The complexity of valued CSPs*

Andrei Krokhin[1] and Stanislav Živný[2]

1 **School of Engineering and Computing Sciences**
University of Durham, UK
andrei.krokhin@durham.ac.uk

2 **Department of Computer Science**
University of Oxford, UK
standa.zivny@cs.ox.ac.uk

Abstract

We survey recent results on the broad family of problems that can be cast as valued constraint satisfaction problems (VCSPs). We discuss general methods for analysing the complexity of such problems, give examples of tractable cases, and identify general features of the complexity landscape.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.1.6 Optimization.

Keywords and phrases Constraint satisfaction problems, Optimisation, Tractability.

Digital Object Identifier 10.4230/DFU.Vol7.15301.229

1 Introduction

Computational problems from many different areas involve finding an assignment of values to a set of variables, where that assignment must satisfy some specified feasibility conditions and optimise some specified objective function. In many such problems the objective function can be represented as a sum of functions, each of which depends on some subset of the variables. Examples include: Gibbs energy minimisation, Markov Random Fields (MRF), Conditional Random Fields (CRF), Min-Sum Problems, Minimum Cost Homomorphism, Constraint Optimisation Problems (COP) and Valued Constraint Satisfaction Problems (VCSP) [11, 30, 87, 110, 114, 116].

We focus in this article on a generic framework for such problems that captures their general form. Bringing all such problems into a common framework draws attention to common aspects that they all share, and allows a very general algebraic approach for analysing their complexity to be developed. The primary motivation for this line of research is to understand the general picture of complexity within this general framework, rather than to develop specialised techniques for specific applications.

We will give an overview of this algebraic approach, and the results that have been obtained by using it. We will focus on algorithms for solving problems to optimality and on the computational complexity of such problems. Although there is a survey from two years ago [61], the recent massive progress [43, 77, 78, 105, 72] justifies a new survey.

* Stanislav Živný was supported by a Royal Society University Research Fellowship. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

The Constraint Satisfaction Problem: Complexity and Approximability. *Dagstuhl Follow-Ups*, Volume 7, ISBN 978-3-95977-003-3.
Editors: Andrei Krokhin and Stanislav Živný; pp. 229–261

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

The generic framework we use is the *valued constraint satisfaction problem* (VCSP), defined formally as follows. Throughout the paper, let D be a fixed finite set and let $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ denote the set of rational numbers with (positive) infinity.

▶ **Definition 1.** We denote the set of all functions $\phi : D^m \to \overline{\mathbb{Q}}$ by $\mathbf{\Phi}_D^{(m)}$ and let $\mathbf{\Phi}_D = \bigcup_{m \geq 1} \mathbf{\Phi}_D^{(m)}$. We will often call the functions in $\mathbf{\Phi}_D$ *cost functions* over D.

Let $V = \{x_1, \ldots, x_n\}$ be a set of variables. A *valued constraint* over V is an expression of the form $\phi(\mathbf{x})$ where $\mathbf{x} \in V^m$ and $\phi \in \mathbf{\Phi}_D^{(m)}$. The number m is called the *arity* of the constraint, the function ϕ is called the *constraint function*, and the tuple $\mathbf{x}$ the *scope* of the constraint.

We will call the elements of D *labels* (for variables), and say that the cost functions in $\mathbf{\Phi}_D$ take *values*.

▶ **Definition 2.** An instance of the *valued constraint satisfaction problem* (VCSP) is specified by a finite set $V = \{x_1, \ldots, x_n\}$ of variables, a finite set D of labels, and an *objective function* Φ expressed as follows:

$$\Phi(x_1, \ldots, x_n) = \sum_{i=1}^{q} \phi_i(\mathbf{x}_i) \tag{1}$$

where each $\phi_i(\mathbf{x}_i)$, $1 \leq i \leq q$, is a valued constraint over V. Each constraint can appear multiple times in Φ. The goal is to find an *assignment* of labels to the variables (or *labelling*) that minimises Φ.

Note that the value of the function Φ for any assignment of labels to the variables in V is given by the sum of the values taken by the constraints; this value will sometimes be called the *cost* of the assignment. An infinite value for any constraint indicates an infeasible assignment.

If the constraint functions in some VCSP instance are finite-valued, i.e., take only finite values, then every assignment is feasible, and the problem is to identify an assignment with minimum possible cost (i.e., we need to deal only with the optimisation issue). On the other hand, if each constraint function in an instance takes only two values: one finite value (possibly specific to the constraint) and ∞, then all feasible assignments are equally good, and so the only question is whether any such assignment exists (i.e., we need to deal only with the feasibility issue). If we have neither of the above cases then we need to deal with both feasibility and optimisation. To emphasise where this can happen we sometimes call VCSPs general-valued.

In Section 2 we give examples to show that many standard combinatorial optimisation problems can be conveniently expressed in the VCSP framework. In Section 3 we define certain algebraic properties of the constraints that can be used to identify tractable cases. Section 4 describes the basics of a general algebraic theory for analysing the complexity of different forms of valued constraints. In Sections 5 and 6 we use this algebraic theory to identify several tractable and intractable cases. In Section 7 we discuss the oracle model for representing the objective function. Finally, Section 8 gives a brief summary and identifies some open problems.

2 Problems and frameworks captured by the VCSP

In this section we will give examples of specific problems and previously studied frameworks that can be expressed as VCSPs with restricted forms of constraints.

▶ **Definition 3.** Any set $\Gamma \subseteq \mathbf{\Phi}_D$ is called a *valued constraint language* over D, or simply a *language*. We will denote by VCSP(Γ) the class of all VCSP instances in which the constraint functions are all contained in Γ.

Valued constraint languages may be infinite, but it will be convenient to follow [18, 25] and define the complexity of a valued constraint language in terms of its finite subsets. We assume throughout that P $\neq$ NP.

▶ **Definition 4.** A valued constraint language Γ is called *tractable* if VCSP(Γ') can be solved (to optimality) in polynomial time for every finite subset $\Gamma' \subseteq \Gamma$, and Γ is called *NP-hard* if VCSP(Γ) is NP-hard for some finite $\Gamma' \subseteq \Gamma$.

One advantage of defining tractability in terms of finite subsets is that the tractability of a valued constraint language is independent of whether the cost functions are represented explicitly (say, via full tables of values, or via tables for the finite-valued parts) or implicitly (via oracles).

▶ **Example 5** (1-in-3-Sat)**.** Let $D = \{0, 1\}$ and let $\Gamma_{\mathsf{1\text{-}in\text{-}3}}$ be the language that contains just the single ternary cost function $\phi_{\mathsf{1\text{-}in\text{-}3}} : D^3 \to \overline{\mathbb{Q}}$ defined by

$$\phi_{\mathsf{1\text{-}in\text{-}3}}(x, y, z) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if exactly one of } x, y, z \text{ is } 1 \\ \infty & \text{otherwise} \end{cases}.$$

The problem VCSP($\Gamma_{\mathsf{1\text{-}in\text{-}3}}$) is exactly the 1-in-3-Sat problem. This problem is NP-hard [95, 44], so $\Gamma_{\mathsf{1\text{-}in\text{-}3}}$ is NP-hard.

▶ **Example 6** (NAE-Sat)**.** Let $D = \{0, 1\}$ and let Γ_{nae} be the language that contains just the single ternary cost function $\phi_{\mathsf{nae}} : D^3 \to \overline{\mathbb{Q}}$ defined by

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

The problem VCSP(Γ_{nae}) is exactly the Not-All-Equal-Sat problem, also known as the 3-Uniform Hypergraph 2-Colourability problem. This problem is NP-hard [44], so Γ_{nae} is NP-hard.

▶ **Example 7** (Max-k-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

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

The problem VCSP(Γ_{xor}) corresponds to the problem of minimising the number of monochromatic edges in a k-colouring (where $k = |D|$) of the graph G formed by the scopes of the constraints. This problem is known as the Maximum k-Cut problem (or simply Max-Cut when $|D| = 2$), and is NP-hard [44].

Hence, for any choice of D with $|D| \geq 2$, the language Γ_{xor} is NP-hard.

▶ **Example 8** (Potts model)**.** Let Γ_{Potts} be the language that contains all unary cost functions and the single binary cost function ϕ_{Potts}: $D^2 \to \overline{\mathbb{Q}}$ defined by

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

The problem VCSP(Γ_{Potts}) corresponds to finding the minimum energy state of the Potts model (with external field) from statistical mechanics [93]. This model is also used as the basis for a standard Markov Random Field approach to a wide variety of problems in machine vision [11]. For $|D| = 2$, the function ϕ_{Potts} is submodular (see Example 19) and we will show that this implies that Γ_{Potts} is tractable. For $|D| > 2$, Γ_{Potts} is NP-hard as it includes, as a special case, the multiway cut problem, which is NP-hard [34].

▶ **Example 9** ((s,t)-Min-Cut)**.** Let $G = (V, E)$ be a directed weighted graph such that for every $(u, v) \in E$ there is a weight $w(u, v) \in \mathbb{Q}_{\geq 0}$ and let $s, t \in V$ be distinguished 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 label $d \in D$ and cost $c \in \overline{\mathbb{Q}}$, we define

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

For any weight $w \in \mathbb{Q}_{\geq 0}$, we define

$$\phi_{\mathsf{cut}}^w(x, y) \stackrel{\text{def}}{=} \begin{cases} w & \text{if } x = 0 \text{ and } y = 1 \\ 0 & \text{otherwise} \end{cases}.$$

We denote by Γ_{cut} the set $\{\eta_0^\infty, \eta_1^\infty\} \cup \{\phi_{\mathsf{cut}}^w \mid w \in \mathbb{Q}_{\geq 0}\}$. A minimum-weight (s,t)-cut in a graph G with set of nodes $V = \{x_1, \ldots, x_n\}$ corresponds to the set of variables assigned the label 0 in a minimal cost assignment to the VCSP instance defined by

$$\Phi(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \eta_0^\infty(s) + \eta_1^\infty(t) + \sum_{(x_i,x_j)\in E} \phi_{\mathsf{cut}}^{w(x_i,x_j)}(x_i, x_j).$$

The unary constraints ensure that the source and target nodes must be assigned the labels 0 and 1, respectively, in any minimal cost assignment.

Furthermore, it is an easy exercise to show that any instance of VCSP(Γ_{cut}) on n variables can be solved in $O(n^3)$ time by a reduction to (s,t)-Min-Cut and then using the standard algorithm [45]. Hence Γ_{cut} is tractable.

▶ **Example 10** (Minimum Vertex Cover)**.** The Minimum Vertex Cover problem asks for a minimum size set W of vertices in a given graph $G = (V, E)$ such that each edge in E has at least one endpoint in W. This problem is NP-hard [44].

Let $D = \{0, 1\}$. We define

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

We denote by Γ_{vc} the language $\{\phi_{\mathsf{vc}}, \eta_0^1\}$, where η_0^1 is the function defined in Example 9 that imposes unit cost for any variable assigned the label 1. A minimum vertex cover in a graph G with set of vertices $V = \{x_1, \ldots, x_n\}$ corresponds to the set of vertices assigned the label 1 in some minimum cost assignment to the VCSP(Γ_{vc}) instance defined by

$$\Phi(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \sum_{x_i\in V} \eta_0^1(x_i) + \sum_{(x_i,x_j)\in E} \phi_{\mathsf{vc}}(x_i, x_j).$$

The binary constraints ensure that in any minimal cost assignment at least one endpoint of each edge belongs to the vertex cover.

Furthermore, it is easy to convert any instance of VCSP(Γ_{vc}) to an equivalent instance of Minimum Vertex Cover by repeatedly assigning the label 1 to all variables which do not appear in the scope of any unary constraints and removing these variables and all constraints involving them. Hence Γ_{vc} is NP-hard.

We will now show how several broad frameworks previously studied in the literature can be expressed as special cases of the VCSP with restricted languages. We will discuss algorithms and complexity classifications for them in Sections 5 and 6.

▶ **Example 11** (CSP)**.** The standard constraint satisfaction problem (CSP) over any fixed set of possible labels D can be seen as the special case of the VCSP where all cost functions take only the values 0 or ∞, representing allowed (satisfying) and disallowed tuples, respectively. Such constraints and cost functions are sometimes called *crisp*. In other words, the CSP can be seen as VCSP(Γ_{crisp}), where Γ_{crisp} is the language consisting of all cost functions on some fixed set D with range $\{0, \infty\}$. Note that the CSP can also be cast as the homomorphism problem for relational structures [37] (cf. Example 12).

Since the CSP includes many known NP-hard problems, such as 1-in-3-Sat (Example 5) and Graph-3-Colouring, the language Γ_{crisp} is clearly NP-hard. However, many tractable subsets of Γ_{crisp} have been identified [95, 62, 37, 18, 14, 19, 57, 6], mostly through an algebraic approach whose extension we discuss in Section 4. There are many surveys on the complexity of the CSP, see the books [33, 32, 80], and also [22, 51, 2].

Feder and Vardi conjectured that the CSP exhibits a *dichotomy*: that is, every finite language $\Gamma \subseteq \Gamma_{\mathsf{crisp}}$ is either tractable or NP-hard [37], thus excluding problems of intermediate complexity, as given by Ladner's Theorem (assuming P≠NP) [83]. The *Algebraic Dichotomy* conjecture, which we state formally and discuss in Section 6, specifies the precise boundary between tractable and NP-hard crisp languages [18].

