Theoretical Computer Science 410 (2009) 3949–3961

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The complexity of weighted Boolean #CSP with mixed signs[☆]

Andrei Bulatov[a], Martin Dyer[b], Leslie Ann Goldberg[c], Markus Jalsenius[c,*], David Richerby[b]

[a] *School of Computing Science, Simon Fraser University, Burnaby, Canada*

[b] *School of Computing, University of Leeds, Leeds, LS2 9JT, UK*

[c] *Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK*

ARTICLE INFO

Article history:
Received 22 December 2008
Received in revised form 27 May 2009
Accepted 2 June 2009
Communicated by J. Diaz

Keywords:
Complexity of counting
Constraint satisfaction
Partition function

ABSTRACT

We give a complexity dichotomy for the problem of computing the partition function of a weighted Boolean constraint satisfaction problem. Such a problem is parameterized by a set Γ of rational-valued functions, which generalize constraints. Each function assigns a weight to every assignment to a set of Boolean variables. Our dichotomy extends previous work in which the weight functions were restricted to being non-negative. We represent a weight function as a product of the form $(-1)^s g$, where the polynomial s determines the sign of the weight and the non-negative function g determines its magnitude. We show that the problem of computing the partition function (the sum of the weights of all possible variable assignments) is in polynomial time if either every function in Γ can be defined by a "pure affine" magnitude with a quadratic sign polynomial or every function can be defined by a magnitude of "product type" with a linear sign polynomial. In all other cases, computing the partition function is $\mathsf{FP}^{\#\mathsf{P}}$-complete.

1. Introduction

The principal result of this paper is a dichotomy theorem for the complexity of computing the partition function of a weighted Boolean constraint satisfaction problem. This problem has a set of functions Γ that are used to assign a weight to any configuration, where a configuration is an assignment of values to the instance's variables. These functions generalize constraint relations in the classical constraint satisfaction problem (CSP), which corresponds to the case where all functions in Γ have range $\{0, 1\}$. The problem we consider here is to compute the *partition function* of a given instance of weighted CSP; that is, the sum of weights of all configurations. Computing the partition function generalizes the problem of counting the number of satisfying solutions of a CSP. We denote by #CSP(Γ) the problem of computing the partition function of weighted CSP instances which use functions from the set Γ.

The term "partition function" originates in statistical physics, and certain problems from statistical physics may be expressed as weighted CSPs. For example, the Potts model [22] can be expressed as a weighted CSP, whereas only the "hard core" version can be expressed as a classical CSP. The two possible hard core versions of the Potts model correspond to graph colouring, in the so-called antiferromagnetic case, and the trivial problem of colouring each component of a graph with a single colour, in the so-called ferromagnetic case.

Here, we extend the work of Dyer, Goldberg and Jerrum [8], who gave a dichotomy for the complexity of #CSP(Γ) when every function in Γ is restricted to have non-negative values. They defined two classes of functions, those that are "pure affine" and those of "product type", and showed that #CSP(Γ) is in FP if, and only if, every function in Γ is pure affine

[☆] Funded in part by the EPSRC grant "The Complexity of Counting in Constraint Satisfaction Problems".
[*] Corresponding author. Tel.: +44 151 795 4281.

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.06.003

or every function is of product type. Otherwise #CSP(Γ) is complete for $\mathsf{FP}^{\#\mathsf{P}}$. The existence of algorithms for testing the properties of being purely affine or of product type means that the dichotomy is decidable.

The contribution of this paper is a dichotomy theorem for #CSP(Γ), where Γ is allowed to contain functions which give values of either sign. Specifically, #CSP(Γ) is either in FP or is $\mathsf{FP}^{\#\mathsf{P}}$-complete. As in the non-negative case, the dichotomy is decidable.

This extension is of particular interest because functions having mixed signs can cause cancellations in the partition function, which may make it easier to compute. Many natural problems can be expressed as weighted #CSP problems with functions of mixed signs. For example, if f is a binary function, an instance I of #CSP($\{f\}$) corresponds to a graph G_I where each variable of I is a vertex and each constraint corresponds to an edge. There is a binary function $f: \{0, 1\}^2 \to \{-1, 1\}$ such that the partition function of #CSP($\{f\}$) counts the number of subgraphs in G_I that have an even number of edges — see the examples in Section 1.3 for details.

1.1. Constraint satisfaction

Constraint satisfaction provides a general framework for modelling decision problems and has many practical applications, particularly in artificial intelligence — see, for example, [19]. Decisions are modelled by variables, which are subject to constraints that model the logical and resource restrictions. Many interesting problems can be modelled in this way, including problems in the areas of satisfiability, scheduling and graph-theory. Consequently, the computational complexity of constraint satisfaction problems has become a major and active area of research [6,15].

A constraint satisfaction problem (CSP) has a finite *domain*, which we may denote by $\{0, 1, \ldots, q-1\}$ for some positive integer q. In this paper we are interested only in the *Boolean* case, where $q = 2$. A *constraint language* Γ with domain $\{0, 1, \ldots, q-1\}$ is a set of relations on $\{0, 1, \ldots, q-1\}$. For example, let $q = 2$, and consider the relation $R = \{(1, 0, 0), (1, 0, 1), (0, 1, 0), (0, 1, 1)\}$. This is a 3-ary relation on the domain $\{0, 1\}$, having four tuples.

Given a constraint language Γ, an *instance* of CSP(Γ) is a set of *variables* $V = \{v_1, \ldots, v_n\}$ and a set of *constraints*. Each constraint has a *scope*, which is a tuple of variables and a relation from Γ of the same arity, which constrains the variables in the scope. A *configuration* σ is a function from V to $\{0, 1, \ldots, q-1\}$. The configuration σ is *satisfying* if the scope of every constraint is mapped to a tuple that is in the corresponding relation. In our example above, a configuration σ satisfies the constraint with scope (v_3, v_7, v_2) and relation R if, and only if, it maps exactly one of v_3 and v_7 to the value 1. For a CSP with constraint language Γ, the decision problem CSP(Γ) is to determine whether a given instance I has a satisfying configuration. The counting problem #CSP(Γ) is to determine the number of distinct satisfying configurations of I.

Varying the constraint language Γ defines the classes CSP and #CSP of decision and counting problems. These contain problems of very different computational complexity. For example, if $\Gamma = \{R_1, R_2, R_3\}$ where R_1, R_2 and R_3 are the three binary relations defined by $R_1 = \{(0, 1), (1, 0), (1, 1)\}$, $R_2 = \{(0, 0), (0, 1), (1, 1)\}$ and $R_3 = \{(0, 0), (0, 1), (1, 0)\}$, then CSP($\Gamma$) is the classical 2-Satisfiability problem, which is in P. On the other hand, there is a similar constraint language that expresses 3-Satisfiability, which is NP-complete. There are cases where the counting problem is harder than the decision problem: if Γ is the constraint language defining 2-Satisfiability, then #CSP(Γ) contains the problem of counting independent sets in graphs, which is #P-complete [21], even for 3-regular graphs [14].

Any problem in CSP is in NP, but not every problem in NP can be expressed in CSP. For example, the question "Is the graph G Hamiltonian?" cannot be expressed in CSP, because the property of being Hamiltonian cannot be captured by constraints of fixed size. This is a limitation of the class CSP, but it also has an advantage. If P $\neq$ NP, there are problems which are neither in P nor NP-complete [16] but, for smaller classes of decision problems, the situation may be more straightforward. A dichotomy theorem may be possible, partitioning all problems in the class into those which are in P and those which are NP-complete, with no problems of intermediate complexity. It has been conjectured, in the seminal paper of Feder and Vardi [11], that there is a dichotomy theorem for CSP. Although much progress has been made towards proving this, it remains unproven to date.

In the Boolean case, the status of CSP was resolved by Schaefer [20]. Schaefer proved a dichotomy for the domain $\{0, 1\}$, giving four conditions on the constraint language Γ. If any of the conditions holds then CSP(Γ) is in P, otherwise CSP(Γ) is NP-complete. For details, the interested reader is referred to Schaefer's paper [20] or to Theorem 6.2 of the textbook [6]. An interesting feature is that Schaefer's conditions are all algorithmically checkable. Thus, given a constraint language Γ with domain $\{0, 1\}$, we can determine whether CSP(Γ) is in P or NP-complete.

