The complexity of approximating conservative counting CSPs*

Xi Chen[1], Martin Dyer[2], Leslie Ann Goldberg[3], Mark Jerrum[4], Pinyan Lu[5], Colin McQuillan[3], and David Richerby[3]

1 **Dept. of Comp. Sci., Columbia University, 450 Comp. Sci. Building, 1214 Amsterdam Avenue, Mailcode: 0401, New York, NY 10027-7003, USA.**
2 **School of Computing, University of Leeds, Leeds, LS2 9JT, UK.**
3 **Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK.**
4 **School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London, E1 4NS, UK.**
5 **Microsoft Research Asia, Microsoft Shanghai Technology Park, No 999, Zixing Road, Minhang District, Shanghai, 200241, China.**

Abstract

We study the complexity of approximation for a weighted counting constraint satisfaction problem #CSP($\mathcal{F}$). In the conservative case, where $\mathcal{F}$ contains all unary functions, a classification is known for the Boolean domain. We give a classification for problems with general finite domain. We define *weak log-modularity* and *weak log-supermodularity*, and show that #CSP($\mathcal{F}$) is in FP if $\mathcal{F}$ is weakly log-modular. Otherwise, it is at least as hard to approximate as #BIS, counting independent sets in bipartite graphs, which is believed to be intractable. We further sub-divide the #BIS-hard case. If $\mathcal{F}$ is weakly log-supermodular, we show that #CSP($\mathcal{F}$) is as easy as Boolean log-supermodular weighted #CSP. Otherwise, it is NP-hard to approximate. Finally, we give a trichotomy for the arity-2 case. Then, #CSP($\mathcal{F}$) is in FP, is #BIS-equivalent, or is equivalent to #SAT, the problem of approximately counting satisfying assignments of a CNF Boolean formula.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases counting constraint satisfaction problem, approximation, complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.148

1 Introduction

A weighted counting constraint satisfaction problem has a fixed finite domain D and a fixed finite "weighted constraint language" $\mathcal{F}$, a set of functions. Every $F \in \mathcal{F}$ maps a tuple of domain elements to a value called a "weight". In the computational problem #CSP($\mathcal{F}$), an instance consists of a set $V = \{v_1, \ldots, v_n\}$ of variables and a set of "weighted constraints". A weighted constraint applies a function from $\mathcal{F}$ to an appropriate-sized tuple of variables.

For example, with Boolean domain $D = \{0, 1\}$, $\mathcal{F}$ might contain a single binary (arity-2) function F defined by $F(0,0) = F(0,1) = F(1,0) = 1$ and $F(1,1) = 2$. An instance might have variables v_1, v_2, v_3 and weighted constraints $F(v_1, v_2)$, $F(v_2, v_3)$. If $\mathbf{x} = (x_1, x_2, x_3)$ is an assignment of domain elements to the variables, the total weight associated with $\mathbf{x}$ is the

* This work was supported by the EPSRC Research Grant "Computational Counting". Xi Chen was also supported by NSF Grant CCF-1139915 and Columbia University start-up funds. Full version: [11].

product of the weighted constraints, evaluated at $\mathbf{x}$. The computational problem is to evaluate the sum of weights of all assignments. In our example, this is $\sum_{\mathbf{x}\in\{0,1\}^3} F(x_1,x_2)F(x_2,x_3) = 1+1+1+2+1+1+2+4 = 13$.

There has been a lot of work on classifying the computational difficulty of exactly solving #CSP($\mathcal{F}$). For some $\mathcal{F}$, this task is computationally easy; for others it is intractable. We briefly summarise what is known: see the surveys of Chen [10] and Lu [21] for more detail.

First, suppose the domain D is Boolean (that is, $D=\{0,1\}$). Creignou and Hermann [14] showed that, when the weights also lie in $\{0,1\}$, #CSP($\mathcal{F}$) is in FP (functions computable in polynomial time) if all $F\in\mathcal{F}$ are affine but, otherwise, it is #P-complete. Dyer, Goldberg, and Jerrum [16] extended this dichotomy to non-negative rational weights, showing that the problem is polynomial-time solvable if (1) every $F\in\mathcal{F}$ is expressible as a product of unary functions, equalities and disequalities, or (2) every $F\in\mathcal{F}$ is a constant multiple of an affine function. Otherwise, the problem is complete in the complexity class $\mathrm{FP}^{\#\mathrm{P}}$. We do not deal with negative weights but the above results have been extended to these [4] and also to complex [9] weights. A dichotomy exists for the related Holant* problem [8] (see also [21]).

Next, consider arbitrary finite D. For $\{0,1\}$ weights, Bulatov's breakthrough [2] showed that #CSP($\mathcal{F}$) is always either in FP or #P-hard. This was simplified by Dyer and Richerby [18], using a new criterion called "strong balance", and extended to non-negative rational weights by Bulatov et al. [3] and to all non-negative algebraic weights by Cai, Chen and Lu [7], using a generalised notion of balance that we use here. Finally, Cai and Chen [6] extended the dichotomy to complex algebraic weights. The criteria for the above dichotomies are known to be decidable [7, 18], except for the complex case, which remains open.

Less is known about the complexity of approximation for #CSP($\mathcal{F}$). The complexity of approximate counting within #P was studied by Dyer, Goldberg, Greenhill and Jerrum [15], who identified three complexity classes for approximation problems within #P:

1. the class of problems with a fully-polynomial randomised approximation scheme (FPRAS),
2. a logically-defined complexity class called $\#\mathrm{RH}\Pi_1$, and
3. a class of problems for which approximation is NP-hard.

A typical complete problem in $\#\mathrm{RH}\Pi_1$ is #BIS, approximately counting independent sets in bipartite graphs. Either all complete problems in $\#\mathrm{RH}\Pi_1$ have an FPRAS, or none do; it is conjectured that none do [19]. #SAT, counting satisfying assignments of CNF Boolean formulas, is complete in class 3. Another important concept in the classification of approximate counting CSPs is log-supermodularity [5]. A function with Boolean domain is log-supermodular if its logarithm is supermodular; we give a formal definition later.

Over the Boolean domain, Dyer, Goldberg and Jerrum [17] gave a trichotomy for the complexity of approximately solving #CSP for $\{0,1\}$ weights. If every $F\in\mathcal{F}$ is affine, then #CSP($\mathcal{F}$) is in FP. Otherwise, it is at least as hard to approximate as #BIS. The hard approximation problems can be divided into #BIS-equivalent cases and #SAT-equivalent cases. When the domain D is Boolean, but arbitrary non-negative weights are allowed, no complete classification is known. However, Bulatov *et al.* [5] gave a classification for the so-called "conservative" case, where $\mathcal{F}$ contains all unary functions. Informally,

- if every function in $\mathcal{F}$ can be expressed in a certain simple way using disequality and unary functions, then, for any finite $\mathcal{G}\subset\mathcal{F}$, #CSP($\mathcal{G}$) has an FPRAS;
- otherwise,
  - for some finite $\mathcal{G}\subset\mathcal{F}$, #CSP($\mathcal{G}$) is at least as hard to approximate as #BIS and,
  - if $\mathcal{F}$ contains any function that is not log-supermodular, then there is a finite $\mathcal{G}\subset\mathcal{F}$ such that #CSP($\mathcal{G}$) is as hard to approximate as #SAT.

