Testing the Complexity of a Valued CSP Language

Vladimir Kolmogorov
Institute of Science and Technology Austria, Klosterneuburg, Austria
vnk@ist.ac.at

Abstract

A *Valued Constraint Satisfaction Problem* (VCSP) provides a common framework that can express a wide range of discrete optimization problems. A VCSP instance is given by a finite set of variables, a finite domain of labels, and an objective function to be minimized. This function is represented as a sum of terms where each term depends on a subset of the variables. To obtain different classes of optimization problems, one can restrict all terms to come from a fixed set Γ of cost functions, called a *language*.

Recent breakthrough results have established a complete complexity classification of such classes with respect to language Γ: if all cost functions in Γ satisfy a certain algebraic condition then all Γ-instances can be solved in polynomial time, otherwise the problem is NP-hard. Unfortunately, testing this condition for a given language Γ is known to be NP-hard. We thus study exponential algorithms for this *meta-problem*. We show that the tractability condition of a finite-valued language Γ can be tested in $O(\sqrt[3]{3}^{|D|} \cdot poly(\texttt{size}(\Gamma)))$ time, where D is the domain of Γ and $poly(\cdot)$ is some fixed polynomial. We also obtain a matching lower bound under the *Strong Exponential Time Hypothesis* (SETH). More precisely, we prove that for any constant $\delta < 1$ there is no $O(\sqrt[3]{3}^{\delta|D|})$ algorithm, assuming that SETH holds.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Valued Constraint Satisfaction Problems, Exponential time algorithms, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.77

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/1803.02289`.

Funding *Vladimir Kolmogorov*: supported by the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no 616160.

1 Introduction

Minimizing functions of discrete variables represented as a sum of low-order terms is a ubiquitous problem occurring in many real-world applications. Understanding complexity of different classes of such optimization problems is thus an important task. In a prominent *VCSP framework* (which stands for *Valued Constraint Satisfaction Problem*) a class is parameterized by a set Γ of cost functions of the form $f : D^n \to \mathbb{Q} \cup \{\infty\}$ that are allowed to appear as terms in the objective. Set Γ is usually called a *language*.

Different types of languages give rise to many interesting classes. A widely studied type is *crisp* languages Γ, in which all functions f are $\{0, \infty\}$-valued. They correspond to *Constraint Satisfaction Problems (CSPs)*, whose goal is to decide whether a given instance has a feasible solution. Feder and Vardi conjectured in [11] that there exists a dichotomy for CSPs, i.e. every crisp language Γ is either tractable or NP-hard. This conjecture was refined by Bulatov, Krokhin and Jeavons [7], who proposed a specific algebraic condition

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 77; pp. 77:1–77:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that should separate tractable languages from NP-hard ones. The conjecture was verified for many special cases [27, 5, 3, 2, 8], and was finally proved in full generality by Bulatov [6] and Zhuk [31].

At the opposite end of the VCSP spectrum are the finite-valued CSPs, in which functions do not take infinite values. In such VCSPs, the feasibility aspect is trivial, and one has to deal only with the optimization issue. One polynomial-time algorithm that solves tractable finite-valued CSPs is based on the so-called basic linear programming (BLP) relaxation, and its applicability (also for the general-valued case) was fully characterized by Kolmogorov, Thapper and Živný [22]. The complexity of finite-valued CSPs was completely classified by Thapper and Živný [29], where it is shown that all finite-valued CSPs not solvable by BLP are NP-hard.

A dichotomy is also known to hold for general-valued CSPs, i.e. when cost functions in Γ are allowed to take arbitrary values in $\mathbb{Q} \cup \{\infty\}$. First, Kozik and Ochremiak showed [23] that languages that do not satisfy a certain algebraic condition are NP-hard. Kolmogorov, Krokhin and Rolínek then proved [21] that all other languages are tractable, assuming the (now established) dichotomy for crisp languages conjectured in [7].

In this paper languages Γ that satisfy the condition in [23] are called *solvable*. Since optimization problems encountered in practice often come without any guarantees, it is natural to ask what is the complexity of checking solvability of a given language Γ. We envisage that an efficient algorithm for this problem could help in theoretical investigations, and could also facilitate designing optimization approaches for tackling specific tasks.

Checking solvability of a given language is known as a *meta-problem* or a *meta-question* in the literature. Note that it can be solved in polynomial time for languages on a fixed domain D (since the solvability condition can be expressed by a linear program with $O(|D|^{|D|^m})$ variables and polynomial number of constraints, where $m = 2$ if the language is finite-valued and $m = 4$ otherwise). This naive solution, however, becomes very inefficient if D is a part of the input (which is what we assume in this paper).

The meta-problem above was studied by Thapper and Živný for finite-valued languages [29], and by Chen and Larose for crisp languages [10]. In both cases it was shown to be NP-complete. We therefore focus on exponential-time algorithms. We obtain the following results for the problem of checking solvability of a given finite-valued language Γ:

- An algorithm with complexity $O(\sqrt[3]{3}^{|D|} \cdot \mathit{poly}(\mathtt{size}(\Gamma)))$, where D is the domain of Γ and $\mathit{poly}(\cdot)$ is some fixed polynomial.
- Assuming the *Strong Exponential Time Hypothesis* (SETH), we prove that for any constant $\delta < 1$ the problem cannot be solved in $O(\sqrt[3]{3}^{\delta|D|} \cdot \mathit{poly}(\mathtt{size}(\Gamma)))$ time.

We also present a few weaker results for general-valued languages (see Section 3).

Other related work. There is a vast literature devoted to exponential-time algorithms for various problems, both on the algorithmic side and on the hardness side. Hardness results usually assume one of the following two hypotheses [15, 16, 9].