▶ **Example 12** (Digraph Homomorphism)**.** Given two digraphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$, a mapping $f : V(G) \to V(H)$ is a *homomorphism* from G to H if f preserves edges, that is, $(u, v) \in E(G)$ implies $(f(u), f(v)) \in E(H)$.

The problem whether an input digraph G admits a homomorphism to a fixed digraph H is also known as the H-Colouring problem and has been actively studied in graph theory [50, 51], see also [84].

For any digraph H, let $D = V(H)$ and let Γ_H be the language that contains just the single binary cost function $\phi_H : D^2 \to \overline{\mathbb{Q}}$ defined by

$$\phi_H(x, y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } (x, y) \in E(H) \\ \infty & \text{otherwise} \end{cases} .$$

For any digraph H, the problem VCSP(Γ_H), which is a special case of the CSP (Example 11), corresponds to the H-colouring problem, where the input digraph G is given by the scopes of the constraints. If we add all unary crisp functions to Γ_H then the resulting VCSP is known as List H-Colouring [50, 51].

It is known that both the Feder-Vardi conjecture and the Algebraic Dichotomy conjecture are equivalent to their restrictions to the H-colouring problem [20, 37].

▶ **Example 13** (Max-CSP)**.** An instance of the (weighted) maximum constraint satisfaction problem (Max-CSP) is an instance of the CSP where the goal is to maximise the (weighted) number of satisfied constraints.

When seeking the optimal solution, maximising the number of satisfied constraints is the same as minimising the number of unsatisfied constraints. Hence for any instance Φ of the Max-CSP, we can define a corresponding VCSP instance Φ' in which each constraint c of Φ is associated with a constraint over the same scope in Φ' which assigns cost 0 to tuples allowed by c, and cost 1 to tuples disallowed by c. It follows that Max-CSP is equivalent to VCSP(Γ_{Max}), where Γ_{Max} is the language consisting of cost functions whose values are restricted to zero and one.

For $D = \{0, 1\}$, the complexity of all subsets of Γ_{Max} has been completely classified in [70]. Initial results for languages over arbitrary finite sets appeared in [21].

▶ **Example 14** (Min-Cost-Hom)**.** Here and in the following examples let Γ_{unary} consist of all unary cost functions and let $\Gamma_{\mathsf{mc}} = \Gamma_{\mathsf{crisp}} \cup \Gamma_{\mathsf{unary}}$ (where Γ_{crisp} is defined in Example 11). Problems of the form VCSP(Γ) with $\Gamma \subseteq \Gamma_{\mathsf{mc}}$ have been studied under the name of the Minimum-Cost Homomorphism problem (or Min-Cost-Hom) [49, 52, 100, 101, 110, 111]. Note that the first three of these papers assume that $\Gamma_{\mathsf{unary}} \subseteq \Gamma$, while the last three do not. In [49, 52] Γ is assumed to be of the form $\{\phi_H\} \cup \Gamma_{\mathsf{unary}}$, where ϕ_H is a binary crisp cost function, as in Example 12.

In any instance of VCSP(Γ_{mc}), the crisp constraints specify the CSP part, i.e., the feasibility aspect of the problem, while the unary constraints specify the optimisation aspect. More precisely, the unary constraints specify the costs of assigning labels to individual variables. Complexity classifications for special cases of Min-Cost-Hom will be discussed in Section 6.

▶ **Example 15** (Min-Ones)**.** An instance of the Boolean Minimum Ones (Min-Ones) problem is an instance of the CSP over $D = \{0, 1\}$ where the goal is to satisfy all constraints and minimise the number of variables assigned the label 1. Such instances correspond to Min-Cost-Hom instances over $\{0, 1\}$ in which all unary constraints are of the form η_0^1 as defined in Example 9 (which impose a unit cost for any variables assigned the label 1). A classification of the complexity of all subsets of this language was obtained in [70, 33].

▶ **Example 16** (Min-Sol)**.** The Minimum Solution problem (Min-Sol) [67, 65] is a generalisation of Min-Ones from Example 15 to D being a larger set of non-negative integers, where the only allowed unary cost function is a particular finite-valued injective function, namely $u(x) = wx$ for some positive $w \in \mathbb{Q}$. Thus, this problem is also a subproblem of Min-Cost-Hom. A complexity classification for Min-Sol problems will be discussed in Section 6.

3 Polymorphisms and fractional polymorphisms

To develop general tools to classify the complexity of different valued constraint languages, we will now define certain algebraic properties of cost functions.

A function $f : D^k \to D$ is called a k-ary *operation* on D. The k-ary *projections*, defined for all $1 \leq i \leq k$, are the operations $\mathrm{e}_i^{(k)}$ such that $\mathrm{e}_i^{(k)}(x_1, \ldots, x_k) = x_i$. For any tuples $\mathbf{x_1}, \ldots, \mathbf{x_k} \in D^m$, we denote by $f(\mathbf{x_1}, \ldots, \mathbf{x_k})$ the tuple in D^m obtained by applying f to $\mathbf{x_1}, \ldots, \mathbf{x_k}$ componentwise.

For a cost function $\phi : D^m \to \overline{\mathbb{Q}}$, we denote by $\mathrm{Feas}(\phi) = \{\mathbf{x} \in D^m \mid \phi(\mathbf{x}) \text{ is finite}\}$ the *feasibility relation* of ϕ. We will view $\mathrm{Feas}(\phi)$ both as a relation and as a $\{0, \infty\}$-valued cost function. Recall from Example 11 that $\{0, \infty\}$-valued cost functions are called *crisp*.

Any valued constraint language Γ defined on D can be associated with a set of operations on D, known as the polymorphisms of Γ, and defined as follows.

▶ **Definition 17** (Polymorphism). Let $\phi : D^m \to \overline{\mathbb{Q}}$ be a cost function. We say that a k-ary operation $f : D^k \to D$ is a *polymorphism* of ϕ if, for any $\mathbf{x_1}, \ldots, \mathbf{x_k} \in \mathrm{Feas}(\phi)$ we have that $f(\mathbf{x_1}, \ldots, \mathbf{x_k}) \in \mathrm{Feas}(\phi)$.

For any valued constraint language Γ over a set D, we denote by $\mathrm{Pol}(\Gamma)$ the set of all operations on D which are polymorphisms of all $\phi \in \Gamma$. We denote by $\mathrm{Pol}^{(k)}(\Gamma)$ the k-ary operations in $\mathrm{Pol}(\Gamma)$.

Note that trivially the projections are polymorphisms of all valued constraint languages.

For $\{0, \infty\}$-valued cost functions (relations) this notion of polymorphism corresponds precisely to the standard notion of polymorphism for relations [10, 62]. This notion of polymorphism has played a key role in the analysis of complexity for the CSP [62, 18]. However, for the analysis of the VCSP we need a more flexible notion that assigns weights to polymorphisms [23].

▶ **Definition 18** (Fractional Polymorphism). Let $\phi : D^m \to \overline{\mathbb{Q}}$ be a cost function. A function $\omega : \mathrm{Pol}^{(k)}(\phi) \to \mathbb{Q}_{\geq 0}$ is called a k-ary *fractional polymorphism* of ϕ if it satisfies the following conditions:

- $\sum_{f \in \mathrm{Pol}^{(k)}(\phi)} \omega(f) = 1$;
- for any $\mathbf{x_1}, \ldots, \mathbf{x_k} \in \mathrm{Feas}(\phi)$

$$\sum_{f \in \mathrm{Pol}^{(k)}(\phi)} \omega(f)\phi(f(\mathbf{x_1}, \ldots, \mathbf{x_k})) \;\leq\; \frac{1}{k}\sum_{i=1}^{k} \phi(\mathbf{x_i})\,. \tag{2}$$

We define $\mathrm{supp}(\omega) = \{f \mid \omega(f) > 0)$ to be the *support* of ω.

▶ Remark. The definition of a fractional polymorphism can be re-stated in probabilistic terms, as follows. Any fractional polymorphism ω can be seen as a probability distribution over $\mathrm{Pol}^{(k)}(\phi)$. We can then re-write Inequality (2) as follows:

$$\mathbb{E}_{f \sim \omega}[\phi(f(\mathbf{x_1}, \ldots, \mathbf{x_k}))] \;\leq\; \mathrm{avg}\{\phi(\mathbf{x_1}), \ldots, \phi(\mathbf{x_k})\}. \tag{3}$$

This is illustrated in Figure 1, which should be read from left to right. Let ω be a probability distribution on $\mathrm{Pol}^{(k)}(\phi)$ and let $\mathrm{supp}(\omega) = \{f_1, \ldots, f_n\}$. Starting with the m-tuples $\mathbf{x_1}, \ldots, \mathbf{x_k}$, we first apply operations $f_1, \ldots, f_n$ to these tuples componentwise, thus obtaining the m-tuples $\mathbf{x'_1}, \ldots, \mathbf{x'_n}$. Inequality 3 amounts to comparing the average of the values of ϕ applied to the tuples $\mathbf{x_1}, \ldots, \mathbf{x_k}$ with the weighted sum of the values of ϕ applied to the tuples $\mathbf{x'_1}, \ldots, \mathbf{x'_n}$, which is the expected value of $\phi(f(\mathbf{x_1}, \ldots, \mathbf{x_k}))$ as f is drawn from ω.

Examples of fractional polymorphisms include *fractional projections*: a k-ary fractional projection ω is defined by $\omega(\mathrm{e}_1^{(k)}) = \ldots = \omega(\mathrm{e}_k^{(k)}) = \frac{1}{k}$. In this case, Inequality 2 holds trivially with equality. Hence, fractional projections are trivially fractional polymorphisms of all valued constraint languages.

If ω is a fractional polymorphism of ϕ, then we say that ϕ *admits* ω as a fractional polymorphism. We say that a language Γ admits a fractional polymorphism ω if ω is a fractional polymorphism of every cost function $\phi \in \Gamma$. It is easy to see that in this case ω is also a fractional polymorphism of any function Φ arising as an instance of $\mathrm{VCSP}(\Gamma)$. The intuition behind the notion of fractional polymorphism is that it allows one to combine several feasible assignments for an instance of $\mathrm{VCSP}(\Gamma)$, in a randomised way, into a new feasible assignment so that the expected value of the new assignment (non-strictly) improves the average value of the original assignments.

$$\begin{array}{lllll} \mathbf{x_1} & \mathbf{x_1}[1] & \mathbf{x_1}[2] & \dots & \mathbf{x_1}[m] \\ \mathbf{x_2} & \mathbf{x_2}[1] & \mathbf{x_2}[2] & \dots & \mathbf{x_2}[m] \\ \vdots & & & \vdots & \\ \mathbf{x_k} & \mathbf{x_k}[1] & \mathbf{x_k}[2] & \dots & \mathbf{x_k}[m] \end{array} \quad \xrightarrow{\phi} \quad \left.\begin{array}{l} \phi(\mathbf{x_1}) \\ \phi(\mathbf{x_2}) \\ \vdots \\ \phi(\mathbf{x_k}) \end{array}\right\} \frac{1}{k}\sum_{i=1}^{k}\phi(\mathbf{x_i})$$

$$\begin{array}{lllll} \mathbf{x'_1} = f_1(\mathbf{x_1},\dots,\mathbf{x_k}) & \mathbf{x'_1}[1] & \mathbf{x'_1}[2] & \dots & \mathbf{x'_1}[m] \\ \mathbf{x'_2} = f_2(\mathbf{x_1},\dots,\mathbf{x_k}) & \mathbf{x'_2}[1] & \mathbf{x'_2}[2] & \dots & \mathbf{x'_2}[m] \\ \vdots & & & \vdots & \\ \mathbf{x'_n} = f_n(\mathbf{x_1},\dots,\mathbf{x_k}) & \mathbf{x'_n}[1] & \mathbf{x'_n}[2] & \dots & \mathbf{x'_n}[m] \end{array} \quad \xrightarrow{\phi} \quad \left.\begin{array}{l} \phi(\mathbf{x'_1}) \\ \phi(\mathbf{x'_2}) \\ \vdots \\ \phi(\mathbf{x'_n}) \end{array}\right\} \sum_{i=1}^{n}\Pr_{\omega}[f_i]\phi(\mathbf{x'_i})$$

Figure 1 Probabilistic definition of a fractional polymorphism.

Note that if Γ consists of crisp functions then the k-ary fractional polymorphisms of Γ are all possible probability distributions on $\mathrm{Pol}^{(k)}(\Gamma)$.

A more restricted form of fractional polymorphism was introduced earlier in [25] and is known as a *multimorphism*. This is essentially a k-ary fractional polymorphism where the value of $\omega(f)$ is of the form ℓ/k, where $\ell \in \mathbb{N}$, for every $f \in \mathrm{supp}(\omega)$.

One can specify a k-ary multimorphism as a k-tuple $\mathbf{f} = \langle f_1, \dots, f_k \rangle$ of k-ary operations f_i on D, where each operation f with $\omega(f) = \ell/k$ for some $\ell > 0$ appears ℓ times, and then the definition simplifies as follows: for all $\mathbf{x_1}, \dots, \mathbf{x_k} \in \mathrm{Feas}(\phi)$,

$$\sum_{i=1}^{k} \phi(f_i(\mathbf{x_1}, \dots, \mathbf{x_k})) \;\le\; \sum_{i=1}^{k} \phi(\mathbf{x_i})\,. \tag{4}$$

Fractional polymorphisms (including the special cases of multimorphisms) have proved to be a valuable tool for identifying tractable valued constraint languages, as we will illustrate in this section. Closely related algebraic objects, known as *weighted polymorphisms*, will be discussed in Section 4.