Yamakami [26] gave an approximation dichotomy when further unary functions are in $\mathcal{F}$ (including those with negative values). The negative weights cause more constraint languages to become intractable [26]. Here, we allow only non-negative weights, since more subtle complexity classifications arise.

Prior to this paper, no complexity classification was known for approximation of $\#\mathrm{CSP}(\mathcal{F})$ when the domain is not Boolean. This is the problem that we address. Our main result, Theorem 2, is a classification for the conservative case (where all unary functions are in $\mathcal{F}$). An informal description is given below. The technical concepts are defined in §1.1.

- If $\mathcal{F}$ is "weakly log-modular" then, for any finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is in FP.
- Otherwise, for some finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is at least as hard to approximate as #BIS, and
  - if $\mathcal{F}$ is "weakly log-supermodular" then, for any finite $\mathcal{G} \subset \mathcal{F}$, there is a finite set $\mathcal{G}'$ of log-supermodular functions on the Boolean domain such that $\#\mathrm{CSP}(\mathcal{G})$ is at least as easy to approximate as $\#\mathrm{CSP}(\mathcal{G}')$;
  - otherwise, $\#\mathrm{CSP}(\mathcal{G})$ is as hard to approximate as #SAT.

Our hardness results build on the approximation classification in the Boolean case [5] and on the key role played by log-supermodular functions. The easiness results use the classification of the exact evaluation of $\#\mathrm{CSP}(\mathcal{F})$ in the general case [7] and balance, in particular. The results concerning weak log-supermodularity build on key studies of the complexity of optimisation CSPs by Takhanov [24], Cohen, Cooper and Jeavons [12] and Kolmogorov and Živný [20]. We use arguments and ideas from these papers, and not merely their results. Thus, we must delve into them in some detail.

Our final result is a trichotomy for the binary case, where all $F \in \mathcal{F}$ have arity at most 2. This additionally uses work of Rudolf and Woeginger [23] on decomposing Monge matrices.

- If $\mathcal{F}$ is weakly log-modular then, for any finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is in FP.
- Otherwise, if $\mathcal{F}$ is weakly log-supermodular, then
  - for every finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is at least as easy to approximate as #BIS and
  - for some finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is at least as hard to approximate as #BIS.
- Otherwise, for some finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is as hard to approximate as #SAT.

1.1 Preliminaries and statement of results

If D is a finite domain with $|D| \geq 2$, we denote the set of functions $D^k \to R$, for some codomain R, by $\mathrm{Func}_k(D, R)$ and then $\mathrm{Func}(D, R) = \bigcup_{k=0}^{\infty} \mathrm{Func}_k(D, R)$. Let EQ be the binary equality function defined by $\mathrm{EQ}(x, x) = 1$ and $\mathrm{EQ}(x, y) = 0$ for $x \neq y$; let $\mathrm{NEQ}(x, y) = 1 - \mathrm{EQ}(x, y)$.

We use the following definitions from [5]. Let $\mathcal{F} \subseteq \mathrm{Func}(D, R)$ and let $V = \{v_1, \ldots, v_n\}$ be a set of variables. Atomic formulas have the form $\varphi = G(v_{i_1}, \ldots, v_{i_a})$ where $G \in \mathcal{F}$, $a = a(G)$ is the arity of G, and $(v_{i_1}, v_{i_2}, \ldots, v_{i_a}) \in V^a$ is the "scope". (The v_{i_j} need not be distinct.) The associated function is $F_\varphi \colon D^n \to R$, where $F_\varphi(\mathbf{x}) = G(\mathbf{x}(v_{i_1}), \ldots, \mathbf{x}(v_{i_a}))$ and $\mathbf{x} \colon V \to D$ is an assignment. To simplify notation, we write $x_j = \mathbf{x}(v_j)$ so $F_\varphi(\mathbf{x}) = G(x_{i_1}, \ldots, x_{i_a})$.

A pps-formula ("primitive product summation formula") is a sum of products of atomic formulas. A pps-formula ψ over $\mathcal{F}$ in variables $V' = \{v_1, \ldots, v_{n+k}\}$ and its associated function $F_\psi \colon D^n \to R$ are defined as follows.

$$\psi = \sum_{v_{n+1}, \ldots, v_{n+k}} \prod_{j=1}^{m} \varphi_j, \qquad F_\psi(\mathbf{x}) = \sum_{\mathbf{y} \in D^k} \prod_{j=1}^{m} F_{\varphi_j}(\mathbf{x}, \mathbf{y}) \,. \tag{1}$$

Here φ_j are atomic formulas over $\mathcal{F}$ in the variables V'. (The variables in $V = \{v_1, \ldots, v_n\}$ are *free* and those in $V' \setminus V$ are *bound*). The vectors $\mathbf{x}$ and $\mathbf{y}$ are assignments $\mathbf{x}\colon V \to D$ and $\mathbf{y}\colon V' \setminus V \to D$. The *functional clone* $\langle\mathcal{F}\rangle_\#$ *generated by* $\mathcal{F}$ is the set of functions representable by pps-formulas over $\mathcal{F} \cup \{\mathrm{EQ}\}$. Note that $\langle\langle\mathcal{F}\rangle_\#\rangle_\# = \langle\mathcal{F}\rangle_\#$ by [5, Lemma 1]. We will rely on this transitivity property implicitly.

▶ **Definition 1.**

1. A *weighted constraint language* $\mathcal{F}$ is a subset of $\mathrm{Func}(D, \mathbb{Q}_{\geq 0})$. Functions in $\mathcal{F}$ are called *weight functions*.
2. A weighted constraint language $\mathcal{F}$ is *conservative* if $\mathcal{U}_D \subseteq \mathcal{F}$, where $\mathcal{U}_D = \mathrm{Func}_1(D, \mathbb{Q}_{\geq 0})$.
3. A weighted constraint language $\mathcal{F}$ is *weakly log-modular* if, for all binary functions $F \in \langle\mathcal{F}\rangle_\#$ and all elements $a, b \in D$, $F(a,a)F(b,b) = F(a,b)F(b,a)$, or $F(a,a) = F(b,b) = 0$, or $F(a,b) = F(b,a) = 0$.
4. $\mathcal{F}$ is *weakly log-supermodular* if, for all binary functions $F \in \langle\mathcal{F}\rangle_\#$ and elements $a, b \in D$, $F(a,a)F(b,b) \geq F(a,b)F(b,a)$ or $F(a,a) = F(b,b) = 0$.
5. A function $F \in \mathrm{Func}_k(\{0,1\}, \mathbb{Q}_{\geq 0})$ is *log-supermodular* if $F(\mathbf{x} \vee \mathbf{y})F(\mathbf{x} \wedge \mathbf{y}) \geq F(\mathbf{x})F(\mathbf{y})$, for all $\mathbf{x}, \mathbf{y} \in \{0,1\}^k$, where $\wedge$ (min) and $\vee$ (max) are applied component-wise. LSM is the set of all log-supermodular functions in $\mathrm{Func}(\{0,1\}, \mathbb{Q}_{\geq 0})$. $\langle\mathsf{LSM}\rangle_\# = \mathsf{LSM}$ [5, Lemma 7].