While the conjectured dichotomy for CSP remains open, Bulatov [3] has recently made a major breakthrough for #CSP. He has shown that there is a dichotomy between FP and #P-complete, for the whole of #CSP. However, his proof sheds very little light on when #CSP(Γ) is in FP, and when it is #P-complete. The difficulty is that, while Γ itself is of fixed size, the criterion of the dichotomy involves finding a defect in any of a potentially infinite class of structures built on Γ. Whether this criterion is algorithmically checkable is an open question.

In the Boolean case, which is our focus here, a decidable dichotomy theorem for #CSP had already been established by Creignou and Hermann [5]. Before stating their theorem we introduce the following definition. A Boolean relation R is *affine* if it is the set of solutions to a system of linear equations over GF(2). A constraint language Γ is affine if every relation $R \in \Gamma$ is affine. Creignou and Hermann prove that #CSP(Γ) is in FP if Γ is affine and is #P-complete, otherwise. There is an algorithm that determines whether a Boolean constraint language Γ is affine, so there is an algorithm that determines whether #CSP(Γ) is in FP or #P-complete. In addition to Creignou and Hermann's dichotomy, Dyer, Goldberg and Jerrum [7] have given an approximation trichotomy for Boolean #CSP. Let #BIS denote the problem of counting the number of independent sets in a bipartite graph and let #SAT denote the problem of counting satisfying assignments to a Boolean formula in conjunctive normal form. Dyer, et al. [7] have shown that if Γ is not affine (hence #CSP(Γ) is #P-complete) then there is an approximation-preserving reduction between #CSP(Γ) and either #BIS or #SAT.

1.2. *Weighted* #CSP

The counting problem #CSP(Γ) can be extended naturally by replacing the relations in Γ by functions. We refer to the corresponding class of problems as *weighted* #CSP. The functions are used to assign weights to configurations and the *partition function* computes the sum of the weights over all configurations. We give a formal definition below. The partition function of a weighted #CSP generalizes the number of satisfying solutions of a classical #CSP. The classical setting may be recovered by restricting the range of every function to $\{0, 1\}$.

In weighted #CSP, a *constraint language* over a finite domain $\mathcal{D}$ is a finite collection of functions $\Gamma = \{f_i : \mathcal{D}^{r_i} \to \mathbb{Q} \mid i \in I\}$. The natural number r_i is called the *arity* of the function f_i; we refer to functions of arity one, two and three as unary, binary and ternary, respectively. In this paper we consider exclusively the Boolean domain, $\mathcal{D} = \{0, 1\}$.

An *instance* of a weighted constraint satisfaction problem over a constraint language Γ is a pair $I = (V, C)$, where $V = \{v_1, \ldots, v_n\}$ is a set of *variables* and C is a finite set of *constraints*. Each constraint is of the form $f(v_{i_1}, \ldots, v_{i_r})$, where f is an r-ary function in the set Γ. To keep notation simple, we will often use $x_1, x_2, \ldots$ as "metavariables", standing for variables in V.

A *configuration* of an instance (V, C) is a function $\sigma : V \to \mathcal{D}$, assigning a value from the domain to each variable. The *weight* of a configuration σ is

$$W(\sigma) := \prod_{f(x_1,\ldots,x_r)\in C} f(\sigma(x_1), \ldots, \sigma(x_r)).$$

We are interested in computing the *partition function* of an instance I. This is the sum $Z(I)$ of the weights of all possible configurations:

$$Z(I) := \sum_{\sigma : V \to \mathcal{D}} W(\sigma).$$

The *weighted constraint satisfaction problem* is the problem of computing $Z(I)$ given an instance I. Since this paper is exclusively about weighted CSPs, we will drop the word "weighted" and write #CSP(Γ) for the weighted constraint satisfaction problem over the constraint language Γ and #CSP for the union of #CSP(Γ) over all rational-weighted constraint languages. For constraint languages with only a single function f, we write #CSP(f), rather than #CSP($\{f\}$).

A constraint $f(x_1, \ldots, x_r)$ is *satisfied* by a configuration σ if $f(\sigma(x_1), \ldots, \sigma(x_r)) \neq 0$. Therefore, the weight of a configuration is zero unless it satisfies every constraint. If we restrict to constraint languages where every function has range $\{0, 1\}$, the weight of every configuration is either zero or one and $Z(I)$ is just the number of satisfying configurations for I. This corresponds precisely to the counting constraint satisfaction problem.

1.3. *Related work*

Bulatov's counting dichotomy [3] can be extended to weighted #CSP as long as the range of every function $f \in \Gamma$ is $\mathbb{Q}^{\geqslant 0}$ (the set of non-negative rationals) [1]. However, it is not known whether it extends to weighted #CSP with functions of mixed signs. Furthermore, there is currently no algorithm known that determines whether #CSP(Γ) is in FP or $\mathrm{FP}^{\#\mathrm{P}}$-complete, given a constraint language Γ. For the special case of graph homomorphisms, an effective dichotomy is known for functions of mixed signs [12]. There is also a dichotomy theorem by Dyer et al. for Boolean weighted #CSP for functions that are non-negative [8]. This is expressed in terms of two classes of functions, *pure affine* and *product type*, which we define in Section 2.1. Dyer et al. give the following theorem.

Theorem 1 ([8, Theorem 4]). *Let Γ be a constraint language in which the range of every function $f \in \Gamma$ is a set of non-negative rationals. If every function in Γ is pure affine, then #CSP(Γ) $\in$ FP. If every function in Γ is of product type, then #CSP(Γ) $\in$ FP. Otherwise, #CSP(Γ) is $\mathrm{FP}^{\#\mathrm{P}}$-complete.*

There exist algorithms that test whether a Boolean constraint language Γ is pure affine or of product type. This means that the dichotomy is effectively decidable.

The contribution of this paper (Theorem 9 below) extends Theorem 1 to constraint languages Γ containing arbitrary rational-valued functions. This is an interesting extension since functions with negative values can cause cancellations and may make the partition function easier to compute.[1] Independently, Cai, Lu and Xia have recently found a wider generalization, giving a dichotomy for the case where Γ can be any set of complex-valued functions [4].

The case of mixed signs has been been considered previously by Goldberg, Grohe, Jerrum and Thurley [12], in the case of one symmetric binary function on an arbitrary finite domain. Their theorem generalizes that of Bulatov and Grohe [2] for

[1] In a related context, recall the sharp distinction in complexity between computing the permanent and the determinant of a matrix.

the non-negative case. Goldberg et al. [12] give two examples, which can also be expressed as Boolean weighted #CSP, and fall within the scope of this paper. The first appeared as an open problem in [2]. The complexity of these problems can be deduced from [12] and from the results of this paper.

Example 2. The first example in [12] is the function $f: \{0,1\} \to \{-1,1\}$, where

$$\begin{array}{ll} f(0,0)=1 & f(0,1)=1 \\ f(1,0)=1 & f(1,1)=-1. \end{array}$$

An instance I of #CSP(f) can be represented by a graph $G=(V,E)$ with n vertices. (In fact the argument remains the same even in the case where G is a multigraph with self-loops.) The set of variables in I is V and, for each edge $(u,v)\in E$, we have the constraint $f(u,v)$ in I. Then $\frac{1}{2}Z(G)+2^{n-1}$ is the number of induced subgraphs of G with an even number of edges. Hence, up to a simple transformation, the partition function $Z(G)$ counts induced subgraphs with an even number of edges. To see this, observe that for every configuration σ, the term $\prod_{(u,v)\in E} f(u,v)$ is 1 if the subgraph of G induced by $\sigma^{-1}(1)$ has an even number of edges and -1 otherwise. In terms of our Theorem 9 below, $f(x,y)=(-1)^{xy}$, so this problem is in FP, and an algorithm for computing $Z(G)$ follows from Lemma 10 below.

Example 3. The second example in [12] is #CSP(f), where

$$\begin{array}{ll} f(0,0)=1 & f(0,1)=-1 \\ f(1,0)=-1 & f(1,1)=1. \end{array}$$

In terms of Theorem 9 below, $f(x,y)=(-1)^{x+y}$, so this problem is also in FP. This can easily be shown directly. Let $G=(V,E)$ be a graph with n vertices. Note that $Z(G)$ is unchanged by removing any circuit from G. Thus we may reduce G to a forest F, which will have edges if, and only if, G was not Eulerian. If F has no edges, then $Z(G)=Z(F)=2^n$. Otherwise F has at least one leaf vertex v, and then we have $Z(G)=Z(F)=Z(F\setminus v)-Z(F\setminus v)=0$. Thus $Z(G)=0$ unless G is Eulerian, in which case $Z(G)=2^n$, and hence the problem is trivially in FP.