▶ **Example 19** (Submodularity)**.** For any finite set V, a rational-valued function h defined on subsets of V is called a *set function*. A set function h is called *submodular* if for all subsets S and T of V,

$$h(S \cap T) + h(S \cup T) \;\le\; h(S) + h(T). \tag{5}$$

Submodular functions are a key concept in operational research and combinatorial optimisation (see, e.g. [38, 96, 109] for extensive information about them). They are often considered to be a discrete analogue of convex functions. Examples of submodular functions include cuts in graphs, matroid rank functions, and entropy functions. There are combinatorial algorithms for minimising submodular functions in polynomial time (see [96, 38], and also [59]).

If we set $D = \{0, 1\}$, then any set function h on V can be associated with a ($|V|$-ary) cost function ϕ defined on the characteristic vectors of subsets of V. The intersection and union operations on subsets correspond to the Min and Max operations on the associated characteristic vectors. Hence h is submodular if and only if the associated cost function ϕ satisfies the following inequality:

$$\phi(\mathrm{Min}(\mathbf{x}_1, \mathbf{x}_2)) + \phi(\mathrm{Max}(\mathbf{x}_1, \mathbf{x}_2)) \;\le\; \phi(\mathbf{x}_1) + \phi(\mathbf{x}_2)\,.$$

But this means that ϕ admits the 2-ary fractional polymorphism ω_{sub}, defined by $\omega_{sub}(\text{Min}) = \omega_{sub}(\text{Max}) = \frac{1}{2}$. This is equivalent to saying that ϕ admits $\langle \text{Min}, \text{Max} \rangle$ as a multimorphism.

▶ **Example 20** (Generalised Submodularity)**.** Let D be a finite *lattice*, i.e., a partially ordered set, where each pair of elements $\{a, b\}$ has a least upper bound, $\vee(a, b)$, and a greatest lower bound, $\wedge(a, b)$. We denote by Γ_{sub} the set of all cost functions over D that admit $\langle \vee, \wedge \rangle$ as a multimorphism. Using a polynomial-time strongly combinatorial algorithm for minimising submodular functions, it was shown in [25] that Γ_{sub} is tractable when D is a totally ordered lattice (i.e., a *chain*). More general lattices will be discussed in Section 5 and Section 7.

▶ **Example 21** (Max)**.** We denote by Γ_{max} the set of all cost functions (over some fixed finite totally ordered set D) that admit $\langle \text{Max}, \text{Max} \rangle$ as a multimorphism, where $\text{Max} : D^2 \to D$ is the binary operation returning the larger of its two arguments. Note that Γ_{max} includes all monotonic decreasing finite-valued cost functions, as well as some non-monotonic crisp cost functions [25]. It was shown in [25] that Γ_{max} is tractable.

▶ **Example 22** (Min)**.** We denote by Γ_{min} the set of all cost functions (over some fixed finite totally ordered set D) that admit $\langle \text{Min}, \text{Min} \rangle$ as a multimorphism, where $\text{Min} : D^2 \to D$ is the binary operation returning the smaller of its two arguments. The tractability of Γ_{min} was established in [25].

▶ **Example 23** (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 5 (see [38, 92] for the precise definition). Examples of bisubmodular functions include rank functions of delta-matroids [38].

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:

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

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

We denote by Γ_{bis} the set of finite-valued cost functions that admit $\langle \text{Min}_0, \text{Max}_0 \rangle$ as a multimorphism. The language Γ_{bis} can be shown to be tractable using the results of [92] (see also [38]).

The definitions of Min_0 and Max_0 still make sense when $D = \{0, 1, 2 \ldots, k\}$, $k \geq 3$. In that case, functions on D that admit $\langle \text{Min}_0, \text{Max}_0 \rangle$ as a multimorphism are called *k-submodular*; they were introduced in [54]. The tractability of k-submodular general-valued constraint languages was shown in [73] and will be discussed in Section 5.

▶ **Example 24** (Skew Bisubmodularity)**.** Let $D = \{0, 1, 2\}$. Recall the definition of operations Min_0 and Max_0 from Example 23. We define

$$\text{Max}_1(x, y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } 0 \neq x \neq y \neq 0 \\ \text{Max}(x, y) & \text{otherwise} \end{cases}.$$

A function ϕ: $D^m \to \overline{\mathbb{Q}}$ is called *α-bisubmodular* [55], for some real $0 < \alpha \leq 1$, if ϕ admits the fractional polymorphism ω defined by $\omega(\text{Min}_0) = \frac{1}{2}$, $\omega(\text{Max}_0) = \frac{\alpha}{2}$, $\omega(\text{Max}_1) = \frac{1-\alpha}{2}$. Note that 1-bisubmodular functions are (ordinary) bisubmodular functions as defined in Example 23. It is shown in [55] that each distinct value of α is associated with a distinct class of α-bisubmodular functions. The tractability of α-bisubmodular valued constraint languages was shown in [73] and will be discussed in Section 5.

▶ **Example 25** ((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 as a multimorphism. It was shown in [24], by a reduction to the minimisation problem for submodular functions (cf. Example 20), that any such Γ is tractable. A different proof of tractability of Γ will be discussed in Section 5. It is shown in [73] that any finite-valued language that admits a symmetric tournament pair multimorphism also admits the submodularity multimorphism with respect to some totally ordered lattice on D (cf. Example 20).

Now let Γ be an arbitrary language that admits any tournament pair as a multimorphism. It was shown in [24], by a reduction to the symmetric tournament pair case, that any such Γ is also tractable. Again, the tractability of Γ will be discussed in more detail in Section 5.

▶ **Example 26** (Tournament in the support)**.** Let Γ be an arbitrary language that admits a binary fractional polymorphism ω such that $\mathrm{supp}(\omega)$ contains a tournament operation, as defined in Example 25. The tractability of Γ was shown in [105], thus generalising Example 25, and will be discussed in Section 5.

▶ **Example 27** (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 was shown in [66], and was used in the classification of the Max-CSP over a four-element set (see Section 5).

▶ **Example 28** (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 $\mathbf{f} = \langle f_1, f_2, f_3\rangle$ be a triple of ternary operations such that f_1, f_2 and f_3 are all majority operations. Let $\phi : D^m \to \overline{\mathbb{Q}}$ be an m-ary cost function that admits $\mathbf{f}$ as a multimorphism. By Inequality (4), 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 $c \in \overline{\mathbb{Q}}$.

Let Γ_{Mjty} be the set of all cost functions that admit as a multimorphism some triple $\mathbf{f} = \langle f_1, f_2, f_3\rangle$ of arbitrary ternary majority operations. The tractability of Γ_{Mjty} was shown in [25].

▶ **Example 29** (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 that admit as a multimorphism some triple $\mathbf{f} = \langle f_1, f_2, f_3\rangle$ of arbitrary ternary minority operations. A similar argument to the one in Example 28 shows that the cost functions in Γ_{Mnty} have range $\{c, \infty\}$, for some finite $c \in \overline{\mathbb{Q}}$. The tractability of Γ_{Mnty} was shown in [25].

▶ **Example 30** (MJN)**.** Let $\mathbf{f} = \langle f_1, f_2, f_3\rangle$ be three ternary operations such that f_1 and f_2 are majority operations, and f_3 is a minority operation. Let Γ_{MJN} be the set of cost functions that admit $\mathbf{f}$ as a multimorphism. The tractability of Γ_{MJN} was shown in [74], generalising an earlier tractability result for a specific $\mathbf{f}$ of this form from [25].

▶ **Example 31** (Majority in the support)**.** Let Γ be an arbitrary language admitting a ternary fractional polymorphism ω such that $\mathrm{supp}(\omega)$ contains a majority operation. The tractability of Γ was shown in [105], thus generalising Examples 28 and 30, and will be discussed in Section 5.

Other tractable valued constraint languages defined by fractional polymorphisms include the so-called $L^\#$-convex languages [38], as well as the weakly and strongly tree-submodular languages defined in [71]. Hirai [53] recently introduced a framework of submodular functions on modular semilattices (defined by a type of fractional polymorphism) that generalises many examples given above, including standard submodularity, k-submodularity, skew bisubmodularity, and tree submodularity. See [53] for the natural (but somewhat technical) definition of this very general framework.

4 A general algebraic theory of complexity

We have seen in the previous section that many tractable cases of the VCSP can be defined by having a particular fractional polymorphism. The algebraic theory developed in [23, 77, 78, 43] establishes that, in fact, every tractable valued constraint language can be exactly characterised by an associated set of algebraic objects known as weighted polymorphisms, which are different but equivalent to fractional polymorphisms. This extends (parts of) the algebraic theory previously developed for the CSP [13, 18, 62] that has led to significant advances in understanding the landscape of complexity for the CSP over the last 10 years (e.g., [1, 6, 14, 15, 17, 19, 57]). In this section, we will give a brief overview of the main results of this algebraic theory for the VCSP. We refer the reader to [23, 77, 78, 43] for full details and proofs.

First we consider the effect of extending a valued constraint language $\Gamma \subseteq \mathbf{\Phi}_D$ to a possibly larger valued constraint language. We first 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 [18, 62, 25, 116].

▶ **Definition 32.** We say that an m-ary cost function ϕ is *expressible* over a constraint language Γ if there exists an instance $\Phi \in \mathrm{VCSP}(\Gamma)$ with variables $V = \{x_1, \dots, x_n, y_1, \dots, y_m\}$, such that

$$\phi(y_1, \dots, y_m) = \min_{x_1,\dots,x_n} \Phi(x_1, \dots, x_n, y_1, \dots, y_m)\,.$$

For a cost function ϕ, we denote by $\mathrm{Opt}(\phi) = \{\mathbf{x} \in \mathrm{Feas}(\phi) \mid \forall \mathbf{y} : \phi(\mathbf{x}) \le \phi(\mathbf{y})\}$ the *optimality* relation, which contains the tuples on which ϕ is *minimised*. Similarly to $\mathrm{Feas}(\phi)$, we will view $\mathrm{Opt}(\phi)$ both as a relation and as a crisp cost function.

▶ **Definition 33.** A valued constraint language $\Gamma \subseteq \mathbf{\Phi}_D$ is called a *weighted relational clone* if it contains the binary equality relation and the unary empty relation; is closed under expressibility, scaling by non-negative rational constants, addition of rational constants, and operators Feas and Opt. We define $\mathrm{wRelClone}(\Gamma)$ to be the smallest weighted relational clone containing Γ.

▶ **Example 34.** Let $D = \{0, 1\}$ and let u_1 be the unary crisp cost function defined by $u_1(0) = \infty$ and $u_1(1) = 0$. We will show that $\phi_{\mathsf{1\text{-}in\text{-}3}} \in \mathrm{wRelClone}(\{\phi_{\mathsf{xor}}, u_1\})$, where $\phi_{\mathsf{1\text{-}in\text{-}3}}$ is from Example 5 and ϕ_{xor} is from Example 7.

First observe that $\phi(x, y, z) = \phi_{\mathsf{xor}}(x, y) + \phi_{\mathsf{xor}}(x, z) + \phi_{\mathsf{xor}}(y, z)$ satisfies $\phi(0, 0, 0) = \phi(1, 1, 1) = 3$ and $\phi(x, y, z) = 1$ otherwise. Next observe that $\phi'(x, y, z) = \min_{w \in D}(\phi(x, y, z) +$

$u_1(w)+\phi_{\mathsf{xor}}(x,w)+\phi_{\mathsf{xor}}(y,w)+\phi_{\mathsf{xor}}(z,w))$ satisfies $\phi'(0,0,0)=3$, $\phi'(1,1,1)=6$, $\phi'(x,y,z)=2$ if there is exactly one 1 among $\{x,y,z\}$ and $\phi'(x,y,z)=3$ otherwise. Thus, $\phi_{\mathsf{1\text{-}in\text{-}3}}=\mathrm{Opt}(\phi')$ and we have $\phi_{\mathsf{1\text{-}in\text{-}3}}\in \mathrm{wRelClone}(\{\phi_{\mathsf{xor}},u_1\})$.

▶ **Theorem 35** ([23, 43]). *A valued constraint language* Γ *is tractable if* $\mathrm{wRelClone}(\Gamma)$ *is tractable and NP-hard if* $\mathrm{wRelClone}(\Gamma)$ *is NP-hard.*

▶ Remark. For weighted relational clones that are not finitely generated, the rational values are replaced by real values in Definition 1, Definition 33 requires topological closedness, and Theorem 35 holds only up to an arbitrary additive error; we refer the reader to [43] for more details.

We now develop tools that will allow an alternative characterisation of any weighted relational clone. We first recall some basic terminology from universal algebra [10, 99]. 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 $\mathbf{O}_D$. Let $f\in\mathbf{O}_D^{(k)}$ and $g_1,\ldots,g_k\in\mathbf{O}_D^{(\ell)}$. The *superposition* of f and $g_1,\ldots,g_k$ is the ℓ-ary operation $f[g_1,\ldots,g_k]$ such that $f[g_1,\ldots,g_k](x_1,\ldots,x_\ell)=f(g_1(x_1,\ldots,x_\ell),\ldots,g_k(x_1\ldots,x_\ell))$.