Note that, in §4, we introduce *valued constraint languages* and *cost functions*, for optimisation CSPs. These should be distinguished from the weighted version, for counting.

The counting problem $\#\mathrm{CSP}(\mathcal{F})$ over a finite, weighted constraint language $\mathcal{F}$ is:

Instance. A pps-formula ψ, consisting of a product of m atomic $\mathcal{F}$-formulas over n free variables $\mathbf{x}$. (Thus, ψ has no bound variables.)
Output. The value $\sum_{\mathbf{x} \in D^n} F_\psi(\mathbf{x})$, where F_ψ is the function defined by ψ.

Where convenient, we write $\#\mathrm{CSP}(F)$ to mean $\#\mathrm{CSP}(\{F\})$ and write $\#\mathrm{CSP}(\mathcal{F}, \mathcal{F}')$ to mean $\#\mathrm{CSP}(\mathcal{F} \cup \mathcal{F}')$. As in [5] and other works, we take the size of a $\#\mathrm{CSP}(\mathcal{F})$ instance to be $n + m$, where n is the number of (free) variables and m is the number of weighted constraints (atomic formulas). In contrast to the unweighted case, the multiplicity of constraints matters, so we cannot bound m in terms of n. We typically denote an instance of $\#\mathrm{CSP}(\mathcal{F})$ by I and the output by $Z(I)$, which is often called the "partition function".

A counting problem, for our purposes, is any function from instances, encoded as words over a finite alphabet Σ, to $\mathbb{Q}_{\geq 0}$. A *randomised approximation scheme* for a counting problem $\#X$ is a randomised algorithm that takes an instance w and returns an approximation Y to $\#X(w)$, where a parameter $\varepsilon \in (0,1)$ specifies the error tolerance. Since the algorithm is randomised, the output Y is a random variable depending on the "coin tosses" made by the algorithm. We require that, for every instance w and every $\varepsilon \in (0,1)$,

$$\Pr\left[e^{-\varepsilon}\#X(w) \leq Y \leq e^{\varepsilon}\#X(w)\right] \geq 3/4\,.$$

The randomised approximation scheme is said to be a *fully polynomial randomised approximation scheme*, or *FPRAS*, if it runs in time bounded by a polynomial in $|w|$ (the length of the word w) and ε^{-1}. See Mitzenmacher and Upfal [22, Definition 10.2].

If $\#X$ and $\#Y$ are counting problems, an "approximation-preserving reduction" [15] (AP-reduction) turns an FPRAS for $\#Y$ into an FPRAS for $\#X$. Specifically, an *AP-reduction from* $\#X$ *to* $\#Y$ is a randomised algorithm $\mathcal{A}$ for computing $\#X$ using an oracle for $\#Y$. $\mathcal{A}$'s input is a pair $(w, \varepsilon) \in \Sigma^* \times (0,1)$, and: (i) oracle calls made by $\mathcal{A}$ are of the form (v, δ), where $v \in \Sigma^*$ is an instance of $\#Y$, and $0 < \delta < 1$ is an error bound with $\delta^{-1} \leq \mathrm{poly}(|w|, \varepsilon^{-1})$;

(ii) $\mathcal{A}$ is a randomised approximation scheme for $\#X$ whenever the oracle is a randomised approximation scheme for $\#Y$; and (iii) the run-time of $\mathcal{A}$ is polynomial in $|w|$ and ε^{-1}. If an AP-reduction from $\#X$ to $\#Y$ exists, we write $\#X \leq_{\mathrm{AP}} \#Y$. A counting problem $\#X$ is $\#Y$-easy if $\#X \leq_{\mathrm{AP}} \#Y$ and it is $\#Y$-hard if $\#Y \leq_{\mathrm{AP}} \#X$. A problem $\#X$ is LSM-*easy* if there is a finite, weighted constraint language $\mathcal{F} \subset \mathsf{LSM}$ such that $\#X \leq_{\mathrm{AP}} \#\mathrm{CSP}(\mathcal{F})$.

The notion of pps-definability is closely related to AP-reductions. In particular, [5, Lemma 17] shows that $G \in \langle\mathcal{F}\rangle_{\#}$ implies that $\#\mathrm{CSP}(\mathcal{F}, G) \leq_{\mathrm{AP}} \#\mathrm{CSP}(\mathcal{F})$. We will use this fact without comment. Note that, subsequent to [15], the notation $\leq_{\mathrm{AP}}$ has been used for a different approximation-preserving reduction which applies to optimisation problems. Since our emphasis is on counting problems, this should not cause confusion.

We now state our main theorem. Note that we have only defined $\#\mathrm{CSP}(\mathcal{F})$ for finite $\mathcal{F}$.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$.*

1. *If $\mathcal{F}$ is weakly log-modular then $\#\mathrm{CSP}(\mathcal{G})$ is in* FP *for every finite $\mathcal{G} \subset \mathcal{F}$.*
2. *If $\mathcal{F}$ is weakly log-supermodular but not weakly log-modular, then $\#\mathrm{CSP}(\mathcal{G})$ is* LSM*-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and* #BIS*-hard for some such $\mathcal{G}$.*
3. *If $\mathcal{F}$ is weakly log-supermodular but not weakly log-modular and consists of functions of arity at most two, then $\#\mathrm{CSP}(\mathcal{G})$ is* #BIS*-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and* #BIS*-equivalent for some such $\mathcal{G}$.*
4. *If $\mathcal{F}$ is not weakly log-supermodular, then $\#\mathrm{CSP}(\mathcal{G})$ is* #SAT*-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and* #SAT*-equivalent for some such $\mathcal{G}$.*

In particular, among conservative #CSPs, there are no new complexity classes below #BIS or above LSM; furthermore there is a trichotomy for conservative weighted constraint languages with no functions of arity greater than two.

#BIS-hardness and relationships with #SAT are proved in §2, restated as Theorem 4. Membership in FP is Theorem 7 in §3. LSM-easiness and #BIS-easiness are established by Theorem 29 in §6. Further, there is an algorithm that determines which case of Theorem 2 holds for $\mathcal{H} \cup \mathcal{U}_D$ where $\mathcal{H}$ is finite, as shown in §7. Full proofs are in [11].

2 Hardness results

Our hardness results use the following special case of [5, Theorem 18]. That result is expressed in terms of efficiently computable reals and we restrict to rationals for simplicity. The statement of [5, Theorem 18] does not imply our lemma but the proof does.