▶ **Conjecture 1** (Exponential Time Hypothesis (ETH)). *Deciding satisfiability of a* 3*-CNF-SAT formula on* n *variables cannot be solved in* $O(2^{o(n)})$ *time.*

▶ **Conjecture 2** (Strong Exponential Time Hypothesis (SETH)). *For any* $\delta < 1$ *there exists integer* k *such that deciding satisfiability of a* k*-CNF-SAT formula on* n *variables cannot be solved in* $O(2^{\delta n})$ *time.*

Below we discuss some results specific to CSPs. Let (k, d)-CSP be the class of CSP problems on a d-element domain where each constraint involves at most k variables. The number of variables in an instance will be denoted as n. A trivial exhaustive search for

a (k,d)-CSP instance runs in $O^*(d^n)$ time, where notation $O^*(\cdot)$ hides factors polynomial in the size of the input. For $(2,d)$-CSP instances the complexity can be improved to $O^*((d-1)^n)$ [26]. Some important subclasses of $(2,d)$-CSP can even be solved in $O^*(2^{O(n)})$ time. For example, [25] and [4] developed respectively $O(2.45^n)$ and $O^*(2^n)$ algorithms for solving the d-coloring problem. On the negative side, ETH is known to have the following implications:

- The $(2,d)$-CSP problem cannot be solved in $d^{o(n)} = 2^{o(n \log d)}$ time [30].
- The Graph Homomorphism problem cannot be solved in $2^{o\left(\frac{n \log d}{\log \log d}\right)}$ time [12]. (This problem can be viewed as a special case of $(2,d)$-CSP, in which a single binary relation is applied to different pairs of variables).

Recently, exponential-time algorithms for crisp NP-hard languages have been studied using algebraic techniques [17, 18, 24]. For example, [18] showed that the following conditions are equivalent, assuming the (now proved) algebraic CSP dichotomy conjecture: (a) ETH fails; (b) there exists a finite crisp NP-hard language Γ that can be solved in subexponential time (i.e. all Γ-instances on n variables can be solved in $O(2^{o(n)})$ time); (c) all finite crisp NP-hard languages Γ can be solved in subexponential time.

The rest of the paper is organized as follows: Section 2 gives a background on the VCSP framework, and Section 3 presents our results. All proofs are given in the full version of this paper [20].

2 Background

We denote $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$, where ∞ is the positive infinity. A function of the form $f : D^n \to \overline{\mathbb{Q}}$ will be called a *cost function over D of arity n*. We will always assume that the set D is finite. The *effective domain* of f is the set $\mathtt{dom}\, f = \{x \mid f(x) < \infty\}$. Note that $\mathtt{dom}\, f$ can be viewed both as an n-ary relation over D and as a function $D^n \to \{0, \infty\}$. We assume that f is represented as a list of pairs $\{(x, f(x)) \ : \ x \in \mathtt{dom}\, f\}$. Accordingly, we define $\mathtt{size}(f) = \sum_{x \in \mathtt{dom}\, f} [n \log |D| + \mathtt{size}(f(x))]$, where the size of a rational number p/q (for integers p, q) is $\log(|p|+1) + \log|q|$.

▶ **Definition 1.** *A valued constraint satisfaction language Γ over domain D is a set of cost functions $f : D^n \to \overline{\mathbb{Q}}$, where the arity n depends on f and may be different for different functions in Γ. The domain of Γ will be denoted as D_Γ. For a finite Γ we define $\mathtt{size}(\Gamma) = |D| + \sum_{f \in \Gamma} \mathtt{size}(f)$.*

A language Γ is called *finite-valued* if all functions $f \in \Gamma$ take finite (rational) values. It is called *crisp* if all functions $f \in \Gamma$ take only values in $\{0, \infty\}$. We denote $\mathrm{Feas}(\Gamma) = \{\mathtt{dom}\, f \mid f \in \Gamma\}$ to be the crisp language obtained from Γ in a natural way. Throughout the paper, for a subset $A \subseteq D$ we use u_A to denote the unary function $D \to \{0, \infty\}$ with $\arg\min u_A = A$. (Domain D should always be clear from the context). For a label $a \in D$ we also write $u_a = u_{\{a\}}$ for brevity.

▶ **Definition 2.** *An instance $\mathcal{I}$ of the valued constraint satisfaction problem (VCSP) is a function $D^V \to \overline{\mathbb{Q}}$ given by*

$$f_{\mathcal{I}}(x) \ = \ \sum_{t \in T} f_t(x_{v(t,1)}, \dots, x_{v(t,n_t)}) \tag{1}$$

It is specified by a finite set of variables V, finite set of terms T, cost functions $f_t : D^{n_t} \to \overline{\mathbb{Q}}$ of arity n_t and indices $v(t,k) \in V$ for $t \in T, k = 1, \dots, n_t$. A solution to $\mathcal{I}$ is a labeling $x \in D^V$ with the minimum total value. The size of $\mathcal{I}$ is defined as $\mathtt{size}(\mathcal{I}) = |V| + |D| + \sum_{t \in T} \mathtt{size}(f_t)$.

The instance $\mathcal{I}$ is called a Γ-instance if all terms f_t belong to Γ.

The set of all Γ-instances will be denoted as $\mathrm{VCSP}(\Gamma)$. A finite language Γ is called *tractable* if all instances $\mathcal{I} \in \mathrm{VCSP}(\Gamma)$ can be solved in polynomial time, and it is *NP-hard* if the corresponding optimization problem is NP-hard. A long sequence of works culminating with recent breakthrough papers [6, 31] has established that every finite language Γ is either tractable or NP-hard.

2.1 Polymorphisms and cores

Let $\mathcal{O}_D^{(m)}$ denote the set of all operations $g : D^m \to D$ and let $\mathcal{O}_D = \bigcup_{m \geq 1} \mathcal{O}_D^{(m)}$. When D is clear from the context, we will sometimes write simply $\mathcal{O}^{(m)}$ and $\mathcal{O}$.

Any language Γ defined on D can be associated with a set of operations on D, known as the polymorphisms of Γ, which allow one to combine (often in a useful way) several feasible assignments into a new one.