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. It is easy to verify that the set of operations $\mathrm{Pol}(\Gamma)$ is a clone. Clones are actively studied in universal algebra; for example, all (countably many) clones on $D=\{0,1\}$ are known, but the situation is known to be much more complicated for larger sets D (see, e.g., [10, 99]). We denote by $\mathbf{J}_D$ the clone of all projections on D.

▶ **Definition 36.** A k-ary *weighting* is a function $\omega:\mathbf{O}_D^{(k)}\to\mathbb{Q}$ such that $\omega(f)<0$ only if f is a projection and $\sum_{f\in\mathbf{O}_D^{(k)}}\omega(f)=0$.

We denote by $\mathbf{W}_D$ the set of all possible weightings on D and by $\mathbf{W}_D^{(k)}$ the set of k-ary weightings in $\mathbf{W}_D$.

Since a weighting is simply a rational-valued function satisfying certain linear inequalities it can be scaled by any non-negative rational to obtain a new weighting. Similarly, any two weightings of the same arity can be added to obtain a new weighting.

The notion of superposition can also be extended to weightings in a natural way, by forming a superposition with each argument of the weighting, as follows.

▶ **Definition 37.** For any $\omega\in\mathbf{W}_D^{(k)}$ and any $g_1,\ldots,g_k\in\mathbf{O}_D^{(\ell)}$, we define the *superposition* of ω and $g_1,\ldots,g_k$, to be the function $\omega[g_1,\ldots,g_k]:\mathbf{O}_D^{(\ell)}\to\mathbb{Q}$ defined by

$$\omega[g_1,\ldots,g_k](f') \stackrel{\text{def}}{=} \sum_{\substack{f\in\mathbf{O}_D^{(k)}\\ f[g_1,\ldots,g_k]=f'}} \omega(f)\,. \tag{6}$$

It follows immediately from the definition of superposition that the sum of the weights in any superposition $\omega[g_1,\ldots,g_k]$ is equal to the sum of the weights in ω, which is zero, by Definition 36. However, it is not always the case that an arbitrary superposition satisfies the other condition in Definition 36, that negative weights are only assigned to projections. Hence we make the following definition:

▶ **Definition 38.** If the result of a superposition is a valid weighting, then that superposition will be called a *proper* superposition.

For a weighting $\omega\in\mathbf{W}_D^{(k)}$, we denote $\mathrm{supp}(\omega)=\{f\in\mathbf{O}_D^{(k)}\mid\omega(f)>0\}$.

▶ **Definition 39.** Let W be a non-empty set of weightings on a fixed domain D. We define $\mathrm{supp}(W) = \mathbf{J}_D \cup \bigcup_{\omega \in W} \mathrm{supp}(\omega)$.

We call W a *weighted clone* if it is closed under non-negative scaling, addition of weightings of equal arity, and proper superposition with operations from $\mathrm{supp}(W)$.

For any weighted clone W, $\mathrm{supp}(W)$ is a clone, which we call the *support clone* of W. This easy but important fact has been observed in [77, 78, 105, 43]. We remark that the inclusion of the Opt operator in the definition of weighted relational clones [43] ensures that there is no need for distinguishing the support clone from the positive support clone of weighted clones [77, 78].

▶ **Example 40.** For any clone, C, the set $\mathbf{W}_C$ containing all possible weightings of C is a weighted clone with support clone C.

▶ **Example 41.** For any clone, C, the set $\mathbf{W}^0_C$ containing all *zero-valued* weightings of C is a weighted clone with support clone C. $\mathbf{W}^0_C$ contains exactly one weighting of each possible arity, which assigns the value 0 to all operations in C of that arity.

▶ Remark. For weighted clones that are not finitely generated, the rational weights are replaced by real weights in Definition 36 and Definition 39 requires topological closedness; again, we refer the reader to [43] for more details.

We link weightings and cost functions by the concept of weighted polymorphisms.

▶ **Definition 42** (Weighted Polymorphism)**.** Let $\phi : D^m \to \overline{\mathbb{Q}}$ be a cost function and let ω be a k-ary weighting on D. We call ω a *weighted polymorphism* of ϕ if $\mathrm{supp}(\omega) \subseteq \mathrm{Pol}(\phi)$ and for any $\mathbf{x_1}, \ldots, \mathbf{x_k} \in \mathrm{Feas}(\phi)$

$$\sum_{f \in \mathrm{supp}(\omega)} \omega(f)\phi(f(\mathbf{x_1}, \ldots, \mathbf{x_k})) \;\leq\; 0\,. \tag{7}$$

For a set $W \subseteq \mathbf{W}_D$ which may contain weightings with different support clones over D, we can extend each of these weightings with zeros, as necessary, so that they are weightings of the same support clone C, where C is the smallest clone containing all the clones that are support clones of weightings in W. For any set $W \subseteq \mathbf{W}_D$, we define $\mathrm{wClone}(W)$ to be the smallest weighted clone containing this set of extended weightings obtained from W.

▶ **Definition 43.** For any $W \subseteq \mathbf{W}_D$, we denote by $\mathrm{Imp}(W)$ the set of all cost functions in $\mathbf{\Phi}_D$ which admit all weightings $\omega \in W$ as weighted polymorphisms. [1]

It follows immediately from the definition of a Galois connection [10] that, for any set D, the mappings wPol and Imp form a Galois connection between $\mathbf{W}_D$ and $\mathbf{\Phi}_D$, as illustrated in Figure 2. A characterisation of this Galois connection for finite sets D is given by the following theorem from [23, 43]:

▶ **Theorem 44** (Galois Connection for Valued Constraint Languages [23, 43])**.**

1. *For any finite D, and any finite $\Gamma \subseteq \mathbf{\Phi}_D$,* $\mathrm{Imp}(\mathrm{wPol}(\Gamma)) = \mathrm{wRelClone}(\Gamma)$.
2. *For any finite D and any finite $W \subseteq \mathbf{W}_D$,* $\mathrm{wPol}(\mathrm{Imp}(W)) = \mathrm{wClone}(W)$.

[1] The name Imp is chosen to suggest that such cost functions are *improved* by weightings in W in the sense of the remark and discussion after Definition 18.

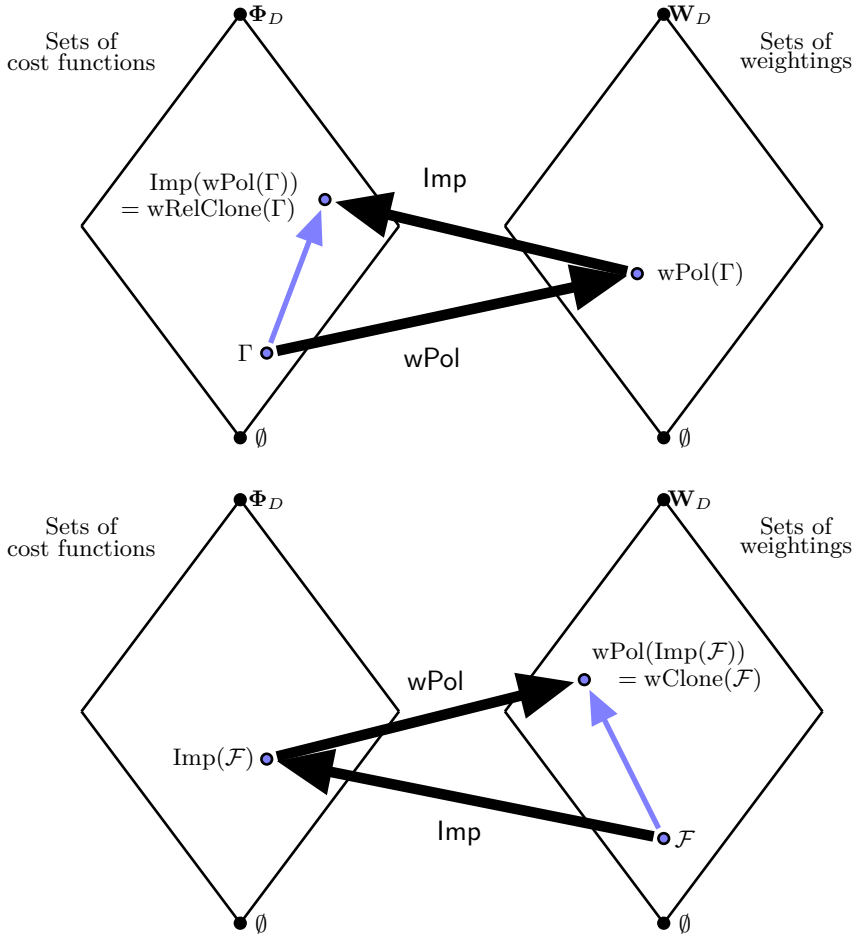

Figure 2 Galois connection between $\mathbf{\Phi}_D$ and $\mathbf{W}_D$.

▶ Remark (fractional vs weighted polymorphisms). Fractional and weighted polymorphisms are effectively the same things, just written differently. A k-ary fractional polymorphism ω can be transformed into a weighted polymorphism if, in Inequality 2, we move the right-hand side to the left and write $\phi(\mathbf{x_i})$ as $\phi(\mathrm{e}_i^{(k)}(\mathbf{x_1}, \ldots, \mathbf{x_i}, \ldots, \mathbf{x_k}))$. A similar simple manipulation transforms a k-ary weighted polymorphism into a fractional one.

Each of the two concepts is more convenient to work with depending on the context. Fractional polymorphisms are more convenient to work with when describing properties of operations in the support of fractional polymorphisms, and this relates to algorithmic and complexity consequences discussed in Sections 5 and 6. On the other hand, weighted polymorphisms are more convenient to work with when building an algebraic theory [23, 43, 77, 78].

It follows from Theorem 44 that to identify all tractable valued constraint languages on a finite set D it is sufficient to study the possible weighted clones of weighted polymorphisms on D. This provides an algebraic approach to the identification of tractable cases. In particular, it follows that weighted polymorphisms completely determine the computational complexity of valued constraint languages (the same way polymorphisms determine the computational complexity of crisp languages [62]).

For a valued constraint language Γ we denote by $\mathrm{supp}(\Gamma)$ the support clone of the weighted

clone of weighted polymorphisms of Γ; i.e., $\mathrm{supp}(\Gamma) = \mathrm{supp}(\mathrm{wPol}(\Gamma))$.

▶ **Definition 45.** A valued constraint language Γ is called a *core* if all unary operations in $\mathrm{supp}(\Gamma)$ are bijections. Moreover, Γ is called a *rigid core* if the only unary operation in $\mathrm{supp}(\Gamma)$ is the identity operation.

For $d \in D$, let u_d be a unary function on D such that $u_d(d) = 0$ and $u_d(x) = \infty$ if $x \neq d$. Rigid cores are important due to the following result.

▶ **Theorem 46** ([77, 78])**.**

1. *For any valued constraint language* Γ *there exists a valued constraint language* Γ' *which is a core such that* $\mathrm{VCSP}(\Gamma)$ *and* $\mathrm{VCSP}(\Gamma')$ *are polynomial-time equivalent.*
2. *For any core valued constraint language* Γ *there exists a valued constraint language* Γ' *which is a rigid core such that* $\mathrm{VCSP}(\Gamma)$ *and* $\mathrm{VCSP}(\Gamma')$ *are polynomial-time equivalent.*

The core language Γ' in Theorem 46 (1) is built from Γ as follows. Let $u \in \mathrm{supp}(\Gamma)$ be a unary operation with the minimum size of the set range $U = u(D)$ among all such operations. Then $\Gamma_{|U} = \{\phi_{|U} \mid \phi \in \Gamma\}$ is a core. The rigid core language Γ' in Theorem 46 (2) is $\Gamma' = \Gamma_{|U} \cup \{u_d \mid d \in U\}$, with domain U.

A k-ary operation $f : D^k \to D$ is called *idempotent* if $f(x, \ldots, x) = x$ for every $x \in D$. It is easy to show that Γ is a rigid core if and only if every operation from $\mathrm{supp}(\Gamma)$ is idempotent.

Thus, the intuition behind moving to the rigid core is that (a) one removes labels from the domain that can always be (uniformly) replaced in any solution to an instance without increasing its value, and (b) algebras with only idempotent operation are known to have much more structure (than the general case), leading to the applicability of more powerful algebraic results.

The Galois connection described in Theorem 44 implies the following result.

▶ **Theorem 47.** *Let* Γ *and* Γ' *be two valued constraint languages on the same domain and of finite size. If* $\mathrm{wPol}(\Gamma) \subseteq \mathrm{wPol}(\Gamma')$ *then* $\mathrm{VCSP}(\Gamma')$ *polynomial-time reduces to* $\mathrm{VCSP}(\Gamma)$*.*

This is a generalisation of the analogous result for the CSP: if $\mathrm{Pol}(\Gamma) \subseteq \mathrm{Pol}(\Gamma')$ for crisp languages Γ and Γ' then $\mathrm{CSP}(\Gamma')$ polynomial-time reduces to $\mathrm{CSP}(\Gamma)$ [62].