▶ **Lemma 3.** *Let F be a function in $\mathrm{Func}_2(\{0,1\}, \mathbb{Q}_{\geq 0})$.*

- *If $F \notin \langle \mathrm{NEQ}, \mathcal{U}_{\{0,1\}} \rangle_{\#}$, then $\#\mathrm{CSP}(\mathcal{G})$ is* #BIS*-hard for some finite $\mathcal{G} \subset \{F\} \cup \mathcal{U}_{\{0,1\}}$.*
- *If $F \notin \langle \mathrm{NEQ}, \mathcal{U}_{\{0,1\}} \rangle_{\#} \cup \mathsf{LSM}$, then $\#\mathrm{CSP}(\mathcal{G})$ is* #SAT*-hard for some finite $\mathcal{G} \subset \{F\} \cup \mathcal{U}_{\{0,1\}}$.*

▶ **Theorem 4.** *Let $\mathcal{F}$ be a conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$.*

- *If $\mathcal{F}$ is not weakly log-modular, $\#\mathrm{CSP}(\mathcal{G})$ is* #BIS*-hard for some finite $\mathcal{G} \subset \mathcal{F}$.*
- *If $\mathcal{F}$ is not weakly log-supermodular, $\#\mathrm{CSP}(\mathcal{G})$ is* #SAT*-hard for some finite $\mathcal{G} \subset \mathcal{F}$.*
- *For all finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is* #SAT*-easy.*

Proof (sketch). Functions in $\langle \mathrm{NEQ}, \mathcal{U}_{\{0,1\}} \rangle_{\#}$ can have only certain forms. If $\mathcal{F}$ is not weakly log-modular, we construct $F \in \langle\mathcal{F}\rangle_{\#}$ that does not have any of these forms and #BIS-hardness follows from Lemma 3. The proof of #SAT-hardness is similar. #SAT-easiness follows from the reduction to the unweighted case [3], which is #SAT-easy [15]. ◀

3 Balance and weak log-modularity

In this section we use the notion of balance to show that weak log-modularity implies tractability. We may associate a matrix $\mathbf{M}$ with an undirected bipartite graph $G_{\mathbf{M}}$ whose vertex partition consists of the set of rows R and columns C of $\mathbf{M}$. A pair $(r, c) \in R \times C$ is an edge of $G_{\mathbf{M}}$ if, and only if, $\mathbf{M}_{rc} \neq 0$. A *block* of $\mathbf{M}$ is a submatrix whose rows and columns form a connected component in $G_{\mathbf{M}}$. $\mathbf{M}$ has block-rank 1 if all its blocks have rank 1.

A weighted constraint language $\mathcal{F}$ is *balanced* [7,18] if, for every function $F(x_1, \ldots, x_n) \in \langle\mathcal{F}\rangle_{\#}$ with arity $n \geq 2$, and every k with $0 < k < n$, the $|D|^k \times |D|^{n-k}$ matrix $F((x_1, \ldots, x_k), (x_{k+1}, \ldots, x_n))$ has block-rank 1.

A function $F\colon \{0,1\}^n \to \mathbb{R}$ has *rank 1* if it has the form $F(x_1, \ldots, x_k) = U_1(x_1) \cdots U_k(x_k)$. A function $F\colon D^n \to \mathbb{Q}_{\geq 0}$ is *essentially pseudo-Boolean* if its support is a subset of $D_1 \times \cdots \times D_n$ with $|D_1|, \ldots, |D_n| \leq 2$. The *projection* of a relation $R \subseteq D^n$ onto indices $1 \leq i < j \leq n$ is the set of $(a, b) \in D^2$ such that there exists $\mathbf{x} \in R$ with $x_i = a$ and $x_j = b$. A *generalised* NEQ is a relation $\{(x_i, x_j), (y_i, y_j)\} \subset D^2$ for some $x_i \neq y_i$ and $x_j \neq y_j$.

▶ **Lemma 5.** *Let $F\colon D^n \to \mathbb{Q}_{\geq 0}$ be an essentially pseudo-Boolean function which is not of rank 1. If $\{F\} \cup \mathcal{U}_D$ is weakly log-modular, then some binary projection of the support of F is a generalised* NEQ.

▶ **Lemma 6.** *Every conservative weakly log-modular weighted constraint language is balanced.*

Proof (sketch). If $\mathcal{F}$ is not balanced, there is a function $F \in \langle\mathcal{F}\rangle_{\#}$ and a partition of its variables for which the corresponding matrix $\mathbf{M}$ is not block-rank-1. We show that $\mathbf{M}$ has a 2×2 submatrix that is not block-rank-1 and use this to construct an essentially pseudo-Boolean function G that is not of rank 1 and none of whose binary projections is a generalised NEQ. Now, use Lemma 5. ◀

This, along with the dichotomy of Cai, Chen and Lu [7] for the complexity of exact evaluation of #CSP, gives us the tractable case of Theorem 2, since balanced problems can be solved exactly, in polynomial time.

▶ **Theorem 7.** *Let $\mathcal{F}$ be a conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$. If $\mathcal{F}$ is weakly log-modular then, for any finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G}) \in \mathrm{FP}$.*

4 Valued clones, valued CSPs and relational clones

To define valued clones, we use the same set-up as §1.1 except that summation is replaced by minimisation and product is replaced by sum. Let D be a finite domain with $|D| \geq 2$ and let R be a codomain with $\{0, \infty\} \subseteq R$, where ∞ obeys the following rules for all $x \in R$: $x + \infty = \infty$, $x \leq \infty$ and $\min\{x, \infty\} = x$. Let Φ be a subset of $\mathrm{Func}(D, R)$. Let $V = \{v_1, \ldots, v_n\}$ be a set of variables. For each atomic formula $\varphi = G(v_{i_1}, \ldots, v_{i_a})$ we use the notation f_φ to denote the function represented by φ, so $f_\varphi(\mathbf{x}) = G(x_{i_1}, \ldots, x_{i_a})$.

A psm-formula ("primitive sum minimisation formula") is a minimisation of a sum of atomic formulas. A psm-formula ψ over Φ in variables $V' = \{v_1, \ldots, v_{n+k}\}$, and its associated function f_ψ are defined, analogously to (1), by

$$\psi = \min_{v_{n+1}, \ldots, v_{n+k}} \sum_{j=1}^{m} \varphi_j, \qquad f_\psi(\mathbf{x}) = \min_{\mathbf{y} \in D^k} \sum_{j=1}^{m} f_{\varphi_j}(\mathbf{x}, \mathbf{y}),$$

where the φ_j are atomic formulas and $\mathbf{x}\colon \{v_1, \ldots, v_n\} \to D$, $\mathbf{y}\colon \{v_{n+1}, \ldots, v_{n+k}\} \to D$ are assignments.

The *valued clone* $\langle\Phi\rangle_V$ *generated by* Φ is the set of all functions that can be represented by a psm-formula over $\Phi \cup \{\text{eq}\}$, where eq is the binary equality function on D given by $\text{eq}(x,x) = 0$ and $\text{eq}(x,y) = \infty$ for $x \neq y$.

We next introduce valued constraint satisfaction problems (VCSPs), which are optimisation problems. In the work of Kolmogorov and Živný [20], the codomain is $R = \mathbb{Q}_{\geq 0} \cup \{\infty\}$. It will be useful to extend the codomain to include irrational numbers. This causes no problems since, apart from Theorem 23, we use only calculations from their papers, not complexity results. For Theorem 23, we avoid irrational numbers and, in fact, restrict to cost functions taking values in $\{0, \infty\} \subset R$. Furthermore, all the real numbers we use are either rationals, or their logarithms, so are efficiently computable.

Let $\overline{\mathbb{R}}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$ be the set of non-negative real numbers together with ∞.