▶ **Definition 3.** *An operation $g \in \mathcal{O}_D^{(m)}$ is a* polymorphism *of a cost function $f : D^n \to \overline{\mathbb{Q}}$ if, for any $x^1, x^2, \ldots, x^m \in \mathtt{dom}\, f$, we have that $g(x^1, x^2, \ldots, x^m) \in \mathtt{dom}\, f$ where g is applied component-wise.*

For any valued constraint language Γ over a set D, we denote by $\mathrm{Pol}^{(m)}(\Gamma)$ the set of all operations on $\mathcal{O}_D^{(m)}$ which are polymorphisms of every $f \in \Gamma$. We also let $\mathrm{Pol}(\Gamma) = \bigcup_{m \geq 1} \mathrm{Pol}^{(m)}(\Gamma)$.

Clearly, if g is a polymorphism of a cost function f, then g is also a polymorphism of $\mathtt{dom}\, f$. For $\{0, \infty\}$-valued functions, which naturally correspond to relations, the notion of a polymorphism defined above coincides with the standard notion of a polymorphism for relations. Note that the projections (aka dictators), i.e. operations of the form $e_n^i(x_1, \ldots, x_n) = x_i$, are polymorphisms of all valued constraint languages. Polymorphisms play the key role in the algebraic approach to the CSP, but, for VCSPs, more general constructs are necessary, which we now define.

▶ **Definition 4.** *An m-ary* fractional operation *ω on D is a probability distribution on $\mathcal{O}_D^{(m)}$. The support of ω is defined as $\mathrm{supp}(\omega) = \{g \in \mathcal{O}_D^{(m)} \mid \omega(g) > 0\}$.*

For an operation $g \in \mathcal{O}^{(m)}$ we will denote χ_g to be characteristic vector of g, i.e. the fractional operation with $\chi_g(g) = 1$ and $\chi_g(h) = 0$ for $h \neq g$.

▶ **Definition 5.** *A m-ary fractional operation ω on D is said to be a* fractional polymorphism *of a cost function $f : D^n \to \overline{\mathbb{Q}}$ if, for any $x^1, x^2, \ldots, x^m \in \mathtt{dom}\, f$, we have*

$$\sum_{g \in \mathrm{supp}(\omega)} \omega(g) f(g(x^1, \ldots, x^m)) \leq \frac{1}{m}(f(x^1) + \ldots + f(x^m)). \tag{2}$$

For a constraint language Γ, $\mathrm{fPol}^{(m)}(\Gamma)$ will denote the set of all m-ary fractional operations that are fractional polymorphisms of each function in Γ. Also, let $\mathrm{fPol}(\Gamma) = \bigcup_{m \geq 1} \mathrm{fPol}^{(m)}(\Gamma)$, $\mathrm{supp}^{(m)}(\Gamma) = \bigcup_{\omega \in \mathrm{fPol}^{(m)}(\Gamma)} \mathrm{supp}(\omega)$ and $\mathrm{supp}(\Gamma) = \bigcup_{m \geq 1} \mathrm{supp}^{(m)}(\Gamma)$.

(It is easy to check that $\mathrm{supp}(\Gamma) \subseteq \mathrm{Pol}(\Gamma)$, and $\mathrm{supp}(\Gamma) = \mathrm{Pol}(\Gamma)$ if Γ is crisp).

Next, we will need the notion of *cores*.

▶ **Definition 6.** *Language Γ on domain D is called a* core *if all operations $g \in \mathrm{supp}^{(1)}(\Gamma)$ are bijections. Subset $B \subseteq D$ is called a* core of Γ *if $B = g(D)$ for some operation $g \in \mathrm{supp}^{(1)}(\Gamma)$ and the language $\Gamma[B]$ is a core, where $\Gamma[B]$ is the language on domain B obtained by restricting each function $f : D^n \to \overline{\mathbb{Q}}$ in Γ to B^n.*

The following facts are folklore knowledge. We do not know an explicit reference (at least in the case of general-valued languages), so we prove them in [20] for completeness.

▶ **Lemma 7.** *Let B be a subset of $D = D_\Gamma$ such that $B = g(D)$ for some $g \in \mathrm{supp}^{(1)}(\Gamma)$.*
(a) *Set B is a core of Γ if and only if $|B| = \texttt{core-size}(\Gamma) \stackrel{\text{def}}{=} \min\{|g(D)| : g \in \mathrm{supp}^{(1)}(\Gamma)\}$.*
(b) *There exists vector $\omega \in \mathrm{fPol}^{(1)}(\Gamma)$ such that $g(D) \subseteq B$ for all $g \in \mathrm{supp}(\omega)$. Furthermore, if B is a core of Γ then such ω can be chosen so that $g(a) = a$ for all $g \in \mathrm{supp}(\omega)$ and $a \in B$.*
(c) *Let $\mathcal{I}$ be a Γ-instance on variables V. Then $\min_{x \in B^V} f_{\mathcal{I}}(x) = \min_{x \in D^V} f_{\mathcal{I}}(x)$.*
For a language Γ we denote $\mathcal{B}_\Gamma^{\texttt{core}}$ to be the set of subsets $B \subseteq D$ which are cores of Γ, and $\mathcal{O}_\Gamma^{\texttt{core}}$ to be set of operations $g \in \mathrm{supp}^{(1)}(\Gamma)$ such that $g(D) \in \mathcal{B}_\Gamma^{\texttt{core}}$ (or equivalently such that $|g(D)| = \texttt{core-size}(\Gamma)$).

2.2 Dichotomy theorem

Several types of operations play a special role in the algebraic approach to (V)CSP.