The algebraic theory of the CSP extends beyond clones to finite algebras and varieties of algebras (see [13, 18, 85], see also the surveys in [32]), where a variety is a class of algebras over the same signature defined by a set of identities. This extension explains why the complexity of a (crisp) language is determined by the identities, i.e., universally quantified equations, satisfied by its polymorphisms, which is why we usually define the relevant operations by identities. This extension was instrumental in obtaining most state-of-the-art results in this area (e.g. [1, 4, 6, 14, 15, 17, 19, 57, 85]). The basics of this theory were extended to VCSPs in [77, 78], introducing weighted algebras and weighted varieties, which produced a hardness result and a tractability conjecture for VCSPs - we discuss them in Section 6.

While weighted clones were introduced primarily for the understanding of the computational complexity of valued constraint languages, there have been several studies of the purely algebraic structure of weighted clones [113, 112, 104, 63].

5 Algorithms

A curious feature of research into the tractability of constraint languages is that all languages known to be tractable have been shown tractable by using very few algorithmic techniques.

Despite many tractability results concerning crisp languages (i.e., the CSP), only two algorithmic techniques have so far been sufficient, and the applicability of each of them individually has been characterised by specific algebraic conditions.

The first technique is based on enforcing local consistency, which is a natural algorithm for dealing with (crisp) constraints. There are several closely related variants of this algorithm, we will describe one of them. Fix parameters $1 \leq \kappa \leq \ell$. For a given CSP instance, this algorithm starts by adding a new constraint for each subset of variables of size ℓ, the new constraints initially allowing all tuples. Then the algorithm repeatedly discards (i.e., disallows) tuples of labels in the constraints as follows. For every set W of at most κ variables and every pair of constraints ϕ_1, ϕ_2 whose scopes contain W, discard all assignments for ϕ_1 whose restriction on W is inconsistent with the restriction of ϕ_2 to W. Eventually, either all assignments (for at least one constraint) are discarded or else local consistency is established; this procedure takes polynomial time for any fixed D and any fixed κ, ℓ. The former case implies no feasible assignments. One says that a CSP has *relational width* (κ, ℓ) if the latter case implies the existence of a feasible assignment. A CSP has bounded relational width if it has relational width (κ, ℓ) for some κ, ℓ. The power of local consistency, i.e., a precise characterisation of crisp languages that give rise to VCSP instances solvable by some form of local consistency, has recently been established [3, 6, 15, 86] (see also [16, 75]).

▶ **Theorem 48** (Bounded Width [3, 6, 15, 86])**.** *Let Γ be a crisp language. Then the following are equivalent:*

1. VCSP(Γ) *has bounded relational width;*
2. VCSP(Γ) *has relational width (2,3);*
3. *For all but finitely many k,* Pol(Γ) *contains a k-ary operation satisfying, for all $x, y \in D$,*

$$f(y, x, x, \ldots, x) = f(x, y, x, x, \ldots, x) = f(x, x, \ldots, x, y); \tag{8}$$

4. *For every $k \geq 3$,* Pol(Γ) *contains a k-ary operation satisfying (8);*

A k-ary ($k \geq 2$) idempotent operation satisfying (8) is called a *weak near-unanimity* operation. We remark that that there are many algebraic conditions equivalent to Pol(Γ) containing weak near-unanimity operations of all but finitely many arities. One such condition is that Pol(Γ) contains two weak near-unanimity operations, 3-ary w_3 and 4-ary w_4, such that $w_3(x, x, y) = w_4(x, x, x, y)$ [76].

The second standard algorithmic technique for the CSP is based on the property of having a polynomial-sized representation (a generating set) for the solution set of any instance [17, 57]. Roughly, the algorithm works by starting from the empty set and adding the constraints of the instance one by one while maintaining (in polynomial time) a small enough representation of the current solution set (of feasible assignments). At the end (i.e., after all constraints have been added), either this representation is non-empty and contains a solution to the instance or else there is no solution. In a way, this technique is a generalisation of Gaussian elimination. This algorithm is often called "few subpowers" because it is related to a certain algebraic property to do with the number of subalgebras in powers of an algebra. The power of this algorithm was established in [57]. A k-ary ($k \geq 3$) operation $f : D^k \to D$ is called an *edge* operation if, for all $x, y \in D$,

$$f(y, y, x, x, \ldots, x) = f(y, x, y, x, x, \ldots, x) = x$$

and, for all $4 \leq i \leq k$,

$f(x, \ldots, x, y, x, \ldots, x) = x$ where y is in position i.

▶ **Theorem 49** (Few Subpowers [57]). *Let* Γ *be a crisp language. Then* VCSP(Γ) *is solvable by the few subpowers algorithm if* Pol(Γ) *contains an edge operation.*

The converse to this theorem is true in the following sense: the absence of edge operations from Pol(Γ) implies that the presence of small enough representations is not guaranteed, see [57] for details. Interestingly, the few subpowers algorithm makes use of the actual edge operations in its work (in contrast with bounded width, where the weak near-unanimity operations are used only to guarantee correctness).

For the general VCSP another algorithm, based on linear programming, has been the most thoroughly investigated. Every VCSP instance has a natural integer linear programming (ILP) formulation. Let Φ be a VCSP instance defined by $\Phi(\mathbf{x}) = \sum_{i=1}^{q} \phi_i(\mathbf{x}_i)$, with a set of variables V. For each i, let S_i be the set of variables appearing in $\mathbf{x}_i$; assignments to $\mathbf{x}_i$ are naturally associated with elements in D^{S_i}.

The ILP formulation involves variables $\mu_x(a)$, where $x \in V$ and $a \in D$, and $\lambda_i(\mathbf{s})$, where $1 \le i \le q$ and $\mathbf{s} \in D^{S_i}$. All variables take values in $\{0, 1\}$. The intuition is that $\mu_x(a) = 1$ in a solution to the ILP formulation if x is assigned label a in the corresponding solution to Φ. The ILP formulation includes constraints (9c) that ensure that, for any $x \in V$, exactly one variable $\mu_x(a)$ is assigned 1. Similarly, $\lambda_i(\mathbf{s}) = 1$ corresponds to $\mathbf{s}$ being assigned to $\mathbf{x}_i$. The ILP formulation also includes constraints (9b) enforcing consistency between the two types of variables in the ILP. The ILP instance BILP(Φ) associated with Φ is defined as follows:

$$\text{BILP}(\Phi) \stackrel{\text{def}}{=} \min \quad \sum_{i=1}^{q} \sum_{\mathbf{s} \in D^{S_i}} \phi_i(\mathbf{s})\, \lambda_i(\mathbf{s}) \tag{9a}$$

$$\text{s.t.} \quad \sum_{\mathbf{s} \in D^{S_i} \mid \mathbf{s}(x) = a} \lambda_i(\mathbf{s}) = \mu_x(a), \quad 1 \le i \le q,\ x \in S_i,\ a \in D \tag{9b}$$

$$\sum_{a \in D} \mu_x(a) = 1, \quad x \in V \tag{9c}$$

$$\lambda_i(\mathbf{s}) = 0, \quad 1 \le i \le q,\ \phi_i(\mathbf{s}) = \infty \tag{9d}$$

Note that terms in (9a) corresponding to (9d) are assumed to be equal to 0.

If we allow the variables in BILP(Φ) to take arbitrary real values in the interval $[0, 1]$, we obtain a relaxation called the *basic LP relaxation* (BLP) of Φ, denoted by BLP(Φ). The variables can then be seen as probability distributions on D and D^{S_i}, respectively. The marginalization constraints (9b) impose that μ_x is the marginal of $\lambda_i(\mathbf{s})$, for each constraint and each variable x in the scope of that constraint.

We remark that an LP relaxation of the VCSP, similar or closely related to (9), has been proposed independently by many authors; we refer the reader to [73] and the references therein.

Given a VCSP instance Φ, the optimal value of BLP(Φ), which can be found in polynomial time, is always a lower bound for the optimal value of Φ. We say that BLP *solves* Φ if the optimal value of BLP(Φ) is equal to the optimal value of Φ. Moreover, we say that BLP solves a valued constraint language Γ if BLP solves every instance $\Phi \in$ VCSP(Γ). It is shown in [73] that in all cases where BLP solves Γ, a standard self-reduction method can be used to obtain an assignment that minimises any Φ in VCSP(Γ) in polynomial time. For $d \in D$, let u_d be a unary function on D such that $u(d) = 0$ and $u(x) = \infty$ if $x \neq d$. For rigid cores, the self-reduction method goes through the variables in some order, finding $d \in D$ for the current variable v such that instances Φ and $\Phi + u_d(v)$ have the same optimal value (which can be checked by BLP), updating $\Phi := \Phi + u_d(v)$, and moving to the next variable. At the

end, the instance will have a unique feasible assignment whose value is the optimum of the original instance. Hence if BLP solves Γ, then Γ is tractable.

The power of BLP for valued constraint languages was fully characterised in [103]. To state this result, we first introduce some further terminology about operations. A k-ary operation $f : D^k \to D$ is called *symmetric* if for every permutation π on $\{1, \dots, k\}$, $f(x_1, \dots, x_k) = f(x_{\pi(1)}, \dots, x_{\pi(k)})$. A fractional polymorphism ω is called symmetric if $\mathrm{supp}(\omega)$ is non-empty and contains symmetric operations only. Finally, we say that an operation f is *generated* from a set of operations $F \subseteq \mathbf{O}_D$ if $f \in \mathrm{Clone}(F)$.

▶ **Theorem 50** (Power of BLP for Arbitrary Languages [103]). *Let Γ be a valued constraint language. Then the following are equivalent:*

1. *BLP solves Γ;*
2. *For every $k \geq 2$, Γ admits a k-ary symmetric fractional polymorphism;*
3. *For every $k \geq 2$, Γ admits a fractional polymorphism (not necessarily k-ary) ω_k such that $\mathrm{supp}(\omega_k)$ generates a symmetric k-ary operation.*

Condition (3) has turned out to be very useful for proving the tractability of many valued constraint languages. A binary operation $f : D^2 \to D$ is called a *semilattice* operation if f is associative, commutative, and idempotent. Since any semilattice operation trivially generates symmetric operations of all arities, Theorem 50 shows that any valued constraint language with a binary fractional polymorphism whose support includes a semilattice operation is solvable using the BLP. This immediately implies that all of the following cases are solvable using the BLP, and hence tractable: languages with a (generalised) submodular multimorphism (Example 20), a bisubmodular multimorphism (Example 23), a k-submodular multimorphism for any k (Example 23), a symmetric tournament pair multimorphism (Example 25), or a skew bisubmodular fractional polymorphism (Example 24), or the fractional polymorphisms describing submodularity on modular semilattices [53]. Moreover, a not very difficult argument can be used to show that languages with a 1-defect multimorphism (Example 27) also satisfy condition (3) of Theorem 50 [103], and thus are tractable.

Examples of problems VCSP(Γ) that are tractable, but not solvable by BLP, include (the crisp) 2-Sat problem and 3-Lin-k, the (crisp) problem of solving system of linear equations modulo k with 3 variables per equation, and some languages with a tournament pair multimorphism (Example 25).

Recall that, for any crisp language Γ, any probability distribution on k-ary polymorphisms of Γ is a fractional polymorphism of Γ. Thus, Theorem 50 can be re-stated for crisp languages as follows.

▶ **Theorem 51** (Power of BLP for Crisp Languages [82]). *Let Γ be a crisp constraint language. Then the following are equivalent:*

1. *BLP solves Γ;*
2. *For every $k \geq 2$, $\mathrm{Pol}(\Gamma)$ contains a k-ary symmetric polymorphism;*

It is unknown whether condition (2) in Theorem 51 is decidable, hence the same can be said about the conditions in Theorem 50.

Recent work identified a sufficient instance-based condition for binary CSPs admitting a cyclic polymorphism (cf. Section 6) that are solvable by BLP [12].

For valued constraint languages where the cost functions take only finite values, Theorem 50 has been strengthened further [73].

▶ **Theorem 52** (Power of BLP for Finite-Valued Languages [73]). *Let Γ be a valued constraint language where every cost function takes only finite values. Then the following are equivalent:*

1. *BLP solves* Γ*;*
2. *For every* $k \geq 2$*,* Γ *admits a* k*-ary symmetric fractional polymorphism;*
3. *For some* $k \geq 2$*,* Γ *admits a* k*-ary symmetric fractional polymorphism;*
4. Γ *admits a binary symmetric fractional polymorphism;*
5. Γ *admits a fractional polymorphism* ω *such that* $\mathrm{supp}(\omega)$ *generates a symmetric operation.*

In contrast with Theorems 50 and 51, condition (4) in Theorem 52 is decidable because deciding whether Γ admits a fixed-arity symmetric fractional polymorphism (by its definition) amounts to solving a linear program.