1.4. Complexity

Since our weights are arbitrary rationals, the partition function Z is not, in general, an integer-valued function. As such, #CSP(Γ) is not, in general, in the class #P. However, it is easy to see that, for every constraint language Γ, there is a partition function Z' in #P and an FP-computable integer-valued function K such that, for all instances I of #CSP(Γ), $Z(I)=Z'(I)/K(I)$. This is achieved by "clearing denominators" (see [13]).

Following [13], we write $\#\mathrm{P}_{\mathbb{Q}}$ for the class of functions of the form f/g, where $f\in\#\mathrm{P}$ and $g\in\mathrm{FP}$. It is immediate that

$$\#\mathrm{CSP}\subseteq\#\mathrm{P}_{\mathbb{Q}}\subseteq\mathrm{FP}^{\#\mathrm{P}}.$$

Proposition 4. *Every* #CSP *problem that is* #P*-hard is* $\mathrm{FP}^{\#\mathrm{P}}$*-complete.*

Proof. If $Z(I)\in\#\mathrm{CSP}$ is #P-hard, we can use an oracle for $Z(I)$ to construct an oracle for $Z'(I)$, as described above. With this oracle, we can compute any problem in $\mathrm{FP}^{\#\mathrm{P}}$ in a polynomial number of steps. Therefore, $Z(I)$ is $\mathrm{FP}^{\#\mathrm{P}}$-complete. □

Let Γ be a constraint language. We say that Γ *simulates* a function $f\notin\Gamma$ if, given an instance I of #CSP($\Gamma\cup\{f\}$), we can construct, in polynomial time, an instance I' of #CSP(Γ) such that $Z(I)=K(I)Z(I')$ for some FP-computable function K. This generalizes parsimonious reductions [18]; clearly, if Γ simulates f then $\#\mathrm{CSP}(\Gamma\cup\{f\})\leqslant_{\mathrm{T}}\#\mathrm{CSP}(\Gamma)$, where $\leqslant_{\mathrm{T}}$ denotes polynomial-time Turing reducibility. If $\#\mathrm{CSP}(\Gamma)\leqslant_{\mathrm{T}}\#\mathrm{CSP}(\Gamma')\leqslant_{\mathrm{T}}\#\mathrm{CSP}(\Gamma)$, we write $\#\mathrm{CSP}(\Gamma)\equiv_{\mathrm{T}}\#\mathrm{CSP}(\Gamma')$.

1.5. Organization of the paper

Our paper is organized as follows. In Section 2, we define notation and the classes of functions we will use throughout the paper. In Section 3, we state our dichotomy result and prove the polynomial-time cases. The remaining sections prove that all other cases are $\mathrm{FP}^{\#\mathrm{P}}$-complete. We give useful tools for proving hardness in Section 4. In Sections 5 and 6, we show, respectively, that any constraint language containing a pure affine function of degree greater than 2 is #P-hard and that and any language with a function of product type of degree greater than 1 can be made #P-hard by adding a simple function. Finally, we complete the proof of the dichotomy in Section 7, showing that the simple function is can be simulated by the functions already present.

2. Some notation

All sets and other objects referred to in this paper are finite unless it is stated otherwise. We write $\bar{a}$ for a tuple of elements $(a_1,\ldots,a_r)$ for some r and, for natural numbers $m\leqslant n$, we write $[m,n]$ for the set $\{m,m+1,\ldots,n\}$.

The *support* of a function $f : X^r \to \mathbb{Q}$ is the r-ary relation $\{\bar{a} \mid f(\bar{a}) \neq 0\}$. For a function $g : X \to Y$ and a tuple $\bar{a} \in X^r$, we write $g(\bar{a})$ for the tuple $(g(a_1), \ldots, g(a_r))$.

We write $\mathcal{F}_B$ for the set of all functions, of all positive arities, from the set $\{0, 1\}$ to $\mathbb{Q}$, the rationals, and $\mathcal{F}_B^{\geqslant 0}$ for the subset of $\mathcal{F}_B$ consisting of all functions with non-negative ranges. We write $\mathcal{P}_k$ for the set of multivariate polynomials in variables $x_1, \ldots, x_k$ over GF(2). We sometimes write $p(x_1, \ldots, x_k)$ for a polynomial $p \in \mathcal{P}_k$ or other function, to emphasize that p is a function of those variables.

A function $f(x_1, \ldots, x_k)$ *depends on* a variable x_i if there are constants $c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_k \in \{0, 1\}$ such that

$$f(c_1, \ldots, c_{i-1}, 0, c_{i+1}, \ldots, c_k) \neq f(c_1, \ldots, c_{i-1}, 1, c_{i+1}, \ldots, c_k).$$

2.1. Classes of functions

In this section, we define the classes of functions that we use throughout the paper. Our definitions of pure affine functions and functions of product type are those used by Dyer et al. [8] but multiplied by a term $(-1)^s$ for some polynomial s, which determines the sign.

Recall that a relation over $\{0, 1\}$ is *affine* if it is the solution set of a set of linear equations over GF(2). We say that a function $f \in \mathcal{F}_B$ is affine if it has affine support.

Definition 5. A k-ary function $f \in \mathcal{F}_B$ is *pure affine* if there is a constant $w \in \mathbb{Q}^{>0}$, an affine function $g \in \mathcal{P}_k$ and a polynomial $s \in \mathcal{P}_k$ such that

$$f(\bar{x}) = w(-1)^{s(\bar{x})} g(\bar{x}). \tag{1}$$

Note that the range of f in (1) is included in $\{-w, 0, w\}$. The polynomial g is uniquely defined, up to the identities $x \oplus x = 0$ and $x^2 = x$. However, because the value of f does not depend on the value of s for values of its inputs where $g(\bar{x}) = 0$, there may be several distinct polynomials s for which the identity (1) holds. If s is of minimal degree d such that (1) holds, we say that s is *degree-minimized* with respect to g and that f is *pure affine of degree d*. For the purposes of this paper, we consider the constant zero and one polynomials to have degree zero.

We write $\chi_=$ and $\chi_{\neq}$ for the binary equality and disequality functions, respectively, defined as

$$\chi_=(x, y) = x \oplus y \oplus 1 \qquad \chi_{\neq}(x, y) = x \oplus y.$$

Definition 6. A k-ary function $f \in \mathcal{F}_B$ is of *product type* if there are unary functions $U_1(x_1), \ldots, U_k(x_k) : \{0, 1\} \to \mathbb{Q}^{\geqslant 0}$, a polynomial $g \in \mathcal{P}_k$ that is a product of binary functions of the form $\chi_=$ and $\chi_{\neq}$, and a polynomial $s \in \mathcal{P}_k$ such that

$$f(\bar{x}) = (-1)^{s(\bar{x})} U_1(x_1) \cdots U_k(x_k) g(\bar{x}). \tag{2}$$

The function f is of *product type of degree d* if s is of degree d and is degree-minimized with respect to $U_1(x_1) \cdots U_k(x_k) g(\bar{x})$.

Let $f \in \mathcal{F}_B$ be of product type and let $(-1)^{s(\bar{x})} U_1(x_1) \cdots U_k(x_k) g(\bar{x})$ be an expression of f as in the definition. We call a variable x_i in the representation *determined* if exactly one of the terms in g is an equality or disequality involving x_i, $U_i(0) = U_i(1) = 1$ and s does not depend on x_i.

Example 7. Let $f(x_1, \ldots, x_5)$ be the 5-ary function with $f(0, 0, 1, 0, 1) = 8$, $f(0, 1, 1, 0, 1) = 10$, $f(1, 0, 0, 1, 1) = -12$, $f(1, 1, 0, 1, 1) = 15$ and $f = 0$ for all other inputs. Then f is of product type of degree 2, because we can write

$$f(x_1, \ldots, x_5) = (-1)^{x_1 x_2 \oplus x_1} U_1(x_1) \cdots U_5(x_5) \chi_{\neq}(x_1, x_3) \chi_=(x_1, x_4),$$

where $U_1(0) = 2$, $U_1(1) = 3$, $U_2(0) = 4$, $U_2(1) = 5$, $U_3(0) = U_3(1) = U_4(0) = U_4(1) = 1$, $U_5(0) = 0$ and $U_5(1) = 1$. The variables x_3 and x_4 are determined.

