103–114.

Schiex, Thomas, Fargier, Hélène, and Verfaillie, Gérard. 1995. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In: *Proceedings*

of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95).

Schrijver, Alexander. 2000. A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time. *Journal of Combinatorial Theory, Series B*, **80**(2), 346–355.

Schrijver, Alexander. 2003. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics, vol. 24. Springer.

Takhanov, Rustem. 2010a. A Dichotomy Theorem for the General Minimum Cost Homomorphism Problem. Pages 657–668 of: *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*.

Takhanov, Rustem. 2010b. Extensions of the Minimum Cost Homomorphism Problem. Pages 328–337 of: *Proceedings of the 16th International Computing and Combinatorics Conference (COCOON'10)*. Lecture Notes in Computer Science, vol. 6196. Springer.

Topkis, Donald M. 1998. *Supermodularity and Complementarity*. Princeton University Press.

Ullman, Jeffrey D. 1989. *Principles of Database and Knowledge-Base Systems*. Vol. 1 & 2. Computer Science Press.

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

Werner, Tomáš. 2007. A Linear Programming Approach to Max-Sum Problem: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **29**(7), 1165–1179.

Willard, Ross. 2010. Testing Expressibility Is Hard. Pages 9–23 of: *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP'10)*. Lecture Notes in Computer Science, vol. 6308. Springer.

Zanuttini, Bruno, and Živný, Stanislav. 2009. A note on some collapse results of valued constraints. *Information Processing Letters*, **109**(11), 534–538.

Živný, Stanislav. 2009. *The Complexity and Expressive Power of Valued Constraints*. Ph.D. thesis, University of Oxford.

Živný, Stanislav, and Jeavons, Peter G. 2010. Classes of Submodular Constraints Expressible by Graph Cuts. *Constraints*, **15**(3), 430–452.

Živný, Stanislav, Cohen, David A., and Jeavons, Peter G. 2009. The Expressive Power of Binary Submodular Functions. *Discrete Applied Mathematics*, **157**(15), 3347–3358.