The Sherali-Adams hierarchy [97] provides successively tighter LP relaxations of an integer LP. Higher levels of this hierarchy can potentially solve more VCSPs than BLP does. Fix parameters $1 \leq \kappa \leq l$. For a VCSP instance $\Phi(\mathbf{x}) = \sum_{i=1}^{q} \phi_i(\mathbf{x}_i)$, with a set of variables V, its Sherali-Adams relaxation $\mathrm{SA}_{\kappa,\ell}(\Phi)$ is defined as follows. First, we ensure that every non-empty $S \subseteq V$ with $|S| \leq \ell$ appears in some term $\phi_i(\mathbf{x}_i)$, possibly by adding 0-valued constraints (this is similar to how we start the local consistency algorithm described above). The variables are $\lambda_i(\mathbf{s})$ for every $1 \leq i \leq q$ and every tuple $\mathbf{s} \in D^{S_i}$, they take values in the interval $[0, 1]$.

$$\mathrm{SA}_{\kappa,\ell}(\Phi) \stackrel{\text{def}}{=} \min \quad \sum_{i=1}^{q} \sum_{\mathbf{s} \in D^{S_i}} \phi_i(\mathbf{s})\, \lambda_i(\mathbf{s}) \tag{10a}$$

$$\text{s.t.} \quad \lambda_j(\mathbf{t}) = \sum_{\mathbf{s} \in D^{S_i} \,:\, \mathbf{s}_{|S_j} = \mathbf{t}} \lambda_i(\mathbf{s}), \quad 1 \leq i, j \leq q,\ S_j \subseteq S_i,\ |S_j| \leq \kappa,\ \mathbf{t} \in S_j \tag{10b}$$

$$\sum_{\mathbf{s} \in D^{S_i}} \lambda_i(\mathbf{s}) = 1, \quad 1 \leq i \leq q \tag{10c}$$

$$\lambda_i(\mathbf{s}) = 0, \quad 1 \leq i \leq q,\ \phi_i(\mathbf{s}) = \infty \tag{10d}$$

As with BLP, terms in (10a) corresponding to (10d) are assumed to be equal to 0.

Say that a valued language Γ has *valued relational width* (κ, ℓ) if the optimum value of $\mathrm{SA}_{\kappa,\ell}(\Phi)$ is equal to the optimal value of Φ for every instance Φ of VCSP(Γ). Also, say that Γ has *bounded* valued relational width if it has valued relational width (κ, ℓ) for some κ, ℓ.

▶ **Theorem 53** (Power of SA for Arbitrary Languages [105, 108])**.** *Let* Γ *be a valued constraint language. Then the following are equivalent:*

1. Γ *has bounded valued relational width;*
2. Γ *has valued relational width (2,3);*
3. *For every* $k \geq 3$*,* $\mathrm{supp}(\Gamma)$ *contains a* k*-ary (not necessarily idempotent) operation satisfying the weak near-unanimity identities described in (8).*

We remark that condition (3) in the above theorem plays a role not only here and in Theorem 48, it also characterize the so-called robust approximability of CSPs [5].

Examples of languages solvable by SA but *not* by BLP include (the crisp) 2-Sat problem and certain languages admitting a tournament pair multimorphism (cf. Example 25), see [73, Example 5] for more details. The problem 3-Lin-k mentioned above is an example of a tractable problem (solvable by few subpowers) not solvable by SA.

Finally, we also remark that it has recently been shown that if a valued constraint language Γ does not have bounded valued relational width then not only is VCSP(Γ) not solved by a constant level of the Sherali-Adams hierarchy but actually VCSP(Γ) is not solved even by linear levels of the Lasserre semidefinite programming relaxation [107].

6 Complexity classifications

As mentioned before, the ultimate goal of the research direction that we survey is to obtain complexity classifications: clear descriptions of which VCSPs are tractable and which are not. The usual way to approach this task has been to first restrict constraint languages under consideration to rigid cores, this can be done without loss of generality [77, 78]. After that, one proves an algebraic dichotomy theorem, which states that every rigid core language either expresses some NP-hard problem and therefore is NP-hard itself, or else it has polymorphisms with some nice properties (usually in the form of identities).

We mentioned above that the tractability of constraint languages seems to arise from very few algorithmic techniques. Interestingly, the hardness of constraint languages seems to arise from a single specific problem 1-in-3-Sat (see Example 5)!

A *Taylor* operation is a k-ary ($k \geq 2$) idempotent operation f such that, for each $1 \leq i \leq k$, it satisfies an identity of the form

$$f(\square_1, \square_2, \ldots, \square_k) = f(\triangle_1, \triangle_2, \ldots, \triangle_k) \tag{11}$$

where $\square_i = x, \triangle_i = y$ and $\square_s, \triangle_t \in \{x, y\}$ for all s, t. (Note that such identities are the weakest identities that prevent f from being a projection.) The following theorem can be derived from [102] (see also [18]).

▶ **Theorem 54.** *For a crisp constraint language Γ on D that is a rigid core,*

- *either* $\mathrm{Pol}(\Gamma)$ *contains a Taylor operation,*
- *or else* $\mathrm{wRelClone}(\Gamma)$ *contains a crisp function ϕ such that*

$$\mathrm{Opt}(\phi) = \{(x, y, z) \in A^3 \mid \phi_{\text{1-in-3}}(g(x), g(y), g(z)) = 0\} \tag{12}$$

for some $A \subseteq D$ and some function $g : A \to \{0, 1\}$.

It follows from Theorem 35 that if a crisp rigid core Γ has no Taylor polymorphism then $\mathrm{VCSP}(\Gamma)$ is NP-hard.

For a crisp language Γ that is a rigid core, having a Taylor polymorphism is equivalent to any one of the following conditions:

1. Γ has a weak near-unanimity polymorphism of some arity $k \geq 2$ [13, 90];
2. Γ has a *cyclic* polymorphism [4], i.e. a k-ary ($k \geq 2$) idempotent polymorphism f such that

$$f(x_1, x_2, \ldots, x_k) = f(x_2, \ldots, x_k, x_1); \tag{13}$$

3. Γ has a 6-ary *Siggers* polymorphism [98], i.e. a 6-ary idempotent polymorphism f that satisfies identities

$$f(x, x, x, x, y, y) = f(x, y, x, y, x, x),$$
$$f(y, y, x, x, x, x) = f(x, x, y, x, y, x);$$

One can visualise these identities by thinking of a three-element complete graph (triangle) whose vertices are $\begin{pmatrix} x \\ x \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix}, \begin{pmatrix} y \\ x \end{pmatrix}$. Then the pairs of vertices that appear in the 6 coordinates (first coordinate on the right and the first coordinate on the left, and so on) give a complete list of edges of the triangle, each edge being a pair of directed edges in opposite directions.

4. Γ has a 4-ary polymorphism f [69] (sometimes also called a Siggers polymorphism) satisfying the identity[2]

$$f(y,x,y,z) = f(x,y,z,x),$$

The Algebraic CSP Dichotomy conjecture (originally stated in [18] in a different, but equivalent, form) mentioned in Example 11 is the following.

▶ Conjecture 1 (Algebraic CSP Dichotomy Conjecture). A crisp language Γ that is a rigid core is tractable if Γ has a Taylor polymorphism (or, equivalently, satisfies one of the four conditions above), and it is NP-hard otherwise.

The hardness part is known, as explained above, and it is the tractability part that is the conjecture. This conjecture refines the original Feder-Vardi dichotomy conjecture [37] by specifying the boundary in algebraic terms. Conjecture 1 was confirmed in many special cases, for example, for crisp languages over two-element sets [95] and three-element sets [14] and for crisp languages containing all unary crisp functions [19, 1]. The conjecture is still open, but widely believed to hold in full generality.

Obviously, a complete classification of VCSPs would include a complete classification for crisp languages. It turns out, however, that the latter classification implies the former one as we now discuss.

A fractional operation ω is said to be cyclic if all operations in $\mathrm{supp}(\omega)$ are cyclic. The following lemma is contained in the proof of Theorem 50 in [78].

▶ **Lemma 55.** *Let Γ be a rigid core on a set D. Then the following are equivalent:*

1. $\mathrm{supp}(\Gamma)$ *contains a Taylor operation of arity at least 2;*
2. Γ *has a cyclic fractional polymorphism of (some) arity at least 2;*
3. Γ *has a cyclic fractional polymorphism of every prime arity $p > |D|$.*

The following theorem is Corollary 51 from [78].

▶ **Theorem 56** ([78]). *Let Γ be a valued constraint language that is a rigid core. If* $\mathrm{supp}(\Gamma)$ *does not contain a Taylor operation then Γ is NP-hard.*

It is actually shown in [78] that if $\mathrm{supp}(\Gamma)$ does not contain a Taylor operation then $\mathrm{wRelClone}(\Gamma)$ contains a (not necessarily crisp) function ϕ satisfying condition (12).

Kozik and Ochremiak state a conjecture (which they attribute to L. Barto) that the above theorem describes all NP-hard valued constraint languages, and all other languages are tractable. Using Lemma 55, we restate their original conjecture via cyclic fractional polymorphisms.

▶ Conjecture 2 (Algebraic VCSP Dichotomy Conjecture [77, 78]). Let Γ be a valued constraint language that is a rigid core. If Γ has a cyclic fractional polymorphism of arity at least 2, then Γ is tractable, and it is NP-hard otherwise.

Note that, for fixed D, the problem of checking whether a given finite rigid core Γ has a cyclic fractional polymorphism of some arity can be solved in polynomial time. Indeed, if $p > |D|$ is some fixed prime number, then it is sufficient to check for a cyclic fractional polymorphism of arity p. Such polymorphisms, by definition, are solutions to a system of linear inequalities. Since the number of cyclic operations of arity p on D is constant (because

[2] Using different variables, $f(r,a,r,e) = f(a,r,e,a)$ – mnemonic due to Ryan O'Donnell.

we assume that D is a fixed finite set), the system will have size polynomial in Γ and its feasibility can be decided by linear programming.

Recall that, for a (possibly infinite) crisp language, any probability distribution on polymorphisms (of the same arity) is a fractional polymorphism. Then Theorem 56 is a direct generalisation of the above-mentioned corresponding result for crisp languages. Moreover, Conjecture 2, when restricted to crisp languages, gives precisely Conjecture 1.

For a constraint language Γ, let $\mathrm{Feas}(\Gamma) = \{\mathrm{Feas}(\phi) \mid \phi \in \Gamma\}$. Thus, $\mathrm{CSP}(\mathrm{Feas}(\Gamma))$ is the problem of deciding whether an given instance of $\mathrm{VCSP}(\Gamma)$ has a feasible solution. It is obvious that, for $\mathrm{VCSP}(\Gamma)$ to be tractable, $\mathrm{CSP}(\mathrm{Feas}(\Gamma))$ must also be tractable.

▶ **Theorem 57** (Classification of General-Valued Languages [72]). *Let Γ be a valued constraint language over domain D that is a rigid core. If the following conditions hold then* $\mathrm{VCSP}(\Gamma)$ *is tractable:*

1. *Γ has a cyclic fractional polymorphism of arity at least 2, and*

2. $\mathrm{Feas}(\Gamma)$ *is tractable.*

Otherwise, Γ is not tractable.

Notice that the above theorem shows that a classification for crisp languages implies the classification for all languages, whether or not the boundary is as predicted by the Algebraic CSP Dichotomy Conjecture. In particular, it follows that Conjecture 1 implies Conjecture 2. Moreover, if the Algebraic CSP Dichotomy Conjecture holds then condition (2) can be removed from Theorem 57, since it would be implied by condition (1).

Theorem 57 implies classification within any class of languages Γ such that the classification for the class of corresponding languages $\mathrm{Feas}(\Gamma)$ is known. For example, this is the case for the class of languages containing all unary crisp functions [19, 1] or for the class of languages over a two- or three-element domain [95, 14].

The necessity of conditions (1-2) in Theorem 57 is clear. To explain why they are sufficient, we need to give a definition.

Let Φ be a VCSP instance over variables V. For each variable $v \in V$, let $D_v = \{d \in D \mid d = \sigma(v)$ for some feasible solution σ for $\Phi\}$. Then the $(1, \infty)$-*minimal instance* $\bar{\Phi}$ associated with Φ is the VCSP instance obtained from Φ by adding, for each $v \in V$, the constraint $u_{D_v}(x_v)$, where u_{D_v} is a crisp function such that $u_{D_v}(d) = 0$ if and only if $d \in D_v$. Note that if Γ is a rigid core and the problem $\mathrm{CSP}(\mathrm{Feas}(\Gamma))$ is tractable, then, for any instance Φ of $\mathrm{VCSP}(\Gamma)$, one can construct the associated $(1, \infty)$-minimal instance in polynomial time. Indeed, to find out whether a given $d \in D$ is in D_v, one only needs to decide whether the CSP instance obtained from $\mathrm{Feas}(\Phi)$ by adding the constraint $u_d(x_v)$ is satisfiable. Since Γ is a rigid core, one can assume that $u_d \in \Gamma$, so the latter instance is also an instance of $\mathrm{CSP}(\mathrm{Feas}(\Gamma))$.

If Γ is a rigid core satisfying the conditions in Theorem 57 then, for every instance Φ of $\mathrm{VCSP}(\Gamma)$, the optimal value of $BLP(\bar{\Phi})$ is the same as the optimal value of Φ, as proved in [72]. This algorithm (first computing $\bar{\Phi}$ and then finding its optimal value) allows one to find the optimal value of any instance in polynomial time and then find an optimal solution via self-reduction, as discussed earlier.

We would like to point out two surprising features of Theorem 57. The first one is that the algorithm described above that solves all tractable cases uses feasibility checking only as a black-box. The second one is that the proof of Theorem 57 does not involve structural universal algebra used for CSP classifications and also in the proof of Theorem 56.

Tighter tractability conditions (than those given in Theorem 57) are known for a number of important special cases.

For finite-valued languages, condition (2) in Theorem 57 is trivial and can be removed, while condition (1) can be replaced by a much stronger condition.