It is convenient to impose certain restrictions on expressions for functions of product type. We say that the expression for f is *normalized* if the following conditions are met:

- at least one variable in every equality and disequality term in g is determined,
- if, for some i, $U_i(0) = 0$ or $U_i(1) = 0$, then g and s do not depend on x_i, and
- s is degree-minimized with respect to $U_1(x_1) \cdots U_k(x_k) g(\bar{x})$.

Note that the expression given in Example 7 is normalized: the variables x_3 and x_4 are determined; $U_5(0) = 0$, so neither s nor g depends on x_5; and no sign polynomial of degree 0 or 1 is equivalent to $x_1 x_2 \oplus x_1$, even with the flexibility given by the numerous inputs for which $f = 0$.

Lemma 8. *Every function $f \in \mathcal{F}_B$ that is of product type is defined by a normalized expression.*

Proof. Let $(-1)^{s(\bar{x})} U_1(x_1) \cdots U_k(x_k) g(\bar{x})$ be a non-normalized expression defining f.

Suppose g contains a term $\chi_=(x_i, x_j)$ where neither x_i nor x_j is determined. First, substitute x_i for x_j in every other term of g and in s. Replace U_j with the function that maps both 0 and 1 to 1 and U_i with the function $U_i(x_i) U_j(x_i)$. The variable x_j is now determined in the resulting expression, which still defines f.

Suppose g contains a term $\chi_{\neq}(x_i, x_j)$ where neither x_i nor x_j is determined. We proceed as above but substitute $x_i \oplus 1$ for x_j. Having done so, there may be terms $\chi_{=}(x_\ell, x_i \oplus 1)$ and $\chi_{\neq}(x_\ell, x_i \oplus 1)$; replace these with $\chi_{\neq}(x_\ell, x_i)$ and $\chi_{=}(x_\ell, x_i)$, respectively, and similarly for the terms with the parameters the other way round.

Suppose that $U_i(c) = 0$ for some $c \in \{0, 1\}$ but g or s depends on x_i. Since f is zero if $x_i = c$, we may replace x_i with $c \oplus 1$ throughout g and s. Performing such a replacement in a term of g results in that term becoming a unary function, which can be incorporated into the corresponding U_j.

Finally, if s is not degree-minimized, replace it with a polynomial in the appropriate variables that is. □

We say that a k-ary function $f: \{0, 1\}^k \to \mathbb{Q}$ is *positive pure affine* or of *positive product type* if it can be written according to Definition 5 or Definition 6, respectively, but choosing the sign polynomial s to be identically zero. Thus, positive pure affine and positive product type correspond exactly to the definitions of pure affine and product type used by Dyer et al. for functions $\{0, 1\}^k \to \mathbb{Q}^{\geqslant 0}$ [8]. Observe that, if a function $f: \{0, 1\}^k \to \mathbb{Q}^{\geqslant 0}$ is pure affine (respectively, of product type) then it is positive pure affine (respectively, of positive product type). This is because we must have $s(\bar{x}) = 0$ whenever $f(\bar{x}) \neq 0$ and, when $f(\bar{x}) = 0$, we can set $s(\bar{x}) = 0$ without altering the value of f. Thus, all properties of the functions that Dyer et al. call "pure affine" or "of product type" in [8] carry over to non-negative functions that we call pure affine and of product type, respectively.

3. The dichotomy

We now give our main result, a complexity dichotomy for Boolean #CSP with rational weights. In this section, we prove the tractability of the polynomial-time cases and comment on our definitions of the classes of pure affine and product-type functions. Proving $\mathrm{FP}^{\#\mathrm{P}}$-completeness of the remaining cases requires considerably more work and is the subject of the remainder of the paper.

Theorem 9. *Let $\Gamma \subseteq \mathcal{F}_{\mathrm{B}}$. If every function in Γ is pure affine of degree at most 2, then #CSP(Γ) is in FP. If every function in Γ is of product type of degree at most 1, then #CSP(Γ) is in FP. Otherwise, #CSP(Γ) is $\mathrm{FP}^{\#\mathrm{P}}$-complete.*

Proof. The two polynomial-time cases are covered by Lemmas 10 and 12 in this section. If we are not in one of these cases, then Γ must contain functions f and g (not necessarily distinct) such that f is not pure affine of degree at most 2 and g is not of product type of degree at most 1. $\mathrm{FP}^{\#\mathrm{P}}$-completeness follows from Lemmas 13 and 30. □

Following from the observations at the end of the previous section, if we have $\Gamma \subseteq \mathcal{F}_{\mathrm{B}}^{\geqslant 0}$, then Theorem 9 is equivalent to Theorem 4 of Dyer et al. [8].

It is worth pointing out that we cannot simply dispense with the sign polynomial in the definitions of pure affine and product type and, instead, allow the constants and unary functions to take negative values. Temporarily call a function $f: \{0, 1\}^k \to \mathbb{Q}$ *weakly pure affine* if there is a constant $w \in \mathbb{Q}$ and an affine polynomial $g \in \mathcal{P}_k$ such that $f(\bar{x}) = wg(\bar{x})$ and of *weak product type* if there are unary functions $U_i: \{0, 1\} \to \mathbb{Q}$ and a product g of equalities and disequalities such that $f(\bar{x}) = U_1(x_1) \cdots U_k(x_k)g(\bar{x})$. It is not hard to see that every function that is weakly pure affine or of weak product type is pure affine or of product type, respectively. However, the converse does not hold. The function $f(x, y) = (-1)^{xy}$ of Example 2 above is not weakly pure affine (there is no rational w such that its range is $\{0, w\}$) and not of weak product type (it is nowhere zero so there can be no non-trivial equality or disequality terms and the sign cannot be expressed as a combination of unary functions). However, it is trivially pure affine and of product type (of degree two in both cases).

Lemma 10. *Let $\Gamma \subseteq \mathcal{F}_{\mathrm{B}}$. If every function in Γ is pure affine of degree at most 2, then #CSP(Γ) $\in$ FP.*

Proof. Let $\Gamma = \{f_1, \ldots, f_m\}$, where each $f_i = w_i(-1)^{s_i}g_i$ and let $\Gamma' = \{f_1', \ldots, f_m'\}$, where each $f_i' = f_i/w_i = (-1)^{s_i}g_i$. Note that the range of each f_i' is included in $\{-1, 0, 1\}$.

Let I be an instance of #CSP(Γ) and, for each $i \in [1, m]$, let k_i be the number of constraints in I that involve the function f_i. Let I' be the instance of #CSP(Γ') made by replacing each constraint $f(\bar{x})$ in I with $f'(\bar{x})$. We have

$$Z(I) = Z(I') \prod_{1\leqslant i\leqslant m} w_i^{k_i},$$

so it suffices to show that we can compute $Z(I')$ in a polynomial number of steps.

If there are k constraints and n variables in I', $Z(I')$ is a sum of terms of the form

$$\prod_{1\leqslant j\leqslant k} (-1)^{s_{i_j}(\bar{x}_j)} g_{i_j}(\bar{x}_j) = (-1)^{s(v_1,\ldots,v_n)} \prod_{1\leqslant j\leqslant k} g_{i_j}(\bar{x}_j),$$

where $s(\bar{v}) = \sum_{1\leqslant j\leqslant k} s_{i_j}(\bar{x}_j)$.