▶ **Definition 8.** A *cost function* is a function $D^k \to \overline{\mathbb{R}}_{\geq 0}$. A *valued constraint language* is a set of cost functions $\Phi \subseteq \text{Func}(D, \overline{\mathbb{R}}_{\geq 0})$.

For a valued constraint language Φ, a problem in VCSP(Φ) has instance ψ, a psm-formula which is a sum of m atomic Φ-formulas in n free variables $\mathbf{x}$, and computes the value

$$\text{minCost}(\psi) = \min_{\mathbf{x} \in D^n} f_\psi(\mathbf{x}), \quad \text{where } f_\psi \text{ is the function defined by } \psi.$$

We typically use the notation of Kolmogorov and Živný. An instance is usually denoted by the letter I. In this case, we use f_I to denote the function specified by the psm-formula corresponding to instance I, so the value of the instance is denoted by $\text{minCost}(I)$. The psm-formula corresponding to I is a sum of atomic formulas (since all of the variables are free variables). We refer to each of these atomic formulas as a *valued constraint* and we represent these by the multiset T of all valued constraints in the instance I. For each valued constraint $t \in T$ we use k_t to denote its arity, f_t to denote the function represented by the corresponding atomic formula, and σ_t to denote its scope, which is given as a tuple $(i(t,1), \ldots, i(t,k_t)) \in \{1, \ldots, n\}^{k_t}$ containing the indices of the variables in the scope. Thus,

$$f_I(\mathbf{x}) = \sum_{t \in T} f_t(x_{i(t,1)}, \ldots, x_{i(t,k_t)}). \tag{2}$$

We will use $\mathbf{x}[\sigma_t]$ as an abbreviation for the tuple $(x_{i(t,1)}, \ldots, x_{i(t,k_t)})$. In this abbreviated notation, the function defined by instance I may be written as $f_I(\mathbf{x}) = \sum_{t \in T} f_t(\mathbf{x}[\sigma_t])$.

Now, let $[0,1]_{\mathbb{Q}} = [0,1] \cap \mathbb{Q}$. For reasons which will be clear below, it will be useful to work with weight functions in $\text{Func}(D, [0,1]_{\mathbb{Q}})$. For such a weight function F, let the cost function $\ell(F) \in \text{Func}(D, \overline{\mathbb{R}}_{\geq 0})$ be the function defined by

$$(\ell(F))(\mathbf{x}) = \begin{cases} -\ln F(\mathbf{x}) & \text{if } F(\mathbf{x}) > 0 \\ \infty & \text{if } F(\mathbf{x}) = 0. \end{cases}$$

For example, $\ell(\text{EQ}) = \text{eq}$. For a weighted constraint language $\mathcal{F} \subseteq \text{Func}(D, [0,1]_{\mathbb{Q}})$, let $\ell(\mathcal{F})$ be the valued constraint language defined by $\ell(\mathcal{F}) = \{\ell(F) \mid F \in \mathcal{F}\}$.

There is a bijection between instances of #CSP($\mathcal{F}$) and VCSP($\ell(\mathcal{F})$), given by replacing each function F_t in the former by the function $f_t = \ell(F_t)$ in the latter, with scopes unchanged. Note that $f_I(\mathbf{x}) = -\ln F_I(\mathbf{x})$, for any assignment $\mathbf{x}$, with the convention $-\ln 0 = \infty$.

▶ **Definition 9.** A valued constraint language is *conservative* if it contains all arity-1 cost functions $D \to \overline{\mathbb{R}}_{\geq 0}$.

The mapping $F \mapsto \ell(F)$ from $\text{Func}(D, [0,1]_{\mathbb{Q}})$ to $\text{Func}(D, \overline{\mathbb{R}}_{\geq 0})$ is not surjective since not all real numbers are logarithms of rationals. Similarly, for any weighted constraint language

$\mathcal{F}$, the valued constraint language $\ell(\mathcal{F})$ is not conservative. Finally, note that we define $\ell(F)$ only for $F \in \mathrm{Func}(D, [0,1]_{\mathbb{Q}})$. The extension to $F \in \mathrm{Func}(D, \mathbb{Q}_{\geq 0})$ produces negative-valued cost functions. We wish to avoid this since Kolmogorov and Živný [20] do not allow it.

▶ **Definition 10.** A cost function is *crisp* [13] if $f(\mathbf{x}) \in \{0, \infty\}$ for all $\mathbf{x}$.

▶ **Definition 11.** For a cost function f, let $\mathrm{Feas}(f)$ be the relation $\mathrm{Feas}(f) = \{\mathbf{x} \mid f(\mathbf{x}) < \infty\}$.

Thus, any cost function f can be associated with its underlying relation. Similarly, we can represent any relation by a crisp cost function f for which $f(\mathbf{x}) = 0$ if and only if $\mathbf{x}$ is in the relation. A *crisp constraint language* is a set of relations, which we always represent as crisp cost functions, not as functions with codomain $\{0, 1\}$. For a valued constraint language Φ, the crisp constraint language $\mathrm{Feas}(\Phi)$ is given by $\mathrm{Feas}(\Phi) = \{\mathrm{Feas}(f) \mid f \in \Phi\}$. A *relational clone* is simply a crisp constraint language $\mathrm{Feas}(\langle\Phi\rangle_V)$ for a valued constraint language Φ.

▶ **Lemma 12.** *Suppose* $\Phi \subseteq \mathrm{Func}(D, \overline{\mathbb{R}}_{\geq 0})$. *Then* $\langle \mathrm{Feas}(\Phi)\rangle_V = \mathrm{Feas}(\langle\Phi\rangle_V)$.

Proof. The mapping $\rho\colon \overline{\mathbb{R}}_{\geq 0} \to \{0, \infty\}$ defined by $\rho(\infty) = \infty$ and $\rho(x) = 0$, for all $x < \infty$, is a homomorphism of semirings, from $(\overline{\mathbb{R}}_{\geq 0}, \min, +)$ to $(\{0, \infty\}, \min, +)$. ◀

▶ **Definition 13.** A crisp constraint language is *conservative* if it includes all arity-1 relations.

5 STP/MJN multimorphisms and weak log-supermodularity

In [20, Corollary 12], Kolmogorov and Živný showed that the VCSP associated with a conservative valued constraint language Φ is tractable iff Φ has an STP/MJN multimorphism.

We define STP/MJN multimorphisms below. In this section, we show that, if a weighted constraint language $\mathcal{F} \in \mathrm{Func}(D, [0,1]_{\mathbb{Q}})$ is weakly log-supermodular, the corresponding valued constraint language $\ell(\mathcal{F})$ has an STP/MJN multimorphism. In §6, this will enable us to use such a multimorphism (via the work of Kolmogorov and Živný [20] and Cohen, Cooper and Jeavons [12]) to prove #BIS-easiness and LSM-easiness of the weighted counting CSP.

Our proofs rely on work of Kolmogorov and Živný [20] and Takhanov [24]. We start with some general definitions. Note that some of these definitions in [20] differ from those in [12].