▶ **Theorem 58** (Classification of Finite-Valued Languages [106]). *Let* Γ *be a finite-valued constraint language that is a core. Either* Γ *has a binary symmetric fractional polymorphism (and hence is solvable by BLP), or else* Γ *is NP-hard.*

It is shown in [106] that, for any NP-hard finite-valued Γ, wRelClone(Γ) contains a binary function ϕ such that $\mathrm{minarg}(\phi) = \{(a,b),(b,a)\}$ for some distinct $a, b \in D$ (which can be obtained from Γ even without using the operator Opt), thus simulating Max-Cut (see Example 7). We have seen in Example 34 that, on domain $\{0,1\}$, wRelClone$(\{\phi_{\mathsf{xor}}, u_0, u_1\})$ contains $\phi_{\mathsf{1\text{-}in\text{-}3}}$. Since weighted relational clones are closed under scaling by non-negative rational constants and addition of rational constants, we have $\phi_{\mathsf{1\text{-}in\text{-}3}} \in \mathrm{wRelClone}(\{\phi, u_0, u_1\})$.

Theorem 58 generalises several previous classification results for finite-valued languages. Tractability in these earlier results was often characterised by (more) specific binary symmetric fractional polymorphisms:

- A core $\{0,1\}$-valued language[3] over a two-element set [70, 33], or over a three-element set [64], or including all unary $\{0,1\}$-valued functions [36] is tractable if it is submodular on a chain (cf. Examples 19 and 20), and NP-hard otherwise.
- A core $\{0,1\}$-valued language over a four-element set [66] is tractable if it is submodular on some lattice (cf. Example 20) or 1-defect (cf. Example 27) and NP-hard otherwise.
- A core finite-valued language over a two-element set [25] is tractable if it is submodular (cf. Example 19) and NP-hard otherwise.
- A core finite-valued language over a three-element set [55] is tractable if it is submodular on a chain (cf. Example 20) or skew bisubmodular (cf. Example 24) and NP-hard otherwise.
- A finite-valued language containing all $\{0,1\}$-valued unary cost functions [74] is tractable if it is submodular on a chain (cf. Example 25) and NP-hard otherwise.

Theorem 58 also implies a classification of the so-called Min-0-Ext problems [53].

A tight complete complexity classification for valued constraint languages over a two-element set was established in [25]. Note that on a two-element set there is precisely one majority operation, as defined in Example 28, which we will denote by Mjrty, and precisely one minority operation, as defined in Example 29, which we will denote by Mnrty. There are also precisely two constant operations, which will be denoted Const_0 and Const_1.

▶ **Theorem 59** (Classification of Boolean Languages [25]). *A valued constraint language* Γ *on* $D = \{0,1\}$ *is tractable if it admits at least one of the following eight multimorphisms. Otherwise* wRelClone(Γ) *contains* $\phi_{\mathsf{1\text{-}in\text{-}3}}$ *and* Γ *is NP-hard.*

1. $\langle \mathrm{Const}_0 \rangle$
2. $\langle \mathrm{Const}_1 \rangle$
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. $\langle \mathrm{Mjrty}, \mathrm{Mjrty}, \mathrm{Mnrty} \rangle$.

Let us compare Theorem 59 with a classification of crisp Boolean languages, originally established by Schaefer in [95] and restated here using polymorphisms (see, e.g. [22]): A

[3] $\{0,1\}$-valued languages correspond to Max-CSPs, cf. Example 13.

crisp constraint language on $D = \{0, 1\}$ is tractable if it admits one of the following six polymorphisms: Const_0, Const_1, Min, Max, Mjrty, Mnrty; otherwise it is NP-hard. These six tractable cases are covered by cases (1-4), (6), and (7) in Theorem 59. The six cases correspond to sets of Boolean relations that are 0-valid, or 1-valid, or expressible by Horn clauses, dual Horn clauses, 2-clauses, or linear equations over the field with 2 elements, respectively.

The hardness part of Theorem 59 can be rederived using the algebraic theory described in Section 4; see [31, 23] for details. We remark that if we restrict to core Boolean valued constraint languages, the first two cases in Theorem 59 disappear as those languages are not cores (and in fact are solvable trivially). The original proof of Theorem 59 identified ϕ_{nae} (Example 6) and ϕ_{xor} (Example 7) as sources of hardness [25]. However, for rigid cores we have seen in Example 34 how to obtain $\phi_{\mathsf{1\text{-}in\text{-}3}}$ from ϕ_{xor} and it is well known that $\phi_{\mathsf{1\text{-}in\text{-}3}}$ and ϕ_{nae} can express each other in this case (see, e.g. [95]).

Another general complexity classification result concerns languages that contain all $\{0, 1\}$-valued unary cost functions. Note that a fractional polymorphism ω is called *conservative* if $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$ for all $f \in \mathrm{supp}(\omega)$.

▶ **Theorem 60** (Classification of Conservative Languages [74]). *Let Γ be a valued constraint language on a set D such that Γ contains all $\{0, 1\}$-valued unary cost functions on D. Then either Γ admits a conservative binary multimorphism $\langle f_1, f_2\rangle$ and a conservative ternary multimorphism $\langle f'_1, f'_2, f'_3\rangle$ and there is a family M of 2-element subsets of D, such that:*

1. *for every $\{a, b\} \in M$, $\langle f_1, f_2\rangle$ restricted to $\{a, b\}$ is a symmetric tournament pair (see Example 25), and*
2. *for every $\{a, b\} \notin M$, $\langle f'_1, f'_2, f'_3\rangle$ restricted to $\{a, b\}$ is an MJN multimorphism (see Example 30),*

in which case Γ is tractable, or else Γ is NP-hard.

It is shown in [108] that the tractable cases in Theorem 60 can be equivalently characterised by the condition that $\mathrm{supp}(\Gamma)$ contains a majority operation (see also Example 31).

The original algorithm for solving the tractable cases identified in Theorem 60 was similar to (and in fact inspired) the general algorithm for tractable VCSPs: after establishing some sort of local consistency, any instance admits a symmetric tournament pair multimorphism [74] and is thus solvable using BLP. It was shown in [108] that all tractable languages identified in Theorem 60 in fact have valued relational width (2,3).

Recall the Min-Cost-Hom and Min-Sol problems discussed in Examples 14 and 16 respectively. Recall that a Min-Cost-Hom problem corresponds to VCSP(Γ) for some language Γ containing only crisp cost functions and unary cost functions. We now briefly describe the classification results so far obtained for these problems that give more information than the general Theorem 57. It may seem that the Min-Cost-Hom framework is rather more restrictive than the (general-valued) VCSP. However, it was shown in [26], by adapting the main result of [20], that for every problem VCSP(Γ), where Γ is finite, there is a polynomial-time equivalent Min-Cost-Hom problem, VCSP(Γ'), where Γ' contains only a single crisp binary function and a single finite-valued unary function. This mirrors a similar reduction from the general CSP (Example 11) to the digraph homomorphism problem (Example 12) which was first established in [37].

The complexity classification for Min-Cost-Hom for languages containing all unary cost functions was established in [100]. The tractable case can be reduced, after a preprocessing step using local consistency techniques, to a certain problem on perfect graphs known to be solvable in polynomial time using linear programming [48]. For the special case of digraphs

(i.e., when the only non-unary cost function allowed is a single binary crisp cost function), a complexity classification in graph-theoretic terms was obtained in [52]. The classification of Min-Cost-Hom for languages containing all unary crisp cost functions was initially studied in [101] and fully established in [110].

The complexity of Min-Cost-Hom for all languages over a three-element set was classified in [111]. The only tractable cases either admit a fractional polymorphism with a semilattice operation in its support or a certain type of tournament pair. The former case is tractable using BLP by Theorem 50 and the latter case is tractable using a reduction to the result in [100] discussed above.

Min-Sol problems are Min-Cost-Hom problems where the only unary cost function in Γ is a specific injective and finite-valued cost function. The classification of Min-Sol problems for various special cases was established in [70, 65, 68, 110]. It was shown in [108] that every Min-Sol problem satisfies either the conditions of Theorem 56 or the conditions of Theorem 53, thus providing a full dichotomy for such problems. That dichotomy, as well as Theorem 60, are corollaries of the following result, which does not follow from Theorem 57.

▶ **Theorem 61** (Classification of General-Valued Languages with an Injective Function [108]). *Let Γ be a valued constraint language Γ on D that is a rigid core. Assume that Γ can express a unary finite-valued cost function $u : D \to \mathbb{Q}$ that is injective, i.e. $u(a) \neq u(b)$ for any $a \neq b \in D$. Then either Γ has bounded valued relational width (and hence is solvable by SA), or Γ is NP-hard.*

7 The oracle model

In this paper we have assumed that the objective function in our problem is represented as a sum of functions each defined on some subset of the variables. There is a rich tradition in combinatorial optimisation of studying problems where the objective function is represented instead by a value-giving *oracle*. In this model a problem is tractable if it can be solved in polynomial time using only polynomially many queries to the oracle (where the polynomial is in the number of variables). Note that any query to the oracle can be simulated in linear time in the VCSP model. Hence, a tractability result (for a class of functions) in the oracle model automatically transfers to the VCSP model, while hardness results automatically transfer in the opposite direction.

One class of functions that has received particular attention in the oracle model is the class of submodular functions (cf. Example 19). There are several known algorithms for minimising a (finite-valued) submodular function using only a polynomial number of calls to a value-giving oracle (see [58, 59, 88, 96]).

The fastest general (strongly polynomial) algorithm [88] runs in $O(n^3 \log^2 n \cdot EO + n^4 \log^{O(1)} n)$ time, where EO is the time for function evaluation by oracle. However, for some submodular valued constraint languages Γ, VCSP(Γ) can be solved more efficiently than by using these general approaches. For example, the (submodular) language Γ_{cut} defined in Example 9 can be solved in cubic time using the Min-Cut-based algorithm described in Example 9. A similar efficient approach can be used for all languages that are expressible over Γ_{cut}. However, it was shown in [115, 117] that not all submodular functions are expressible over Γ_{cut}, so this approach cannot be directly extended to solve arbitrary submodular VCSP instances. It is currently an open question whether the minimisation problem for submodular functions defined by sums of bounded arity submodular functions in the VCSP model is easier than general submodular function minimisation in the oracle model.

Other classes of finite-valued functions that can be efficiently minimised in the oracle model include bisubmodular and α-bisubmodular functions (Examples 23 and 24) [39, 41, 92, 42, 56], functions with a 1-defect multimorphism (Example 27) [66], and functions that are submodular on certain lattices (Example 20) [40, 79, 81]. The complexity of submodular function minimisation in the oracle model over arbitrary non-distributive lattices is still unknown (in the VCSP model, all such language are tractable, by Theorem 50).

The following general problem was mentioned in [55, 66, 103]: which fractional/weighted polymorphisms ω are sufficient to guarantee an efficient minimization algorithm, in the value-oracle model, for the class of functions $\mathrm{Imp}(\omega)$? This problem can be asked only for classes of finite-valued functions, or for classes of general-valued functions - in the latter case, some representation of feasible tuples should be part of input. Natural candidates for which the question is open include the k-submodularity multimorphism for $k \geq 3$ from Example 23 and submodularity multimorphisms on many lattices from Example 20.

8 Conclusions and future directions

We have shown 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 described in Section 4 the very general algebraic techniques now being developed to classify the complexity of these languages.

This classification is still incomplete. In fact, even in the special case of the CSP (Example 11), where all cost functions take only the values 0 or ∞, there is still no complete classification of complexity for the corresponding constraint languages. This problem has been studied for many years, beginning with the seminal work of Feder and Vardi who conjectured that any such language will be either tractable or NP-complete [37]. This conjecture is still unresolved. However, the Algebraic Dichotomy conjecture (see Conjecture 1) specifies the boundary between tractable and NP-hard languages, and it has been proved in many important cases. It was stated in [61] that "it is desirable to develop the algebraic theory of VCSPs to the point where one could make a credible algebraic dichotomy conjecture for the VCSP, in order to have a specific target to aim at." Now, only two years after [61], this algebraic theory has been developed, the VCSP dichotomy conjecture stated (see Conjecture 2) and proved to be equivalent to Conjecture 1 (see Theorem 57).

It is an interesting open question to find tight(er) conditions characterising tractable cases, both for the general case and for important special cases. For example, the tractability condition for finite-valued languages (see Theorem 58), is considerably tighter then the condition from Conjecture 2. It could probably be made tighter still: for $|D| = 2, 3$, tight (i.e., irreducible) descriptions are given in [25, 55], respectively.

Another interesting open question is to improve the efficiency of the general algorithm for tractable VCSPs described after Theorem 57. The current algorithm involves solving the feasibility problem for a given instance many (specifically, $O(|V| \cdot |D|)$) times, perhaps this can be improved.

In this survey, we looked only at solving VCSPs to optimality. There is plenty of literature on (in)approximability of CSP-related problems (see, e.g., the recent survey [89]), but many problems about the approximability of VCSPs are still open: for example, the open problems

from [35], stated there for $\{0,1\}$-valued CSPs, make perfect sense for arbitrary VCSPs. Note that the equivalence of maximisation and minimisation as described in Example 13 does not work when dealing with approximation properties. Fixed-parameter approximability for $\{0,1\}$-valued CSPs was considered in [9]. An interesting application of VCSP classifications to study fixed-parameter tractability appeared in [60].