We can write $Z(I') = N^+ - N^-$, where N^+ is the number of configurations of the variables of I with weight 1 and N^- is the number with weight -1. Now, N^+ is the number of solutions of the simultaneous equations

$$g_{i_1}(\bar{x}_1) = \cdots = g_{i_k}(\bar{x}_k) = 1$$

over GF(2) that have $s(\bar{v}) = 1$ and N^- is the number of solutions with $s(\bar{v}) = 0$. Since Γ' is pure affine of degree at most 2, each g_i is linear and s is quadratic. Lemma 11 below shows that the number of solutions to such a system of equations can be computed in polynomial time. □

Lemma 11. *There is a polynomial-time algorithm for the following problem: given a multivariate quadratic polynomial q over* GF(2) *and k multivariate linear polynomials $\ell_1, \dots, \ell_k$ over* GF(2), *determine the number of solutions that satisfy $q = 0$, $\ell_1 = 0, \dots, \ell_k = 0$ simultaneously.*

Proof. Suppose q and $\ell_1, \dots, \ell_k$ are in variables $x_1, \dots, x_n$ and suppose, without loss of generality, that ℓ_k depends on x_n. The polynomial ℓ_k evaluates to 0 if, and only if, $x_n = h(x_1, \dots, x_{n-1}) = \ell_k \oplus x_n$, where h is a linear polynomial in $x_1, \dots, x_{n-1}$. Substitute h for x_n in q and $\ell_1, \dots, \ell_{k-1}$ to obtain q' and $\ell'_1, \dots, \ell'_{k-1}$, respectively. The number of solutions that satisfy $q = 0, \ell_1 = 0, \dots, \ell_k = 0$ is the same as the number of solutions that satisfy $q' = 0, \ell'_1 = 0, \dots, \ell'_{k-1} = 0$, which may be found recursively. We process recursively until the system of equations contains one quadratic equation and no linear equations, or only linear equations. The number of solutions to a quadratic polynomial equation over GF(2) can be computed in polynomial time [10,17]. The number of solutions of a system of linear equations over GF(2) can be computed by Gaussian elimination in polynomial time. □

The case where every function in Γ is of product type of degree at most 1 is essentially the same as the corresponding case for non-negative functions [8] but we give a full proof for completeness.

Lemma 12. *Let $\Gamma \subseteq \mathcal{F}_B$. If every function in Γ is of product type of degree at most 1, then* #CSP(Γ) $\in$ FP.

Proof. Observe that, since each function $f \in \Gamma$ is of product type of degree at most 1, each can be written in the form

$$\begin{aligned} f(\bar{x}) &= (-1)^{x_{i_1}+\dots+x_{i_\ell}+c} U_1(x_1) \cdots U_k(x_k) g(\bar{x}) \\ &= (-1)^{x_{i_1}} \cdots (-1)^{x_{i_\ell}} (-1)^c U_1(x_1) \cdots U_k(x_k) g(\bar{x}), \end{aligned}$$

for some $c \in \{0, 1\}$. Thus, we can, instead, write $f(\bar{x}) = U'_1(x_1) \cdots U'_k(x_k) \cdot h(\bar{x})$ where each U'_i is a function $\{0, 1\} \to \mathbb{Q}$ instead of $\{0, 1\} \to \mathbb{Q}^{\geqslant 0}$. The remainder of the proof is the same as the corresponding case for non-negative functions.

Let I be an instance of #CSP(Γ), with variables V. Let $\approx$ be the finest equivalence relation over V such that $v_i \approx v_j$ if $i = j$ or some constraint in I requires that either $v_i = v_j$ or $v_i \neq v_j$. We process each equivalence class in turn, independently of the others.

Let $S \subseteq V$ be an equivalence class of $\approx$. If there is no assignment to the variables in S that satisfies the equalities and disequalities in I's constraints, then $Z(I) = 0$ and we are done. Otherwise, S must have a partition into sets S_0 and S_1 so that each variable in S_0 must have the same value and each variable in S_1 (which may be empty) must have the opposite value. The variables in S contribute one weight, say α, to $Z(I)$ if the variables in S_0 are set to 0 and another weight, say β, if they are set to 1. Thus, we can write $Z(I) = (\alpha + \beta)Z'(I)$, where $Z'(I)$ is the partition function $Z(I)$ with all terms involving the variables in S deleted. We may then proceed to factor out the next equivalence class. □

4. Useful reductions

In this section, we give several reductions that are useful for proving hardness of weighted Boolean #CSPs.

4.1. Pinning

Let δ_0 and δ_1 be the unary functions defined as

$$\begin{aligned} \delta_0(0) &= 1 \qquad \delta_1(0) = 0 \\ \delta_0(1) &= 0 \qquad \delta_1(1) = 1. \end{aligned}$$

These functions are referred to as *pinning* functions, since a constraint $\delta_c(x)$ "forces" the variable x to take value c by giving weight zero to any configuration with $x \neq c$. The proof of the following lemma is identical to the proof of [8, Lemma 8], except that the condition "$f(x) > f(\bar{x}) \geqslant 0$" in the first sentence of Case 2 needs to be replaced with "$f(x) \neq f(\bar{x})$".

Lemma 13. *For every $\Gamma \subseteq \mathcal{F}_B$,* #CSP($\Gamma \cup \{\delta_0, \delta_1\}$) $\leqslant_T$ #CSP(Γ).

4.2. Arity reduction

Given a k-ary function $f \in \mathcal{F}_B$ and $i \in [1, k]$, the function obtained by *projecting out the ith variable* is

$$g(x_1, \dots, x_{k-1}) = \sum_{y \in \{0,1\}} f(x_1, \dots, x_{i-1}, y, x_i, \dots, x_{k-1}).$$

The following is a special case of [8, Lemma 6]. Although that lemma is stated only for classes of non-negative rational functions, the proof does not rely on this.

Lemma 14. *Let $\Gamma \subseteq \mathcal{F}_B$, let $f \in \Gamma$ and let g be defined by projecting out a variable of f.* #CSP($\Gamma \cup \{g\}$) $\leqslant_T$ #CSP(Γ).

The *contraction* of a ternary function $f \in \mathcal{F}_B$ is the function

$$g(x_1, x_2) = \sum_{y,z \in \{0,1\}} f(x_1, y, z) f(y, z, x_2).$$

(In principle, we could define contractions in terms of any sequence of function arguments but we only use the version defined here.)

Lemma 15. *Let $\Gamma \subseteq \mathcal{F}_{\mathrm{B}}$ and let g be the contraction of some $f \in \Gamma$. $\#\mathrm{CSP}(\Gamma \cup \{g\}) \leqslant_{\mathrm{T}} \#\mathrm{CSP}(\Gamma)$.*

Proof. Replace each constraint C of the form $g(x, y)$ with the two constraints $f(x, x_C, y_C)$ and $f(x_C, y_C, y)$, where x_C and y_C are new variables, used only in these two constraints. $\square$

4.3. Arithmetic techniques

For a constant $q \in \mathbb{Q}$ and a function $f \in \mathcal{F}_{\mathrm{B}}$, write qf for the function that maps $\bar{x}$ to $qf(\bar{x})$.

Lemma 16. *Let $f \in \Gamma \subseteq \mathcal{F}_{\mathrm{B}}$ and let $q \neq 0$ be rational. $\#\mathrm{CSP}(\Gamma \cup \{qf\}) \leqslant_{\mathrm{T}} \#\mathrm{CSP}(\Gamma)$.*

Proof. Let I be an instance of $\#\mathrm{CSP}(\Gamma \cup \{qf\})$ and let I' be the instance of $\#\mathrm{CSP}(\Gamma)$ made by replacing every constraint $qf(\bar{x})$ in I with $f(\bar{x})$. $Z(I) = q^m Z(I')$, where m is the number of qf-constraints in I. $\square$

Given a constraint language Γ, let Γ^2 be the constraint language that replaces every function $f(\bar{x})$ with the function $(f(\bar{x}))^2$. The following lemma is immediate from the observation that an instance of $\#\mathrm{CSP}(\Gamma^2)$ can be converted to one of Γ with the same partition function just by including an extra copy of each constraint.

Lemma 17. $\#\mathrm{CSP}(\Gamma^2) \leqslant_{\mathrm{T}} \#\mathrm{CSP}(\Gamma)$.

Further, if $\Gamma \subseteq \mathcal{F}_{\mathrm{B}}$, then $\Gamma^2 \subseteq \mathcal{F}_{\mathrm{B}}^{\geqslant 0}$. This fact and the following lemma allow us to re-use results on those functions from [8].

Lemma 18. *$f \in \mathcal{F}_{\mathrm{B}}$ is pure affine (respectively, of product type) if, and only if, $f^2 \in \mathcal{F}_{\mathrm{B}}^{\geqslant 0}$ is pure affine (respectively, of product type).*

Proof. Let $f \in \mathcal{F}_{\mathrm{B}}$ be k-ary. It is clear that, if f is pure affine (respectively, of product type), then so is f^2; we show the converse. We assume that f^2 is not identically zero as this case is trivial.

First, suppose $f^2(\bar{x})$ is pure affine and equal to $w^2 g(\bar{x})$ as in Definition 5. There is a polynomial $s(\bar{x})$ that assigns the correct sign to each input such that $f(\bar{x}) = w(-1)^{s(\bar{x})} g(\bar{x})$. Therefore f is pure affine.