▶ **Definition 14.**

1. A *k-ary operation on D* is a function from D^k to D. An *operation on D* is a k-ary operation, for some k. We omit "on D" when the domain D is clear from the context.
2. A k-tuple $\langle \rho_1, \ldots, \rho_k\rangle$ of k-ary operations $\rho_1, \ldots, \rho_k$ is *conservative* if the multisets $\{\{x_1, \ldots, x_k\}\}$ and $\{\{\rho_1(\mathbf{x}), \ldots, \rho_k(\mathbf{x})\}\}$ are equal for all $\mathbf{x} = (x_1, \ldots, x_k) \in D^k$.
3. $\langle \rho_1, \ldots, \rho_k\rangle$ is a *multimorphism* of an arity-r cost function f if:

$$\sum_{i=1}^{k} f(\rho_i(x_1^1, \ldots, x_1^k), \ldots, \rho_i(x_r^1, \ldots, x_r^k)) \leq \sum_{i=1}^{k} f(\mathbf{x}^i) \quad \text{for all } \mathbf{x}^1, \ldots, \mathbf{x}^k \in D^r.$$

4. $\langle \rho_1, \ldots, \rho_k\rangle$ is a multimorphism of a valued constraint language Φ if it is a multimorphism of every $f \in \Phi$.

Note that we have defined "conservative" for operations and constraint languages (weighted, valued and crisp). These notions are connected, but we do not need that here.

▶ **Observation 15.** If $\langle \rho_1, \ldots, \rho_k\rangle$ is conservative, it is a multimorphism of every unary f.

▶ **Definition 16.**

1. Suppose $M \subseteq D^2$. A pair $\langle \sqcap, \sqcup \rangle$ of binary operations is a *symmetric tournament pair* (STP) on M if it is conservative and both operations are commutative on M. We say that it is an STP if it is an STP on D^2.
2. Suppose $M \subseteq D^2$. A triple $\langle \mathtt{Mj1}, \mathtt{Mj2}, \mathtt{Mn3} \rangle$ of ternary operations is an *MJN on* M if it is conservative and, for all triples $(a, b, c) \in D^3$ with $\{\{a, b, c\}\} = \{\{x, x, y\}\}$ where $x \neq y$ and $(x, y) \in M$, we have $\mathtt{Mj1}(a, b, c) = \mathtt{Mj2}(a, b, c) = x$ and $\mathtt{Mn3}(a, b, c) = y$.
3. An *STP/MJN multimorphism* of a valued constraint language Φ consists of a pair of operations $\langle \sqcap, \sqcup \rangle$ and a triple of operations $\langle \mathtt{Mj1}, \mathtt{Mj2}, \mathtt{Mn3} \rangle$, both of which are multimorphisms of Φ, for which, for some symmetric subset M of D^2, $\langle \sqcap, \sqcup \rangle$ is an STP on M and $\langle \mathtt{Mj1}, \mathtt{Mj2}, \mathtt{Mn3} \rangle$ is an MJN on $\{(a, b) \in D^2 \setminus M \mid a \neq b\}$.
4. $\Phi \subseteq \mathrm{Func}(D, \overline{\mathbb{R}}_{\geq 0})$ is *weakly submodular* if, for all binary $f \in \langle \Phi \rangle_V$ and $a, b \in D$, $f(a, a) + f(b, b) \leq f(a, b) + f(b, a)$ or $f(a, a) = f(b, b) = \infty$.

Our definition of weak submodularity for cost functions restates Kolmogorov and Živný's "Assumption 3". It is nontrivial that weak log-supermodularity for $\mathcal{F}$ is related to weak submodularity for $\ell(\mathcal{F})$. In particular, we cannot expect $\langle \ell(\mathcal{F}) \rangle_V = \ell(\langle \mathcal{F} \rangle_\#)$ to hold in general. However, the following is suitable for our purposes.

▶ **Lemma 17.** *Suppose $\mathcal{F} \subseteq \mathrm{Func}(D, [0, 1]_\mathbb{Q})$ and let $\Phi = \ell(\mathcal{F})$. If $\mathcal{F}$ is weakly log-supermodular then Φ is weakly submodular.*

Proof (sketch). For the contrapositive, take a binary $f \in \langle \Phi \rangle_V$ that is not weakly submodular and let $F^{(k)} \in \langle \mathcal{F} \rangle_\#$ be the function defined by replacing every atomic formula $g(\mathbf{x}, \mathbf{y})$ in the definition of f with $G(\mathbf{x}, \mathbf{y})^k$, where $g = \ell(G)$. For k large enough, $F^{(k)}(\mathbf{x})^{1/k}$ is close enough to the maximum value taken by any $G(\mathbf{x}, \mathbf{y})$ in the definition of F. But this maximum value is $\exp(-f(\mathbf{x}))$ and we can now show that $F^{(k)}$ is not weakly log-supermodular. ◀

Let Γ be a crisp constraint language. A *majority polymorphism* of Γ is a ternary operation ρ such that $\rho(a, a, b) = \rho(a, b, a) = \rho(b, a, a) = a$ for all $a, b \in D$, and for all k-ary relations $R \in \Gamma$, $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$ implies that $(\rho(x_1, y_1, z_1), \ldots, \rho(x_k, y_k, z_k)) \in R$.

The formulation of the following theorem is essentially [24, Theorem 9.1] (but note that Takhanov uses the term "functional clone" in a different way to us).

▶ **Theorem 18** (Takhanov). *Let $N(a, b, c, d)$ be the relation $\{(a, c), (b, c), (a, d)\}$, and let Γ be a conservative relational clone with domain D. At least one of the following holds.*

- *There are distinct $a, b \in D$ such that $N(a, b, a, b) \in \Gamma$.*
- *There are distinct $a, b \in D$ such that $\{(a, a, a), (a, b, b), (b, a, b), (b, b, a)\} \in \Gamma$.*
- *For some $k \geq 1$, there are $a_0, \ldots, a_{2k}, b_0, \ldots, b_{2k} \in D$ such that, for each $0 \leq i \leq 2k$, $a_i \neq b_i$ and, for each $0 \leq i \leq 2k - 1$, $N(a_i, b_i, a_{i+1}, b_{i+1}) \in \Gamma$ and $N(a_{2k}, b_{2k}, a_0, b_0) \in \Gamma$.*
- *Γ has a majority polymorphism.*

To prove the following lemma, we show that the first three bullets of Takhanov's theorem cannot hold, so the fourth must.

▶ **Lemma 19.** *If $\Phi \subseteq \mathrm{Func}(D, \overline{\mathbb{R}}_{\geq 0})$ is conservative and weakly submodular then $\Gamma = \langle \mathrm{Feas}(\Phi) \rangle_V$ has a majority polymorphism.*

The main theorem of this section now follows from Lemmas 17 and 19. The final construction of the STP/MJN multimorphism comes from [20, §§6.1–6.4].

▶ **Theorem 20.** *Let $\mathcal{F}$ be a weighted constraint language with $\mathrm{Func}_1(D, [0, 1]_\mathbb{Q}) \subseteq \mathcal{F} \subseteq \mathrm{Func}(D, [0, 1]_\mathbb{Q})$. If $\mathcal{F}$ is weakly log-supermodular then $\ell(\mathcal{F})$ has an STP/MJN multimorphism.*

6 LSM-easiness and #BIS-easiness

Our aim is to show that $\mathcal{F}$ is LSM-easy if $\ell(\mathcal{F})$ has an STP/MJN multimorphism. This will use arguments from [12] and [20], but we try, as far as possible, to avoid going into the details of their proofs. We start by generalising the notion of an STP multimorphism.