In this survey, we looked at VCSPs over *finite* domains. There is a significant line of research dealing with CSPs over *infinite* domains - see [94, 7] and also [8]. The complexity of VCSPs over infinite domains is almost unexplored beyond CSPs.

The algebraic theory of the VCSP presented in Section 4 is based on the notion of a weighted clone. Not much is known about weighted clones, and this direction is wide open for purely algebraic investigation. Some specific open problems include the (possible) description of weighted clones for $D = \{0,1\}$, the identification of minimal weighted clones, and the investigation of classes of weighted clones supported by a given ordinary clone. Some partial results are presented in two MSc theses [113, 112] and in [104, 63].

In this survey we have focused on the complexity of valued constraint satisfaction problems with restricted constraint languages. It is also possible to ensure tractability by restricting the structure of the constraint scopes - so-called *structural* restrictions [46, 47, 91]. Combining structural restrictions with language restrictions leads to so-called *hybrid* restrictions, and these provide a promising source of new tractable cases [27, 28] which has so far been very little explored - see survey [29].

9 Acknowledgements

We would like to thank Pete Jeavons for his useful comments that improved the presentation of this paper.

References

1 Libor Barto. The dichotomy for conservative constraint satisfaction problems revisited. In *Proceedings of the 26th IEEE Symposium on Logic in Computer Science (LICS'11)*, pages 301–310. IEEE Computer Society, 2011.

2 Libor Barto. Constraint satisfaction problem and universal algebra. *ACM SIGLOG News*, 1(2):14–24, 2014.

3 Libor Barto. The collapse of the bounded width hierarchy. *J. Log. Comput.*, 26(3):923–943, 2016.

4 Libor Barto and Marcin Kozik. Absorbing Subalgebras, Cyclic Terms, and the Constraint Satisfaction Problem. *Logical Methods in Computer Science*, 8(1), 2012.

5 Libor Barto and Marcin Kozik. Robust satisfiability of constraint satisfaction problems. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 931–940, 2012.

6 Libor Barto and Marcin Kozik. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *Journal of the ACM*, 61(1), 2014. Article No. 3.

7 Manuel Bodirsky. Complexity classification in infinite-domain constraint satisfaction. *CoRR*, abs/1201.0856, 2012.

8 Manuel Bodirsky. Constraint satisfaction problems over numeric domains. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 79–111. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017.

9 Édouard Bonnet, László Egri, and Dániel Marx. Fixed-parameter approximability of Boolean MinCSPs. *CoRR*, abs/1601.04935, 2016.

10 Ferdinand Börner. Basics of Galois connections. In *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 38–67. Springer, 2008.

11 Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov Random Fields with Efficient Approximations. In *1998 Conference on Computer Vision and Pattern Recognition (CVPR'98)*, pages 648–655, 1998.

12 Jonah Brown-Cohen and Prasad Raghavendra. Correlation Decay & Tractability of CSPs. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP'16)*, 2016.

13 A.A. Bulatov and P.G. Jeavons. Algebraic structures in combinatorial problems. Technical Report MATH-AL-4-2001, Technische Universität Dresden, 2001.

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

15 Andrei Bulatov. Bounded relational width. Unpublished manuscript, 2009.

16 Andrei Bulatov. Graphs of relational structures: restricted colours. In *LICS'16*, 2016. To appear.

17 Andrei Bulatov and Víctor Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, 36(1):16–27, 2006.

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

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

20 Jakub Bulin, Dejan Delic, Marcel Jackson, and Todd Niven. A finer reduction of constraint problems to digraphs. *Logical Methods in Computer Science*, 11(4), 2015.

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

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

23 David A. Cohen, Martin C. Cooper, Páidí Creed, Peter Jeavons, and Stanislav Živný. An algebraic theory of complexity for discrete optimisation. *SIAM Journal on Computing*, 42(5):915–1939, 2013.

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

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

26 David A. Cohen, Martin C. Cooper, Peter G. Jeavons, Andrei A. Krokhin, Robert Powell, and Stanislav Živný. Binarisation for Valued Constraint Satisfaction Problems. Technical report, August 2016. arXiv:1608.01628.

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

28 Martin C. Cooper and Stanislav Živný. Tractable triangles and cross-free convexity in discrete optimisation. *Journal of Artificial Intelligence Research*, 44:455–490, 2012.

29 Martin C. Cooper and Stanislav Živný. Hybrid tractable classes of constraint problems. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 113–134. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017.

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

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

32 Nadaia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors. *Complexity of Constraints: An Overview of Current Research Themes*, volume 5250 of *Lecture Notes in Computer Science*. Springer, 2008.

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

34 E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The Complexity of Multiterminal Cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

35 Víctor Dalmau and Andrei Krokhin. Robust satisfiability for CSPs: Hardness and algorithmic results. *TOCT*, 5(4):15, 2013.

36 Vladimir Deineko, Peter Jonsson, Mikael Klasson, and Andrei Krokhin. The approximability of Max CSP with fixed-value constraints. *Journal of the ACM*, 55(4), 2008. Article 16.

37 Tomás Feder and Moshe Y. Vardi. 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, 1998.

38 Satoru Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 2nd edition, 2005.

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

40 Satoru Fujishige, Tamás Király, Kazuhisa Makino, Kenjiro Takazawa, and Shin-ichi Tanigawa. Minimizing submodular functions on diamonds via generalized fractional matroid matchings. Technical Report RIMS-1812, Kyoto University, 2015.

41 Satoru Fujishige and Shin-ichi Tanigawa. Polynomial combinatorial algorithms for skew-bisubmodular function minimization. Technical Report RIMS-1837, Kyoto University, 2015.

42 Satoru Fujishige, Shin-ichi Tanigawa, and Yuichi Yoshida. Generalized skew bisubmodularity: A characterization and a min-max theorem. *Discrete Optimization*, 12:1–9, 2014.

43 Peter Fulla and Stanislav Živný. A Galois Connection for Valued Constraint Languages of Infinite Size. *ACM Transactions on Computation Theory*, 8(3), 2016. Article No. 9.

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

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

46 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Tractable Optimization Problems through Hypergraph-Based Structural Restrictions. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09), Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2009.

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

48 M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

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

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

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

52 Pavol Hell and Arash Rafiey. The Dichotomy of Minimum Cost Homomorphism Problems for Digraphs. *SIAM Journal on Discrete Mathematics*, 26(4):1597–1608, 2012.

53 Hiroshi Hirai. Discrete convexity and polynomial solvability in minimum 0-extension problems. *Math. Program.*, 155(1-2):1–55, 2016.

54 Anna Huber and Vladimir Kolmogorov. Towards Minimizing k-Submodular Functions. In *Proceedings of the 2nd International Symposium on Combinatorial Optimization (ISCO'12)*, volume 7422 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 2012.

55 Anna Huber, Andrei Krokhin, and Robert Powell. Skew bisubmodularity and valued CSPs. *SIAM Journal on Computing*, 43(3):1064–1084, 2014.

56 Anna Huber and Andrei A. Krokhin. Oracle tractability of skew bisubmodular functions. *SIAM J. Discrete Math.*, 28(4):1828–1837, 2014.

57 Pawel M. Idziak, Petar Markovic, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM Journal on Computing*, 39(7):3023–3037, 2010.

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

59 Satoru Iwata and James B. Orlin. A Simple Combinatorial Algorithm for Submodular Function Minimization. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, pages 1230–1237, 2009.

60 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching and FPT algorithms. *CoRR*, abs/1310.2841, 2014.

61 Peter Jeavons, Andrei Krokhin, and Stanislav Živný. The complexity of valued constraint satisfaction. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 113:21–55, 2014.

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

63 Peter G. Jeavons, Andrius Vaicenavičius, and Stanislav Živný. Minimal weighted clones with Boolean support. In *Proceedings of the 46th IEEE International Symposium on Multiple-Valued Logic (ISMVL'16)*. IEEE, 2016.

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

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

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

67 Peter Jonsson and Gustav Nordh. Introduction to the MAXIMUM SOLUTION Problem. In *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 255–282. Springer, 2008.

68 Peter Jonsson and Johan Thapper. Approximability of the maximum solution problem for certain families of algebras. In *Proceedings of the 4th International Computer Science Symposium in Russia (CSR'09)*, volume 5675 of *Lecture Notes in Computer Science*, pages 215–226. Springer, 2009.

69 Keith Kearnes, Petar Marković, and Ralph McKenzie. Optimal strong Mal'cev conditions for omitting type 1 in locally finite varieties. *Algebra Universalis*, 72(1):91–100, 2014.

70 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2000.

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

72 Vladimir Kolmogorov, Andrei Krokhin, and Michal Rolinek. The complexity of general-valued CSPs. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1246–1258, 2015.

73 Vladimir Kolmogorov, Johan Thapper, and Stanislav Živný. The power of linear programming for general-valued CSPs. *SIAM Journal on Computing*, 44(1):1–36, 2015.

74 Vladimir Kolmogorov and Stanislav Živný. The complexity of conservative valued CSPs. *Journal of the ACM*, 60(2), 2013. Article No. 10.

75 Marcin Kozik. Weaker consistency notions for all the CSPs of bounded width. In *LICS'16*, 2016. To appear. Full version is available as arXiv:1605.00565.

76 Marcin Kozik, Andrei Krokhin, Matt Valeriote, and Ross Willard. Characterizations of several Maltsev conditions. *Algebra Universalis*, 73(3):205–224, 2015.

77 Marcin Kozik and Joanna Ochremiak. Algebraic properties of valued constraint satisfaction problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 846–858. Springer, 2015.

78 Marcin Kozik and Joanna Ochremiak. Algebraic properties of valued constraint satisfaction problem. *CoRR*, abs/1403.0476, 2015.

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

80 Andrei Krokhin and Stanislav Živný, editors. *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

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

82 Gábor Kun, Ryan O'Donnell, Suguru Tamaki, Yuichi Yoshida, and Yuan Zhou. Linear programming, width-1 CSPs, and robust satisfaction. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 484–495, 2012.

83 R.E. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, 22:155–171, 1975.

84 Benoît Larose. Algebra and the complexity of digraph CSPs: a survey. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 263–281. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017.

85 Benoit Larose and Pascal Tesson. Universal Algebra and Hardness Results for Constraint Satisfaction Problems. *Theoretical Computer Science*, 410(18):1629–1647, 2009.

86 Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra Universalis*, 56(3-4):439–466, 2007.

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

88 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065, 2015. Full version is available at arXiv:1508.04874.

89 Konstantin Makarychev and Yuri Makarychev. Approximation Algorithms for CSPs. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Com-*

plexity and Approximability, volume 7 of *Dagstuhl Follow-Ups*, pages 283–320. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017.

90 Miklós Maróti and Ralph McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3-4):463–489, 2008.

91 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM*, 60(6), 2013. Article No. 42.

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

93 Marc Mezard and Andrea Montanari. *Information, Physics, and Computation*. Oxford University Press, 2009.

94 Michael Pinsker. Algebraic and model theoretic methods in constraint satisfaction. *CoRR*, abs/1507.00931, 2015.

95 Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, pages 216–226. ACM, 1978.

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

97 Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discrete Math.*, 3(3):411–430, 1990.

98 Mark H. Siggers. A strong Mal'cev condition for locally finite varieties omitting the unary type. *Algebra Universalis*, 64(1):15–20, 2010.

99 A. Szendrei. *Clones in Universal Algebra*, volume 99 of *Seminaires de Mathematiques Superieures*. University of Montreal, 1986.

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

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

102 Walter Taylor. Varieties obeying homotopy laws. *Canadian Journal of Mathematics*, 29(3):498–527, 1977.

103 Johan Thapper and Stanislav Živný. The power of linear programming for valued CSPs. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'12)*, pages 669–678. IEEE, 2012.

104 Johan Thapper and Stanislav Živný. Necessary Conditions on Tractability of Valued Constraint Languages. *SIAM Journal on Discrete Mathematics*, 29(4):2361–2384, 2015.

105 Johan Thapper and Stanislav Živný. Sherali-Adams relaxations for valued CSPs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 1058–1069. Springer, 2015.

106 Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. *Journal of the ACM*, 63(4), 2016. Article No. 37.

107 Johan Thapper and Stanislav Živný. The limits of SDP relaxations for general-valued CSPs. Technical report, December 2016. arXiv:1612.01147.

108 Johan Thapper and Stanislav Živný. The power of Sherali-Adams relaxations for general-valued CSPs. Technical report, arXiv:1606.02577, 2016.

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

110 Hannes Uppman. The Complexity of Three-Element Min-Sol and Conservative Min-Cost-Hom. In *Proceedings of the 40th International Colloquium on Automata, Languages, and*

Programming (ICALP'13), volume 7965 of *Lecture Notes in Computer Science*, pages 804–815. Springer, 2013.

111 Hannes Uppman. Computational Complexity of the Extended Minimum Cost Homomorphism Problem on Three-Element Domains. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25, pages 651–662, 2014.

112 Andrius Vaicenavičius. A study of weighted clones. Master's thesis, Mathematical Institute, University of Oxford, 2014.

113 Jiří Vančura. Weighted Clones. Master's thesis, Department of Algebra, Charles University, 2014.

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

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

116 Stanislav Živný. *The complexity of valued constraint satisfaction problems*. Cognitive Technologies. Springer, 2012.

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