Now, suppose $f^2(\bar{x}) = U_1(x_1) \cdots U_k(x_k) g(\bar{x})$, as in Definition 6, is of product type. For each $i \in [1, k]$, let $U_i'(x_i) = \sqrt{U_i(x_i)}$. The functions U_i' are not necessarily rational but we certainly have

$$f(\bar{x}) = (-1)^{s(\bar{x})} U_1'(x_1) \cdots U_k'(x_k) g(\bar{x}), \tag{3}$$

for some suitable polynomial s, as before. By the arguments of Lemma 8, which do not depend on the rationality of the functions U_i', we may assume that this is a normalized expression for f, except for the possible irrationality of the U_i'.

We now describe how the functions U_i' can be replaced by rational functions, keeping the expression for f normalized. The function f is not identically zero so there is a tuple $\bar{a} \in \{0, 1\}^k$ such that $f(\bar{a}) \neq 0$. Since $f \in \mathcal{F}_{\mathrm{B}}$, $f(\bar{a})$ is rational. For $i \in [1, k]$, let $U_i''(x_i) = U_i'(x_i)/U_i'(a_i)$. Then

$$f(\bar{x}) = |f(\bar{a})|(-1)^{s(\bar{x})} U_1''(x_1) \cdots U_k''(x_k) g(\bar{x}). \tag{4}$$

The expression for f in (4) is not necessarily normalized because of the factor $|f(\bar{a})|$; however, as we will see next, the functions U_i'' are rational. Once we have established this fact we will see that the factor $|f(\bar{a})|$ (which is rational) can be included in one of the unary functions U_i'', giving us a normalized expression for f.

Note that, for $i \in [1, k]$, $U_i''(a_i) = 1$, which is rational. Therefore we need to show that $U_i''(a_i \oplus 1)$ is rational. Observe that, for each $i \in [1, k]$ for which x_i is determined in (3), we have $U_i''(0) = U_i''(1) = 1$. We now show that, for each $i \in [1, k]$ for which x_i is not determined, $U_i''(a_i \oplus 1)$ is rational. Suppose $U_i''(a_i \oplus 1) \neq 0$. Let $\bar{a}_i \in \{0, 1\}^k$ be the tuple obtained from $\bar{a}$ by replacing a_i with $a_i \oplus 1$ and replacing a_j with $a_j \oplus 1$ for every determined variable x_j that occurs together with x_i in an equality or disequality function of g. Thus, $g(\bar{a}_i) = 1$ and $|f(\bar{a}_i)| = |f(\bar{a})| U_i''(a_i \oplus 1) > 0$. Since $f \in \mathcal{F}_{\mathrm{B}}$ and $f(\bar{a})$ is rational, $U_i''(a_i \oplus 1)$ is rational.

Finally we notice that the factor $|f(\bar{a})|$ in (4) can be absorbed by any of the unary functions U_i'' for which x_i is not a determined variable in the expression for f in (3). If all variables are determined then we note that $|f(\bar{a})| = 1$ and we can disregard it completely. We finally conclude that $f(\bar{x})$ is of product type. $\square$

4.4. Matrix techniques

Given a $k \times k$ rational matrix, $A = (A_{ij})$, and a directed multigraph $G = (V, E)$, which may have loops, let

$$Z_A(G) = \sum_{\sigma: V \to [1,k]} \prod_{(x,y) \in E} A_{\sigma(x)\sigma(y)}.$$

The problem of computing $Z_A(G)$ for a given input graph G is denoted by $\textsc{Eval}(A)$. Bulatov and Grohe have given the complexity of $\textsc{Eval}(A)$ for any symmetric matrix A with non-negative entries [2]. Here we only need the following special case.

Lemma 19. *Let A be a symmetric* 2×2 *matrix with non-negative rational entries. If A has rank 2 and at most one entry of A is zero then* $\textsc{Eval}(A)$ *is #P-hard.*

For any $k \times k$ rational matrix A, $\textsc{Eval}(A)$ is just the same thing as #CSP(f) for an appropriate binary function f over a domain of size k. In particular, then, 2×2 matrices correspond to binary Boolean functions.

Lemma 20. *Let* $f \in \mathcal{F}_B$ *be a binary function and let A be the matrix*

$$A = \begin{pmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{pmatrix}.$$

Then $\textsc{Eval}(A) \equiv_T$ #CSP(f).

Proof. Given an instance graph $G = (V, E)$ of $\textsc{Eval}(A)$, let I be the instance of #CSP(f) with variables V that has a constraint $f(x, y)$ for every edge (x, y) in E. Thus $Z_A(G) = Z(I)$. □

While Lemma 20 applies to all rational functions f, Lemma 19 can only be used if the resulting matrix is both symmetric and non-negative. The following lemma, essentially due to Dyer and Greenhill [9] will allow us to transform the matrix corresponding to a function f into a symmetric, non-negative matrix. For a matrix $A = (A_{ij})$, we write $A^{(2)}$ for the matrix (A_{ij}^2).

Lemma 21. *For any rational square matrix A, the problems* $\textsc{Eval}(A^{(2)})$, $\textsc{Eval}(AA^{\mathrm{T}})$, $\textsc{Eval}(A^{\mathrm{T}}A)$ *and* $\textsc{Eval}(A^2)$ *are polynomial-time Turing-reducible to* $\textsc{Eval}(A)$.

Proof. For any graph G,

- $Z_{A^{(2)}}(G) = Z_A(G_1)$, where G_1 is the multigraph formed by replacing each edge of G with two parallel edges;
- $Z_{AA^{\mathrm{T}}}(G) = Z_A(G_2)$, where G_2 is the graph obtained by introducing a new vertex v_e for each edge $e = (x, y) \in G$ and replacing e with the edges (x, v_e) and (y, v_e);
- $Z_{A^{\mathrm{T}}A}(G) = Z_A(G_3)$, where G_3 is made in the same way as G_2 but replacing e with (v_e, x) and (v_e, y);
- $Z_{A^2}(G) = Z_A(G_4)$, where G_4 is made in the same way as G_2 but replacing e with (x, v_e) and (v_e, y). □

5. High-degree pure affine functions

We have seen that there is a polynomial-time algorithm for #CSP(Γ) if every function in Γ is pure affine of degree at most two. We now show that computing partition functions of pure affine functions of higher degree is #P-hard. The main result of this section is the following lemma.

Lemma 22. *If* $f \in \mathcal{F}_B$ *is pure affine of degree at least three, then* #CSP(f) *is #P-hard.*

We first consider the restricted case $f(x, y, z) = (-1)^{s(x,y,z)}$, for ternary functions s of degree exactly 3 and then show that the case $f(\bar{x}) = (-1)^{s(\bar{x})}$ of degree-3 functions of arbitrary arity greater than three follows. Finally, we prove Lemma 22.

Lemma 23. *Let* $f(x, y, z) = (-1)^{s(x,y,z)}$ *where* $s \in \mathcal{P}_3$ *is of degree 3.* #CSP(f) *is #P-hard.*

Proof. Since s is of degree 3, it must contain the term xyz. Note that #CSP($(-1)^{s(x,y,z)}$) is equivalent to #CSP($(-1)^{s(x,y,z)+1}$) under Turing reductions, since $Z(I) = (-1)^m Z(I')$ where I and I' are instances of the two problems with the same m constraints. Therefore, we may assume that s does not contain the constant term 1.

Given this assumption, the terms of s are xyz and some subset of the terms xy, yz, zx, x, y and z. By symmetry between the variables, there are twenty cases to consider, listed in Table 1. Each case is proven #P-hard by either projecting out a variable or contracting, as detailed in the table.

For each polynomial s listed in the table, let $f(x, y, z) = (-1)^{s(x,y,z)}$. Note that f has the same value when s is evaluated over $\mathbb{Z}$ as it does when s is evaluated over GF(2), so we need not distinguish between $+$ and $\oplus$. The operation given (projecting out a variable or contracting) produces a new function f' in two variables which we will call x and y. By Lemma 14 (projection), or Lemma 15 (contraction), #CSP(f') $\leqslant_T$ #CSP(f). Further, by Lemma 20, $\textsc{Eval}(A) \leqslant_T$ #CSP(f'), where

$$A = \begin{pmatrix} f'(0,0) & f'(0,1) \\ f'(1,0) & f'(1,1) \end{pmatrix}$$

is given in the table. Let $A' = (AA^{\mathrm{T}})^{(2)}$. For any rational matrix A, the corresponding A' is symmetric and non-negative and, by Lemma 21, $\textsc{Eval}(A') \leqslant_T \textsc{Eval}(A)$. All of the matrices A' given in the table have rank 2 and no zero entries so, by Lemma 19, $\textsc{Eval}(A')$ is #P-hard. □

Lemma 24. *Let* $\bar{x} = x_1 \ldots x_k$ *for some* $k > 3$ *and let* $f(\bar{x}) = (-1)^{s(\bar{x})}$ *for some* $s \in \mathcal{P}_k$ *of degree at least 3.* #CSP(f) *is #P-hard.*

Table 1
The twenty ternary degree-3 polynomials considered in Lemma 23, with the methods used to prove them hard and the corresponding matrices.