▶ **Definition 21.** Let f be an arity-k cost function. A *generalised STP multimorphism of f* is a pair $\langle\sqcap,\sqcup\rangle$, defined as follows. For $1 \le i \le k$, $\sqcap_i$ and $\sqcup_i$ are operations on the set $D_i = \{a \in D \mid \exists \mathbf{x} : x_i = a \text{ and } f(\mathbf{x}) < \infty\}$, and $\langle\sqcap_i,\sqcup_i\rangle$ is an STP of $\{f\}$.

The operation $\sqcap$ is the binary operation on $D_1 \times \cdots \times D_k$ defined by applying $\sqcap_1,\ldots,\sqcap_k$ component-wise. Similarly, $\sqcup$ is defined by applying $\sqcup_1,\ldots,\sqcup_k$ component-wise. We require that, for all $\mathbf{x},\mathbf{y} \in D^k$, $f(\sqcup(\mathbf{x},\mathbf{y})) + f(\sqcap(\mathbf{x},\mathbf{y})) \le f(\mathbf{x}) + f(\mathbf{y})$. Equivalently, we require that $f(\sqcup_1(x_1,y_1),\ldots,\sqcup_k(x_k,y_k)) + f(\sqcap_1(x_1,y_1),\ldots,\sqcap_k(x_k,y_k)) \le f(\mathbf{x}) + f(\mathbf{y})$.

Theorem 22 is closely related to [20, Theorem 11]. Its proof is the same but we stop when [20, Lemma 35] has been proved. We also need an algorithmic consequence.

▶ **Theorem 22** (Kolmogorov and Živný). *Suppose Φ_0 is a finite, valued constraint language which has an STP/MJN multimorphism. Then there is a polynomial-time algorithm that takes an instance I of* VCSP(Φ_0) *and returns a generalised STP multimorphism $\langle\sqcap,\sqcup\rangle$ of f_I. The pair $\langle\sqcap,\sqcup\rangle$ depends only on the STP/MJN multimorphism of Φ_0 and on the relation* Feas(f_I) *underlying f_I. It does not depend in any other way on I.*

▶ **Theorem 23** (Kolmogorov and Živný). *If Φ_0 is a finite, crisp constraint language that has an STP/MJN multimorphism, then there is a polynomial-time algorithm for* VCSP(Φ_0).

For our eventual construction, we would like $\langle\sqcap,\sqcup\rangle$ to induce a generalised STP multimorphism of f_t for each individual valued constraint t in the instance. We do not know whether this is true of the generalised STP multimorphism provided by Kolmogorov and Živný's algorithm, but something sufficiently close to this is true.

▶ **Definition 24.** For an instance I, a valued constraint t and a length-k_t vector $\mathbf{a}$, define $R_{I,t}(\mathbf{a}) = 0$ if there exists $\mathbf{x}$ with $\mathbf{x}[\sigma_t] = \mathbf{a}$ and $f_I(\mathbf{x}) < \infty$ and $R_{I,t}(\mathbf{a}) = \infty$, otherwise. Define $f'_t = f_t + R_{I,t}$. Thus, f'_t is a "trimmed" version of f_t whose domain is precisely the k_t-tuples of values that can actually arise in feasible solutions to instance I.

We will see that, if the scope σ_t contains variables with indices $i(t,1),\ldots,i(t,k_t)$, then $\langle\sqcap[\sigma_t],\sqcup[\sigma_t]\rangle = \langle(\sqcap_{i(t,1)},\ldots,\sqcap_{i(t,k_t)}),(\sqcup_{i(t,1)},\ldots,\sqcup_{i(t,k_t)})\rangle$ is a generalised STP multimorphism of f'_t, even though it might not necessarily be a generalised STP multimorphism of f_t. Note that Theorem 23 has the following consequence.

▶ **Corollary 25.** *Let Φ_0 be a finite, valued constraint language that has an STP/MJN multimorphism. There is a polynomial-time algorithm that takes an instance I of* VCSP(Φ_0), *a valued constraint t and a length-k_t vector $\mathbf{a}$ and returns a (finite) truth table for f'_t.*

▶ **Theorem 26** (Extends Theorem 22). *Suppose Φ_0 is a finite, valued constraint language which has an STP/MJN multimorphism. Consider the algorithm from Theorem 22 which takes an instance I of* VCSP(Φ_0) *(in the form (2)) and returns a generalised STP multimorphism $\langle\sqcap,\sqcup\rangle$ of f_I. Then, for all $t \in T$, $\langle\sqcap[\sigma_t],\sqcup[\sigma_t]\rangle$ is a generalised STP multimorphism of f'_t.*

We say that two n-variable instances I and I' of VCSP(Φ) are *equivalent* if $f_I(\mathbf{x}) = f_{I'}(\mathbf{x})$ for all $\mathbf{x} \in D^n$. Given a finite, valued constraint language $\Phi_0 \subset \mathrm{Func}(D,\overline{\mathbb{R}}_{\ge 0})$, let Φ'_0 be the set of functions of the form $f + R$, for $f \in \Phi_0 \cap \mathrm{Func}_k(D,\overline{\mathbb{R}}_{\ge 0})$, $R \in \mathrm{Func}_k(D,\{0,\infty\})$ and $k \in \mathbb{N}$. Note that Φ'_0 is finite because $\mathrm{Func}_k(D,\{0,\infty\})$ is finite for any finite k. These definitions, along with Corollary 25 and Theorem 26 allow us to prove the following.

▶ **Lemma 27.** *Suppose Φ_0 is a finite, valued constraint language which has an STP/MJN multimorphism. Consider an instance I of $\mathrm{VCSP}(\Phi_0)$. There is an equivalent instance I' of $\mathrm{VCSP}(\Phi_0')$ and a generalised STP multimorphism $\langle \sqcap, \sqcup \rangle$ of $f_{I'}$ which induces a generalised STP multimorphism of f_t for each valued constraint t of I'. Both I' and $\langle \sqcap, \sqcup \rangle$ are polynomial-time computable (given I). Moreover, each operation $\sqcap_i$ and $\sqcup_i$ induces a total order.*

The proof of the next lemma uses a reduction from the given #CSP problem over domain D to a Boolean problem. A variable x ranging over D is simulated by Boolean variables $\{x_d \mid d \in D\}$ and an assignment $x = a$ is simulated by setting $x_b = 1$ for all $b < a$ and $x_b = 0$, otherwise, where "$b < a$" is determined by the appropriate total order from the multimorphism. The constraints needed to prevent assignments to the x_d that do not correspond to an assignment to x can be represented by LSM functions, as can the corresponding translation of a weight function $f\colon D^k \to [0,1]_{\mathbb{Q}}$ to $f'\colon \{0,1\}^{kd} \to [0,1]_{\mathbb{Q}}$. In the case where all functions in $\mathcal{F}$ have arity at most 2, it can be shown that all of the necessary constraints can be implemented using Boolean implication and unary weights.