▶ **Definition 8.** *An operation $g \in \mathcal{O}_D^{(m)}$ is called*
- idempotent *if $g(x, \dots, x) = x$ for all $x \in D$;*
- cyclic *if $m \geq 2$ and $g(x_1, x_2, \dots, x_m) = g(x_2, \dots, x_m, x_1)$ for all $x_1, \dots, x_m \in D$;*
- symmetric *if $m \geq 2$ and $g(x_1, x_2, \dots, x_m) = g(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(m)})$ for all $x_1, \dots, x_m \in D$, and any permutation π on $[m]$;*
- Siggers *if $m = 4$ and $g(r, a, r, e) = g(a, r, e, a)$ for all $a, e, r \in D$.*

A fractional operation ω is said to be idempotent/cyclic/symmetric if all operations in $\mathrm{supp}(\omega)$ have the corresponding property.

Note, the Siggers operation is traditionally defined in the literature as an **idempotent** operation g satisfying $g(r, a, r, e) = g(a, r, e, a)$. Here we follow the terminology in [1] that does not require idempotency. (In [10] such an operation was called *quasi-Siggers*).

We can now formulate the dichotomy theorem.

▶ **Theorem 9.** *Let Γ be a language. If the core of Γ admits a cyclic fractional polymorphism then Γ is tractable [21, 6, 31]. Otherwise Γ is NP-hard [23].*

We will call languages Γ satisfying the condition of Theorem 9 *solvable*. The following equivalent characterizations of solvability are either known or can be easily be derived from previous work [28, 19, 22, 23] (see [20]):

▶ **Lemma 10.** *Let Γ be a language and $g \in \mathrm{supp}^{(1)}(\Gamma)$. The following conditions are equivalent:*
(a) *Γ is solvable.*
(b) *Γ admits a cyclic fractional polymorphism of some arity $m \geq 2$.*
(c) *$\mathrm{supp}(\Gamma)$ contains a Siggers operation.*
(d) *$\Gamma \cup \{u_a \mid a \in B\}$ is solvable for any core B of Γ.*
(e) *$\Gamma[g(D_\Gamma)]$ is solvable.*
Furthermore, a finite-valued language Γ is solvable if and only if it admits a symmetric fractional polymorphism of arity 2.

Note that checking solvability of a given language Γ is a decidable problem. Indeed, condition (c) can be tested by solving a linear program with $|\mathcal{O}_D^{(4)}| = |D|^{|D|^4}$ variables and $O(poly(\texttt{size}(\Gamma)))$ constraints, where we maximize the total weight of Siggers operations subject to linear constraints expressing that $\omega \in \mathbb{R}^{\mathcal{O}_D^{(4)}}$ is a fractional polymorphism of Γ of arity 4.

2.3 Basic LP relaxation

Symmetric operations are known to be closely related to LP-based algorithms for CSP-related problems. One algorithm in particular has been known to solve many VCSPs to optimality. This algorithm is based on the so-called *basic LP relaxation*, or BLP, defined as follows.

Let $\mathbb{M}_n = \{\mu \geq 0 \mid \sum_{x \in D^n} \mu(x) = 1\}$ be the set of probability distributions over labelings in D^n. We also denote $\Delta = \mathbb{M}_1$; thus, Δ is the standard $(|D|-1)$-dimensional simplex. The corners of Δ can be identified with elements in D. For a distribution $\mu \in \mathbb{M}_n$ and a variable $v \in \{1, \dots, n\}$, let $\mu_{[v]} \in \Delta$ be the marginal probability of distribution μ for v:

$$\mu_{[v]}(a) \;=\; \sum_{x \in D^n : x_v = a} \mu(x) \qquad \forall a \in D.$$

Given a VCSP instance $\mathcal{I}$ in the form (1), we define the value $\mathrm{BLP}(\mathcal{I})$ as follows:

$$\begin{aligned} \mathrm{BLP}(\mathcal{I}) \;=\; & \min_{\mu,\alpha} \sum_{t \in T} \sum_{x \in \mathtt{dom}\, f_t} \mu_t(x) f_t(x) && (3)\\ \text{s.t.}\;\; (\mu_t)_{[k]} &= \alpha_{v(t,k)} && \forall t \in T, k \in \{1, \dots, n_t\}\\ \mu_t &\in \mathbb{M}_{n_t} && \forall t \in T\\ \mu_t(x) &= 0 && \forall t \in T, x \notin \mathtt{dom}\, f_t\\ \alpha_v &\in \Delta && \forall v \in V \end{aligned}$$

If there are no feasible solutions then $\mathrm{BLP}(\mathcal{I}) = \infty$. The objective function and all constraints in this system are linear, therefore this is a linear program. Its size is polynomial in $\mathtt{size}(\mathcal{I})$, so $\mathrm{BLP}(\mathcal{I})$ can be found in time polynomial in $\mathtt{size}(\mathcal{I})$.

We say that BLP *solves* $\mathcal{I}$ if $\mathrm{BLP}(\mathcal{I}) = \min_{x \in D^n} f_{\mathcal{I}}(x)$, and BLP solves $\mathrm{VCSP}(\Gamma)$ if it solves all instances $\mathcal{I}$ of $\mathrm{VCSP}(\Gamma)$. The following results are known.

▶ **Theorem 11** ([22]). *(a) BLP solves* $\mathrm{VCSP}(\Gamma)$ *if and only if* Γ *admits a symmetric fractional polymorphism of every arity* $m \geq 2$*. (b) If* Γ *is finite-valued then BLP solves* $\mathrm{VCSP}(\Gamma)$ *if and only if* Γ *admits a symmetric fractional polymorphism of arity* 2 *(i.e. if it is solvable).*

BLP relaxation also plays a key role for general-valued languages, as the following result shows. Recall that u_A for a subset $A \subseteq D$ is the unary function $D \to \{0, \infty\}$ with $\mathtt{dom}\, u_A = A$.