$s(x, y, z)$	Method	A	A'
xyz	Project out z	$\begin{pmatrix} 2 & 2 \\ 2 & 0 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + x$	Project out z	$\begin{pmatrix} 2 & 2 \\ -2 & 0 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + x + y$	Project out z	$\begin{pmatrix} 2 & -2 \\ -2 & 0 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + x + y + z$	Contract	$\begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix}$	$\begin{pmatrix} 400 & 256 \\ 256 & 400 \end{pmatrix}$
$xyz + xy$	Project out z	$\begin{pmatrix} 2 & 2 \\ 2 & 0 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + x$	Project out z	$\begin{pmatrix} 2 & 2 \\ -2 & 0 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + x + y$	Project out z	$\begin{pmatrix} 2 & -2 \\ -2 & 0 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + z$	Project out y	$\begin{pmatrix} 2 & -2 \\ 0 & -2 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + x + z$	Project out y	$\begin{pmatrix} 2 & -2 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + x + y + z$	Contract	$\begin{pmatrix} 2 & -4 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 400 & 64 \\ 64 & 16 \end{pmatrix}$
$xyz + xy + xz$	Project out z	$\begin{pmatrix} 2 & 2 \\ 0 & -2 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + xz + x$	Project out z	$\begin{pmatrix} 2 & 2 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + xz + y$	Project out z	$\begin{pmatrix} 2 & -2 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + xz + x + y$	Project out z	$\begin{pmatrix} 2 & -2 \\ 0 & -2 \end{pmatrix}$	$\begin{pmatrix} 64 & 16 \\ 16 & 16 \end{pmatrix}$
$xyz + xy + xz + y + z$	Contract	$\begin{pmatrix} 2 & -4 \\ 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 400 & 64 \\ 64 & 16 \end{pmatrix}$
$xyz + xy + xz + x + y + z$	Project out x	$\begin{pmatrix} 0 & -2 \\ -2 & 2 \end{pmatrix}$	$\begin{pmatrix} 16 & 16 \\ 16 & 64 \end{pmatrix}$
$xyz + xy + xz + yz$	Contract	$\begin{pmatrix} 4 & -2 \\ -2 & 4 \end{pmatrix}$	$\begin{pmatrix} 400 & 256 \\ 256 & 400 \end{pmatrix}$
$xyz + xy + xz + yz + x$	Project out x	$\begin{pmatrix} 0 & 2 \\ 2 & -2 \end{pmatrix}$	$\begin{pmatrix} 16 & 16 \\ 16 & 64 \end{pmatrix}$
$xyz + xy + xz + yz + x + y$	Project out y	$\begin{pmatrix} 0 & 2 \\ -2 & 2 \end{pmatrix}$	$\begin{pmatrix} 16 & 16 \\ 16 & 64 \end{pmatrix}$
$xyz + xy + xz + yz + x + y + z$	Project out z	$\begin{pmatrix} 0 & -2 \\ -2 & -2 \end{pmatrix}$	$\begin{pmatrix} 16 & 16 \\ 16 & 64 \end{pmatrix}$

Proof. Renaming variables if necessary, we may assume that one of the terms of least degree greater than or equal to three in s is $x_1 \cdots x_\ell$ for some ℓ with $3 \leqslant \ell \leqslant k$. Let $c_4 = \cdots = c_\ell = 1$ and $c_{\ell+1} = \cdots = c_k = 0$ and let $s'(x_1, x_2, x_3) = s(x_1, x_2, x_3, c_4, \ldots, c_k)$ and $f'(x_1, x_2, x_3) = (-1)^{s'(x_1,x_2,x_3)}$.

The degree of s' is 3 since it has only three variables and includes exactly one term $x_1x_2x_3 1 \cdots 1 = x_1x_2x_3$. Therefore, #CSP($f'$) is #P-hard by the previous lemma.

It remains to show that #CSP(f') $\leqslant_T$ #CSP(f). To see this, let I' be any instance of #CSP(f'). We create an instance I'' of #CSP($\{f, \delta_0, \delta_1\}$) such that $Z(I'') = Z(I')$ as follows, and the result is then immediate from Lemma 13. Let $z_4, \ldots, z_k$ be new variables. Let I'' have the constraints $\delta_1(z_4), \ldots, \delta_1(z_\ell), \delta_0(z_{\ell+1}), \ldots, \delta_0(z_k)$ and, for each constraint $f'(y_1, y_2, y_3)$ in I', the constraint $f(y_1, y_2, y_3, z_4, \ldots, z_k)$. □

We now prove the main result of this section, namely that #CSP(f) is #P-hard if f is pure affine of degree at least three.

Proof of Lemma 22. Let $f(x_1, \ldots, x_k)$ be pure affine of degree at least three. Thus, we may write $f(\bar{x}) = w(-1)^{s(\bar{x})}g(\bar{x})$, where $w > 0$, $g \in \mathcal{P}_k$ is affine and $s \in \mathcal{P}_k$ is degree-minimized with respect to g and has degree at least three. By Lemma 16, we may assume that $w = 1$.

Since g is affine, we may write

$$f(\bar{x}) = (-1)^{s(\bar{x})} \prod_{i \in [1,m]} g_i(\bar{x}),$$

where each $g_i \in \mathcal{P}_k$ is linear. We show that f is #P-hard by induction on m. The base case, $m = 0$, is Lemma 24.

For the inductive step $m > 0$, we may assume without loss of generality that g_m depends on x_k. If $f(\bar{x}) \neq 0$, we must have $g_m(\bar{x}) = 1$ and, therefore, $x_k = g_m(\bar{x}) \oplus x_k \oplus 1$. Note that $g_m(\bar{x}) \oplus x_k \oplus 1$ does not depend on x_k. Let $g'_1, \ldots, g'_{m-1}, s' \in \mathcal{P}_{k-1}$ be the polynomials that result from substituting $g_m(\bar{x}) \oplus x_k \oplus 1$ for x_k in $g_1, \ldots, g_{m-1}$ and s, respectively.

Since $s'(\bar{x}) = s(\bar{x})$ whenever $f(\bar{x}) \neq 0$, we have $f(\bar{x}) = (-1)^{s'(\bar{x})} g(\bar{x})$. Because s is degree-minimized with respect to g, s' must have the same degree as s.

Let

$$f'(x_1, \ldots, x_{k-1}) = (-1)^{s'(\bar{x})} \prod_{i \in [1, m-1]} g'_i(\bar{x}).$$

Suppose that s' is not degree-minimized with respect to $g'(\bar{x}) = \prod_i g'_i(\bar{x})$. Then there is another polynomial s'' of strictly lower degree such that $f'(\bar{x}) = (-1)^{s''(\bar{x})} g'(\bar{x})$. But then, $f(\bar{x}) = (-1)^{s''(\bar{x})} g_m(\bar{x}) g'(\bar{x}) = (-1)^{s''(\bar{x})} g(\bar{x})$, contradicting degree-minimality of s. Therefore, s' is degree-minimized with respect to g'. Further, s and s' have the same degree, so #CSP(f') is #P-hard by the inductive hypothesis.

It remains to show that #CSP(f') $\leqslant_T$ #CSP(f). Let I' be an instance of #CSP(f') and let I be the instance of #CSP(f) that has a constraint $f(x_1, \ldots, x_{k-1}, x_C)$ for every constraint $C = f'(x_1, \ldots, x_{k-1})$ in I'. Then $Z(I) = Z(I')$ and we are done. □

6. High-degree product-type functions

We now construct the machinery for the remaining hard case: functions of product type of degree two or more that are not pure affine of degree two.