▶ **Lemma 28.** *If $\mathcal{F} \subseteq \mathrm{Func}(D, [0,1]_{\mathbb{Q}})$ and $\ell(\mathcal{F})$ has an STP/MJN multimorphism, then, for every finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is* LSM*-easy, and #*BIS*-easy if every weight function has arity at most 2.*

Our main theorem, Theorem 2, now follows from Theorems 4 and 7 and the following.

▶ **Theorem 29.** *Let $\mathcal{F}$ be a weakly log-supermodular, conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$.*

- *For any finite $\mathcal{G} \subset \mathcal{F}$, there is a finite $\mathcal{G}' \subset$ LSM such that $\#\mathrm{CSP}(\mathcal{G}) \leq_{\mathrm{AP}} \#\mathrm{CSP}(\mathcal{G}')$.*
- *If all $F \in \mathcal{F}$ have arity at most two, then #CSP($\mathcal{G}$) is #*BIS*-easy for any finite $\mathcal{G} \subset \mathcal{F}$.*

Proof (sketch). Scale $\mathcal{F}$ to obtain a weakly log-supermodular weighted constraint language $\mathcal{F}'$ where all weights are in $[0,1]_{\mathbb{Q}}$. By Theorem 20, $\ell(\mathcal{F}')$ has an STP/MJN multimorphism. The result follows from Lemma 28 and the fact that the scaling does not affect complexity. ◀

7 Algorithmic aspects

Finally, we show that there is an algorithm that determines the complexity of approximating #CSP with constraints from the language $\mathcal{H} \cup \mathcal{U}_D$ for finite $\mathcal{H}$. The proof is by reduction to determining whether a certain finite subset of $\mathcal{H} \cup \mathcal{U}_D$ is balanced, whether $\ell(\mathcal{H})$ has an STP/MJN multimorphism and whether $\mathcal{H}$'s weight functions have arity at most 2. Balance is decidable by [7], the existence of an STP/MJN multimorphism can be checked by brute force, since $\mathcal{H}$ is finite, and we can clearly check the maximum arity of the weight functions.

▶ **Theorem 30.** *There is an algorithm that, given a finite, weighted constraint language $\mathcal{H}$ taking values in $\mathbb{Q}_{\geq 0}$, correctly makes one of the following deductions, where $\mathcal{F} = \mathcal{H} \cup \mathcal{U}_D$:*

1. *#CSP($\mathcal{G}$) is in* FP *for every finite $\mathcal{G} \subset \mathcal{F}$;*
2. *#CSP($\mathcal{G}$) is* LSM*-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and #*BIS*-hard for some such $\mathcal{G}$;*
3. *#CSP($\mathcal{G}$) is #*BIS*-easy for all finite $\mathcal{G} \subset \mathcal{F}$ and #*BIS*-equivalent for some such $\mathcal{G}$;*
4. *#CSP($\mathcal{G}$) is #*SAT*-easy for all finite $\mathcal{G} \subset \mathcal{F}$ and #*SAT*-equivalent for some such $\mathcal{G}$.*

If every function in $\mathcal{H}$ has arity at most 2, the output is not deduction 2.

References

1 E. Boros and P. Hammer. Pseudo-Boolean optimization. *Discrete Appl. Math.*, 123:155–225, 2002.

2 A. Bulatov. The complexity of the counting constraint satisfaction problem. In *Proc. 35th ICALP (Part I)*, LNCS 5125, pp. 646–661. Springer, 2008.

3 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jalsenius, M. Jerrum and D. Richerby. The complexity of weighted and unweighted #CSP. *J. Comput. Syst. Sci.*, 78:681–688, 2012.

4 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jalsenius and D. Richerby. The complexity of weighted Boolean #CSP with mixed signs. *Theor. Comput. Sci.*, 410:3949–3961, 2009.

5 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jerrum and C. McQuillan. The expressibility of functions on the Boolean domain, with applications to counting CSPs. CoRR eprint, arXiv:abs/1108.5288, 2011.

6 J.-Y. Cai and X. Chen. Complexity of counting CSP with complex weights. In *Proc. 44th STOC*, pp. 909–920. ACM, 2012.

7 J.-Y. Cai, X. Chen, and P. Lu. Non-negatively weighted #CSP: An effective complexity dichotomy. In *Proc. 26th CCC*, pp. 45–54. IEEE, 2011.

8 J.-Y. Cai, P. Lu, and M. Xia. Dichotomy for Holant* problems of Boolean domain. In *Proc. 22nd SODA*, pp. 1714–1728. ACM–SIAM, 2011.

9 J.-Y. Cai, P. Lu and M. Xia. Holant problems and counting CSP. In *Proc. 41st STOC*, pp. 715–724. ACM, 2009.

10 X. Chen. Complexity dichotomies of counting problems. *SIGACT News*, 42:54–76, 2011.

11 X. Chen, M. Dyer, L. A. Goldberg, M. Jerrum, P. Lu, C. McQuillan and D. Richerby. The complexity of approximating conservative counting CSPs. CoRR eprint, arXiv:abs/1208.1783, 2012.

12 D. Cohen, M. Cooper and P. Jeavons. Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theor. Comput. Sci.*, 401:36–51, 2008.

13 D. Cohen, M. Cooper, P. Jeavons and A. Krokhin. Soft constraints: Complexity and multimorphisms. In *Proc. 9th CP*, LNCS 2833, pp. 244–258. Springer, 2003.

14 N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Inform. Comput.*, 125:1–12, 1996.

15 M. Dyer, L. A. Goldberg, C. Greenhill and M. Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38:471–500, 2004.

16 M. Dyer, L. A. Goldberg and M. Jerrum. The complexity of weighted Boolean CSP. *SIAM J. Comput.*, 38:1970–1986, 2009.

17 M. Dyer, L. A. Goldberg and M. Jerrum. An approximation trichotomy for Boolean #CSP. *J. Comput. Syst. Sci.*, 76:267–277, 2010.

18 M. Dyer and D. Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM J. Comput.*, to appear. (Conference version in *STOC 2010*.)

19 L. A. Goldberg and M. Jerrum. Approximating the partition function of the ferromagnetic Potts model. *J. Assoc. Comput. Mach.*, to appear. (Conference version in *ICALP 2010*.)

20 V. Kolmogorov and S. Živný. The complexity of conservative valued CSPs. CoRR eprint, arXiv:abs/1110.2809, 2011. (Conference version in *SODA 2012*.)

21 P. Lu. Complexity dichotomies of counting problems. ECCC eprint, 18:93, 2011.

22 M. Mitzenmacher and E. Upfal. *Probability and Computing*. CUP, 2005.

23 R. Rudolf and G. J. Woeginger. The cone of Monge matrices: extremal rays and applications. *Math. Method Oper. Res.*, 42:161–168, 1995.

24 R. Takhanov. A dichotomy theorem for the general minimum cost homomorphism problem. CoRR eprint, arXiv:abs/0708.3226, 2007. (Conference version in *STACS 2010*.)

25 D. Topkis. Minimizing a submodular function on a lattice. *Oper. Res.*, 26:305–321, 1978.

26 T. Yamakami. Approximate counting for complex-weighted Boolean constraint satisfaction problems. *Inform. Comput.*, 219:17–38, 2012.