▶ **Definition 12.** *Consider instance* $\mathcal{I}$ *with the set of variables* V *and domain* D*. For node* $v \in V$ *denote* $D_v = \{a \in D \mid \exists x \in D^V \text{ s.t. } f_{\mathcal{I}}(x) < \infty, x_v = a\}$*. We define* $\mathrm{Feas}(\mathcal{I})$ *and* $\mathcal{I} + \mathrm{Feas}(\mathcal{I})$ *to be the instances with variables* V *and the following objective functions:*

$$f_{\mathrm{Feas}(\mathcal{I})}(x) = \sum_{v \in V} u_{D_v}(x_v) \qquad\qquad f_{\mathcal{I}+\mathrm{Feas}(\mathcal{I})}(x) = f_{\mathcal{I}}(x) + f_{\mathrm{Feas}(\mathcal{I})}(x)$$

It is easy to see that $f_{\mathcal{I}}(x) = f_{\mathcal{I}+\mathrm{Feas}(\mathcal{I})}(x)$ for any $x \in D^V$. However, the BLP relaxations of these two instances may differ.

▶ **Theorem 13** ([21]). *If* Γ *is solvable and* $\mathcal{I}$ *is a* Γ*-instance then BLP solves* $\mathcal{I} + \mathrm{Feas}(\mathcal{I})$*.*

If Γ is solvable and we know a core B of Γ, then an optimal solution for every Γ-instance can be found by using the standard self-reducibility method, in which we repeatedly add unary terms of the form $u_a(x_v)$ to the instance for different $v \in V$ and $a \in B$ and check whether this changes the optimal value of the BLP relaxation. A formal description of the method is given below. (Notations $\mathcal{I}[B]$ and $\mathcal{I} + u_a(x_v)$ should be self-explanatory; in particular, the former is the instance obtained from $\mathcal{I}$ by restricting each term to domain B).

Algorithm 1: `LP-Probe`$(\mathcal{I}, B)$. Input: instance $\mathcal{I}$ with variables V and domain D, set $B \subseteq D$. Output: either a labeling $x^* \in \arg\min_{x \in D^V} f_{\mathcal{I}}(x)$ with $x^* \in B^V$ or a flag in $\{\varnothing, \texttt{FAIL}\}$.

1 compute value $LP^* = BLP(\mathcal{I}+\text{Feas}(\mathcal{I}))$, then update $\mathcal{I} \leftarrow \mathcal{I}[B]$ (or return $\varnothing$ if $LP^* = \infty$)
2 **for** *each variable* $v \in V$ *in some order* **do**
3 **for** *each label* $a \in B$ *in some order* **do**
4 let $\mathcal{I}' = \mathcal{I} + u_a(x_v)$, and compute $LP' = BLP(\mathcal{I}' + \text{Feas}(\mathcal{I}'))$
5 if $LP' = LP^*$ then update $\mathcal{I} \leftarrow \mathcal{I}'$ and go to line 2 (i.e. proceed with the next variable v)
6 return `FAIL`
7 return labeling $x^* \in B^V$ where x^*_v equals the label a for which term $u_a(x_v)$ has been added

▶ **Lemma 14.**

(a) *If* `LP-Probe`$(\mathcal{I}, B)$ *returns a labeling* x^* *then* $x^* \in \arg\min_{x \in D^V} f_{\mathcal{I}}(x)$.

(b) *If* `LP-Probe`$(\mathcal{I}, B)$ *returns* $\varnothing$ *then instance* $\mathcal{I}$ *is infeasible.*

(c) *Suppose that* $\mathcal{I}$ *is a* Γ*-instance where* Γ *is solvable and* $B \in \mathcal{B}_\Gamma^{core}$. *Then* `LP-Probe`$(\mathcal{I}, B) \neq$ `FAIL`.

Proof. Part (a) holds by construction, and part (b) can be derived from the following two facts (which hold under the preconditions of part (b)):

- $\min_{x \in B^V} f_{\mathcal{I}}(x) = \min_{x \in D^V} f_{\mathcal{I}}(x)$ by Lemma 7(c).
- BLP solves all instances to which it is applied during the algorithm. Indeed, by Lemma 10 the language $\Gamma' = \Gamma[B] \cup \{u_a \mid a \in B\}$ is solvable. The initial instance is a Γ-instance, and all instances in line 4 are Γ'-instances. The claim now follows from Theorem 13. ◀

2.4 Meta-questions and uniform algorithms

In the light of the previous discussion, it is natural to ask the following questions about a given language Γ: (i) Is Γ solvable? (ii) Is Γ a core? (iii) What is a core of Γ? Such questions are usually called *meta-questions* or *meta-problems* in the literature. For finite-valued languages their computational complexity has been studied in [29].

▶ **Theorem 15** ([29]). *Problems (i) and (ii) for* $\{0,1\}$*-valued languages are NP-complete and co-NP-complete, respectively.*

▶ **Theorem 16** ([29]). *There is a polynomial-time algorithm that, given a core finite-valued language* Γ*, either finds a binary idempotent symmetric fractional polymorphism* ω *of* Γ *with* $|\text{supp}(\omega)| = O(poly(\texttt{size}(\Gamma)))$*, or asserts that none exists.*

For crisp languages the following hardness results are known.

▶ **Theorem 17** ([14]). *Deciding whether a given crisp language* Γ *with a single binary relation is a core is a co-NP-complete problem. (Equivalently, testing whether a given directed graph has a non-bijective homomorphism onto itself is an NP-complete problem).*

▶ **Theorem 18** ([10]). *Deciding whether a given crisp language* Γ *with binary relations is solvable is an NP-complete problem.*

It is still an open question whether an analogue of Theorem 16 holds for crisp languages, i.e. whether solvability of a given core crisp language Γ can be tested in polynomial time. However, it is known [10] that the answer would be positive assuming the existence of a certain *uniform* polynomial-time algorithm for CSPs.

▶ **Definition 19.** *Let $\mathcal{F}$ be a class of languages. A uniform polynomial-time algorithm for $\mathcal{F}$ is a polynomial-time algorithm that, for each input $(\Gamma, \mathcal{I})$ with $\Gamma \in \mathcal{F}$ and $\mathcal{I} \in \mathrm{VCSP}(\Gamma)$, computes $\min_x f_{\mathcal{I}}(x)$.*

▶ **Theorem 20** ([10])**.** *Suppose that there exists a uniform polynomial-time algorithm for the class of solvable core crisp languages. Then there exists a polynomial-time algorithm that decides whether a given core crisp language is solvable (or equivalently admits a Siggers polymorphism).*

Currently it is not known whether a uniform polynomial-time algorithm for core crisp languages exists. (Algorithms in [6, 31] assume that needed polymorphisms of the language are part of the input; furthermore, the worst-case bound on the runtime is exponential in $|D|$).

We remark that [10] considered a wider range of meta-questions for crisp languages. In particular, they studied the complexity of deciding whether a given Γ admits polymorphism $g \in O_D^{(m)}$ satisfying a given *strong linear Maltsev condition* specified by a set of linear identities. Examples of such identities are $g(x, \dots, x) \approx x$ (meaning that g is idempotent), $g(x_1, x_2, \dots, x_m) \approx g(x_2, \dots, x_m, x_1)$ (meaning that g is cyclic), and $g(r, a, r, e) \approx g(a, r, e, a)$ (meaning that g is Siggers). We refer to [10] for further details.

3 Our results

In this section the domain of language Γ is always denoted as D, and its size as $d = |D|$.

Our algorithms will construct Γ-instances $\mathcal{I}$ on $n = d^m$ variables (where $m \le 4$) with $\mathtt{size}(\mathcal{I}) = O(poly(\mathtt{size}(\Gamma)))$ for some fixed polynomial. We denote $T_{n,\Gamma}$ to be the running time of a procedure that computes $\mathrm{Feas}(\mathcal{I})$ for such $\mathcal{I}$'s. Also, let $T^*_{n,\Gamma}$ be the combined running times of computing $\mathrm{Feas}(\mathcal{J})$ for instances $\mathcal{J}$ during a call to $\mathtt{LP\text{-}Probe}(\mathcal{I}, B)$ for such $\mathcal{I}$ and some subset B. Note, if Γ is finite-valued then computing $\mathrm{Feas}(\mathcal{I})$ is a trivial problem, so $T_{n,\Gamma}$ and $T^*_{n,\Gamma}$ would be polynomial in $n + \mathtt{size}(\Gamma)$.

Conditional cores. First, we consider the problem of computing a core $B \in \mathcal{B}_\Gamma^{\mathtt{core}}$ of a given language Γ. A naive solution is to solve a linear program with $|\mathcal{O}^{(1)}| = d^d$ variables. We will present an alternative technique that runs more efficiently (in the case of finite-valued languages) but is allowed to output an incorrect result if Γ is not solvable. It will be convenient to introduce the following terminology: language Γ is a *conditional core* if either Γ is a core or Γ is not solvable. Similarly, set B is a *conditional core of* Γ if either $B \in \mathcal{B}_\Gamma^{\mathtt{core}}$ or Γ is not solvable. Note, $B = \varnothing$ is a conditional core of Γ if and only if Γ is not solvable.

To compute a conditional core of Γ, we will use the following approach. Consider a pair (Γ, σ) where σ is a string of size $O(poly(\Gamma))$ that specifies set $\mathcal{B}_\sigma$ of candidate cores of Γ. Formally, $\mathcal{B}_\sigma = \{B_1, \dots, B_N\}$ where $\varnothing \ne B_i \subseteq D$ for each $i \in [N]$. We assume that elements of $\mathcal{B}_\sigma$ can be efficiently enumerated, i.e. there exists a polynomial-time procedure for computing B_1 from σ and B_{i+1} from (σ, B_i). If $\mathcal{B}$ is a set of subsets $B \subseteq D$, we will denote

$$\begin{aligned} \mathcal{O}[\mathcal{B}] &= \{g \in \mathcal{O}^{(1)} \mid g(D) = B \text{ for some } B \in \mathcal{B}\} \\ \widehat{\mathcal{O}}[\mathcal{B}] &= \{g \in \mathcal{O}^{(1)} \mid g(D) \subseteq B \text{ for some } B \in \mathcal{B}\} \end{aligned}$$

▶ **Theorem 21.** *There exists an algorithm that for a given input* (Γ, σ) *does one of the following:*

(a) *Produces a fractional polymorphism* $\omega \in \mathrm{fPol}^{(1)}(\Gamma)$ *with* $\mathrm{supp}(\omega) \subseteq \widehat{\mathcal{O}}[\mathcal{B}_\sigma]$ *and* $|\mathrm{supp}(\omega)| \le 1 + \sum_{f\in\Gamma} |\mathtt{dom}\, f|$.

(b) *Asserts that there exists no vector* $\omega \in \mathrm{fPol}^{(1)}(\Gamma)$ *with* $\mathrm{supp}(\omega) \subseteq \widehat{\mathcal{O}}[\mathcal{B}_\sigma]$.

(c) *Asserts that one of the following holds: (i)* Γ *is not solvable; (ii)* $\mathcal{B}_\sigma \cap \mathcal{B}_\Gamma^{core} = \varnothing$.

It runs in $(|\mathcal{B}_\sigma| + O(poly(\mathtt{size}(\Gamma)))) \cdot (T^*_{d,\Gamma} + O(poly(\mathtt{size}(\Gamma))))$ *time and uses* $O(poly(\mathtt{size}(\Gamma)))$ *space.*

The algorithm in the theorem above is based on the ellipsoid method [13], which tests feasibility of a polytope using a polynomial number of calls to the separation oracle. In our case this oracle is implemented via one or more calls to $\mathtt{LP\text{-}Probe}(\mathcal{I}, B)$ for appropriate $\mathcal{I}$ and B.

One possibility would be to use Theorem 21 with the set $\mathcal{B}_\sigma = \{B \subseteq D \mid B \neq \varnothing, B \neq D\}$. If the algorithm returns result (a) then we can take operation $g \in \mathrm{supp}(\omega)$ and call the algorithm recursively for the language $\Gamma[g(D)]$ on a smaller domain. If we get result (b) or (c) then one can show that Γ is a conditional core, so we can stop. For finite-valued languages this approach would run in $O(2^d \cdot poly(\mathtt{size}(\Gamma)))$ time. We will pursue an alternative approach with an improved complexity $O(\sqrt[3]{3}^{\,d} \cdot poly(\mathtt{size}(\Gamma)))$.

This approach will use partitions $\Pi = \{D_1, \dots, D_k\}$ of domain D. For such Π we denote

$$\begin{aligned} \mathcal{O}_\Pi &= \{g \in \mathcal{O}_D^{(1)} \,:\, g(a) = g(b) \qquad \forall a, b \in A \in \Pi\} \\ \Pi^\perp &= \{B \subseteq D \,:\, |B \cap A| = 1 \qquad \forall A \in \Pi\} \end{aligned}$$

We say that Π is a *partition of* Γ if the set $\mathcal{O}_\Pi \cap \mathrm{supp}(\Gamma)$ is non-empty. In particular, the partition $\Pi = \{\{a\} \mid a \in D\}$ of D into singletons is a partition of Γ, since $\mathrm{supp}(\Gamma)$ contains the identity mapping $D \to D$. We say that Π is a *maximal partition of* Γ if Π is a partition of Γ and no coarser partition $\Pi' \succ \Pi$ (i.e. Π' with $\mathcal{O}_{\Pi'} \subset \mathcal{O}_\Pi$) has this property. Clearly, for any Γ there exists at least one Π which is a maximal partition of Γ. By analogy with cores, we say that Π is a *conditional (maximal) partition of* Γ if either Π is a (maximal) partition of Γ or Γ is not solvable.

In the results below Π is always assumed to be a partition of D.

▶ **Lemma 22.**

(a) *If* Π *is a maximal partition of* Γ *then* $\mathcal{B}_\Gamma^{core} \subseteq \Pi^\perp$ *and* $\mathcal{O}_\Gamma^{core} = \mathcal{O}[\Pi^\perp] \cap \mathrm{supp}(\Gamma)$.

(b) *If* Π *is a partition of* D *then* $|\Pi^\perp| \le \sqrt[3]{3}^{\,d}$.

▶ **Theorem 23.** *There exists an algorithm with runtime* $T_{d,\Gamma} + T_{|\Pi|,\Gamma} + O(poly(\mathtt{size}(\Gamma)))$ *that for a given input* (Γ, Π) *does one of the following:*

(a) *Asserts that* Π *is a conditional partition of* Γ.

(b) *Asserts that* Π *is not a partition of* Γ.

As before, the algorithm in Theorem 23 is based on the ellipsoid method. However, now we cannot use procedure $\mathtt{LP\text{-}Probe}(\mathcal{I}, B)$ to implement the separation oracle, since a candidate core B is not available. Instead, we solve the BLP relaxation of instance $\mathcal{I}$ and derive a separating hyperplane from a (fractional) optimal solution of the relaxation.

▶ **Corollary 24.** (1) *A conditional maximal partition* Π *of* Γ *can be computed in* $O(d^2) \cdot T_{d,\Gamma} + O(poly(\mathtt{size}(\Gamma)))$ *time.* (2) *Once such* Π *is found, a conditional core* B *of* Γ *can be computed using* $(|\Pi^\perp| + O(poly(\mathtt{size}(\Gamma)))) \cdot (T^*_{d,\Gamma} + O(poly(\mathtt{size}(\Gamma))))$ *time and* $O(poly(\mathtt{size}(\Gamma)))$ *space. If* $B \neq \varnothing$ *then the algorithm also produces a fractional polymorphism* $\omega \in \mathrm{fPol}(\Gamma)$ *such that* $\mathrm{supp}(\omega) \subseteq \mathcal{O}[\Pi^\perp]$, $|\mathrm{supp}(\omega)| \le 1 + \sum_{f\in\Gamma} |\mathtt{dom}\, f|$ *and* $\mathrm{supp}(\omega)$ *contains an operation* g *with* $g(D) = B$.

In part (1) we use a greedy search that starts with $\Pi = \{\{a\} \mid a \in D\}$ and then repeatedly calls the algorithm in Theorem 23 for coarser partitions Π. In part (2) we call the algorithm from Theorem 21 with $\sigma = \Pi$ and $\mathcal{B}_\sigma = \Pi^\perp$. For further details we refer to [20].

Testing solvability of a conditional core. Once we have found a conditional core B of Γ, we need to test whether language $\Gamma[B]$ is solvable. This problem is known to be solvable in polynomial-time for finite-valued languages [29], see Theorem 16. Their result can be extended as follows.

▶ **Theorem 25.** *There exists an algorithm that for a given language Γ does one of the following:*

(a) *Produces an idempotent fractional polymorphism $\omega \in \mathrm{fPol}(\Gamma)$ certifying solvability of Γ:*
- *ω has arity $m = 2$ and is symmetric, if Γ is finite-valued;*
- *ω has arity $m = 4$ and contains a Siggers operation in the support, if Γ is not finite-valued. Furthermore, in each case vector ω satisfies $|\operatorname{supp}(\omega)| \le 1 + \sum_{f\in\Gamma} \binom{|\mathtt{dom}\, f|}{m}$.*

(b) *Asserts that one of the following holds: (i) Γ is not solvable; (ii) Γ is not a core.*

Its runtime is $O(poly(\mathtt{size}(\Gamma)))$ if Γ is finite-valued, and $O(T^*_{d^4,\Gamma} \cdot poly(\mathtt{size}(\Gamma)))$ otherwise.

Combining procedures in Corollary 24 and the algorithm in Theorem 25 yields our main algorithmic result.

▶ **Corollary 26.** *Solvability of a given finite-valued language Γ can be tested in $O(\sqrt[3]{3}^{\,d} \cdot poly(\mathtt{size}(\Gamma)))$ time. If the answer is positive, the algorithm also returns a fractional polymorphism $\omega_1 \in \mathrm{fPol}^{(1)}(\Gamma)$ with $\operatorname{supp}(\omega_1) \subseteq \mathcal{O}_\Gamma^{\mathtt{core}}$ and a symmetric idempotent fractional polymorphism $\omega_2 \in \mathrm{fPol}^{(2)}(\Gamma[B])$ where $B = g(D)$ for some $g \in \operatorname{supp}(\omega_1)$; furthermore, $|\operatorname{supp}(\omega_m)| \le 1 + \sum_{f\in\Gamma} \binom{|\mathtt{dom}\, f|}{m}$ for $m \in \{1, 2\}$.*

Hardness results. Let us fix a constant $L \in \{1, \infty\}$. As Theorems 15, 17 and 18 state, testing whether Γ is (i) solvable and (ii) is a core are both NP-hard problems for $\{0, L\}$-valued languages. We now present additional hardness results under the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH) (see Conjectures 1 and 2). Note that for Theorem 27 we simply reuse the reductions from [10]. We say that a family of languages $\mathcal{F}$ is *k-bounded* if each $\Gamma \in \mathcal{F}$ satisfies $\mathtt{size}(\Gamma) = O(poly(d))$ for some fixed polynomial, and $\mathtt{arity}(f) \le k$ for all $f \in \Gamma$.

▶ **Theorem 27.** *Suppose that ETH holds. Then there exists a 2-bounded family $\mathcal{F}$ of $\{0, L\}$-valued languages such that the following problems cannot be solved in $O(2^{o(d)})$ time:*

(a) *Deciding whether language $\Gamma \in \mathcal{F}$ is solvable.*

(b) *Deciding whether language $\Gamma \in \mathcal{F}$ is a core.*

▶ **Theorem 28.** *Suppose that SETH holds. Then for any $\delta < 1$ there exists an $O(1)$-bounded family $\mathcal{F}$ of $\{0, L\}$-valued languages such that the following problems cannot be solved in $O(\sqrt[3]{3}^{\,\delta d})$ time:*

(a) *Deciding whether language $\Gamma \in \mathcal{F}$ is solvable (assuming the existence of a uniform polynomial-time algorithm for core crisp languages, in the case when $L = \infty$).*

(b) *Deciding whether language $\Gamma \in \mathcal{F}$ satisfies $\mathtt{core\text{-}size}(\Gamma) \le d/3$.*

References

1. L. Barto, A. Krokhin, and R. Willard. Polymorphisms, and how to use them. In A. Krokhin and S. Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*. Dagstuhl Follow-Ups series, Volume 7, 2017.
2. 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. doi:10.1109/LICS.2011.25.
3. Libor Barto, Marcin Kozik, and Todd Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). *SIAM Journal on Computing*, 38(5):1782–1802, 2009. doi:10.1137/070708093.
4. A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion–exclusion. *SIAM J. Computing*, 39(2):546–563, 2009.
5. Andrei Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006. doi:10.1145/1120582.1120584.
6. Andrei Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.
7. Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.
8. Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, 12(4), 2011. Article 24. doi:10.1145/1970398.1970400.
9. Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5917 of *LNCS*, pages 75–85, 2009.
10. Hubie Chen and Benoit Larose. Asking the Metaquestions in Constraint Tractability. *ACM Transactions on Computation Theory (TOCT)*, 9(3), October 2017. doi:10.1145/3134757.
11. 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. doi:10.1137/S0097539794266766.
12. Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Lower Bounds for the Graph Homomorphism Problem. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 9134 of *LNCS*. Springer, 2015.
13. Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. Springer, 1988.
14. P. Hell and J. Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1-3):117–126, 1992.
15. Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
16. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
17. P. Jonsson, V. Lagerkvist, G. Nordh, , and B. Zanuttini. Strong partial clones and the time complexity of SAT problems. *Journal of Computer and System Sciences*, 84:52–78, 2017.
18. Peter Jonsson, Victor Lagerkvist, and Biman Roy. Time Complexity of Constraint Satisfaction via Universal Algebra. In *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2017.
19. 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.
20. Vladimir Kolmogorov. Testing the complexity of a valued CSP language. arXiv, 2019. arXiv:1803.02289v3.
21. Vladimir Kolmogorov, Andrei Krokhin, and Michal Rolínek. The Complexity of General-Valued CSPs. *SIAM Journal on Computing (SICOMP)*, 46(3):1087–1110, 2017.

22 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.

23 Marcin Kozik and Joanna Ochremiak. Algebraic Properties of Valued Constraint Satisfaction Problem. arXiv, 2015. Extended abstract published in ICALP'15. arXiv:1403.0476.

24 Victor Lagerkvist and Magnus Wahlström. Which NP-Hard SAT and CSP Problems Admit Exponentially Improved Algorithms? arXiv, 2018. arXiv:1801.09488.

25 E.L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5(3):66–67, 1976.

26 Igor Razgon. Complexity Analysis of Heuristic CSP Search Algorithms. In *International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP)*, volume 3978 of *LNCS*, pages 88–99, 2005.

27 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. doi:10.1145/800133.804350.

28 Mark H. Siggers. A strong Mal'cev condition for locally finite varieties omitting the unary type. *Algebra Universalis*, 64(1-2):15–20, 2010.

29 Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. *Journal of the ACM (JACM)*, 63(4), 2016.

30 Patrick Traxler. The time complexity of constraint satisfaction. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 190–201, 2008.

31 Dmitriy Zhuk. A Proof of CSP Dichotomy Conjecture. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.