For any $\lambda \in \mathbb{Q}$, we write $\Theta_\lambda(x)$ for the function $\Theta_\lambda(0) = 1$, $\Theta_\lambda(1) = \lambda$. (In [8], these functions are written U_λ but we wish to avoid the potential for confusion with the functions $U_1, \ldots, U_k$ used to define a k-ary function of product type.)

The main result of this section is the following lemma.

Lemma 25. *Let $f \in \mathcal{F}_B$ be of product type of degree at least two. Then, #CSP($\{f, \Theta_\lambda\}$) is #P-hard for any positive rational $\lambda \neq 1$.*

If f is both of product type of degree two and pure affine of degree two, then #CSP(f) is computable in polynomial time by Lemma 10. In the following section, we will show that, for all other functions of product type of degree two or more, we have #CSP($\{f, \Theta_\lambda\}$) $\equiv_T$ #CSP(f) so Lemma 25 is sufficient for our needs, even though it appears, at first sight, to be weaker than the desired result.

As in the previous section, we first consider simplified cases.

Lemma 26. *Let f be of product type of degree at least two. There are non-zero rationals α and β such that #CSP(f') $\leqslant_T$ #CSP(f), where $f'(x, y) = (-1)^{xy} \Theta_\alpha(x) \Theta_\beta(y)$.*

Proof. Let $(-1)^{s(\bar{x})} U_1(x_1) \cdots U_k(x_k) g(\bar{x})$ be a normalized expression defining f.

We may assume, renaming variables if necessary, that

$$s(\bar{x}) = x_1 x_2 p(x_3, \ldots, x_k) + q(x_1, \ldots, x_k),$$

where p and q are polynomials in the stated variables, p is not identically zero and q contains no term that has $x_1 x_2$ as a factor. Let X be the set of variables on which s depends. Because $x_1 \in X$, there must be an assignment $\sigma : X \to \{0, 1\}$ such that $s(0, \sigma(x_2), \ldots, \sigma(x_k)) \neq s(1, \sigma(x_2), \ldots, \sigma(x_k))$. We may assume that $s(\sigma(\bar{x})) = 1$.

Now let Y be the set of variables on which g depends. Since the expression is normalized, at least one variable in each term $\chi_=(x_i, x_j)$ or $\chi_{\neq}(x_i, x_j)$ is determined and no determined variable appears in s. Therefore, we can extend σ to an assignment $\sigma' : X \cup Y \to \{0, 1\}$ such that $s(\sigma'(\bar{x})) = g(\sigma'(\bar{x})) = 1$.

Further, for every i with $x_i \in X \cup Y$, $U_i(0)$ and $U_i(1)$ are both non-zero. For each $x_i \notin (X \cup Y)$, we must have $U_i(0) \neq 0$ or $U_i(1) \neq 0$ or both; otherwise, f is identically zero (and, thus, of product type of degree zero). Therefore, we can extend σ' to an assignment $\sigma'' : \{x_1, \ldots, x_k\} \to \{0, 1\}$ such that $f(\sigma''(\bar{x})) \neq 0$.

Finally, suppose that $f'(x, y) = f(x, y, \sigma''(x_3), \ldots, \sigma''(x_k))$. Then clearly #CSP($f'$) $\leqslant_T$ #CSP($\{f, \delta_0, \delta_1\}$) so, by Lemma 13, #CSP($f'$) $\leqslant_T$ #CSP(f).

There are constants $w \in \mathbb{Q}$ and $a, b, c \in \{0, 1\}$ such that

$$\begin{aligned} f'(x, y) &= w(-1)^{xy+ax+by+c} U_1(x) U_2(y) \\ &= w(-1)^c (-1)^{xy} U'_1(x) U'_2(y), \end{aligned}$$

where $U'_1(x) = (-1)^{ax} U_1(x)$ and $U'_2(y) = (-1)^{by} U_2(y)$. Putting $\alpha = U'_1(1)/U'_1(0)$ and $\beta = U'_2(1)/U'_2(0)$, we have

$$f'(x, y) = w U'_1(0) U'_2(0) (-1)^c (-1)^{xy} \Theta_\alpha(x) \Theta_\beta(y)$$

and, by Lemma 16, we can discard the constant factor $w U'_1(0) U'_2(0) (-1)^c$. □

Lemma 27. *If $f(x, y) = (-1)^{xy} \Theta_\alpha(x) \Theta_\beta(y)$, where $\alpha \in \mathbb{Q} \setminus \{-1, 0, 1\}$, $\beta \in \mathbb{Q} \setminus \{0\}$, then #CSP($f$) is #P-hard.*

Proof. Let

$$A = \begin{pmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{pmatrix} \begin{pmatrix} 1 & \beta \\ \alpha & -\alpha\beta \end{pmatrix}$$

and let

$$B = (A^{\mathrm{T}}A)^{(2)} = \begin{pmatrix} 1+\alpha^2 & \beta(1-\alpha^2) \\ \beta(1-\alpha^2) & \beta^2(1+\alpha^2) \end{pmatrix}^{(2)}.$$

Since $\beta \neq 0$ and $\alpha^2 \neq 1$, every entry of B is positive. We have

$$|B| = (1+\alpha^2)^4\beta^4 - (1-\alpha^2)^4\beta^4 = 8\alpha^2\beta^4(1+\alpha^4) > 0\,.$$

Therefore, B has rank two and hence $\textsc{Eval}(B)$ is #P-hard by Lemma 19. By Lemmas 20 and 21, $\textsc{Eval}(B) \leqslant_{\mathrm{T}} \textsc{Eval}(A) \equiv_{\mathrm{T}}$ #CSP(f). □

We now prove Lemma 25, namely that, if f is of product type of degree at least two, then #CSP$(\{f, \Theta_\lambda\})$ is #P-hard for any positive rational $\lambda \neq 1$.

Proof of Lemma 25. Let $f \in \mathcal{F}_{\mathrm{B}}$ be of product type of degree at least 2 and let $\lambda \in \mathbb{Q}^{>0} \setminus \{1\}$. By Lemma 26 there are non-zero rational constants α and β such that #CSP$(g) \leqslant_{\mathrm{T}}$ #CSP(f), where

$$g(x, y) = (-1)^{xy}\Theta_\alpha(x)\Theta_\beta(y).$$

If at most one of α and β is 1 or -1, then #CSP(g) is #P-hard by Lemma 27 and we are done. Otherwise, we have $\alpha, \beta \in \{-1, 1\}$. Let

$$g'(x, y) = (-1)^{xy}\Theta_{\alpha\lambda}(x)\Theta_\beta(y).$$

#CSP(g') is #P-hard by Lemma 27, since $\alpha\lambda \notin \{-1, 0, 1\}$. It just remains to show that #CSP$(g') \leqslant_{\mathrm{T}}$ #CSP$(\{g, \Theta_\lambda\})$ but this is easy: given an instance of #CSP(g'), replace every constraint $g'(x, y)$ by the pair of constraints $g(x, y)$ and $\Theta_\lambda(x)$. □

7. Proving the dichotomy

We now have all the tools we need to prove the remaining side of the dichotomy, namely that, unless either every $f \in \Gamma$ is pure affine of degree at most two or every f is of product type of degree at most one, then #CSP(Γ) is #P-hard.

Lemma 28. *If $f \in \mathcal{F}_{\mathrm{B}}$ does not have affine support, then* #CSP(f) *is* #P*-hard.*

Proof. $f^2 \in \mathcal{F}_{\mathrm{B}}^{\geqslant 0}$ has the same support as f. By [8, Lemma 11], #CSP(f^2) is #P-hard and #CSP$(f^2) \leqslant_{\mathrm{T}}$ #CSP(f) by Lemma 17. □

Lemma 29. *If $f \in \mathcal{F}_{\mathrm{B}}$ is not of product type of degree at most one then the problem* #CSP$(\{f, \delta_0, \delta_1, \Theta_\lambda\})$ *is* #P*-hard for any positive rational $\lambda \neq 1$.*

Proof. If f is not of product type then, by Lemma 18, $f^2 \in \mathcal{F}_{\mathrm{B}}^{\geqslant 0}$ is also not of product type. By [8, Lemma 15], #CSP$(\{f^2, \delta_0, \delta_1, \Theta_\lambda\})$ is #P-hard for any positive, rational $\lambda \neq 1$ and the result follows by Lemma 17.

If f is of product type but of degree two or more, the result follows from Lemma 25. □

The next lemma corresponds to [8, Lemma 16] and its proof is based on the same idea as the proof there. The only difference is a slight adjustment to deal with mixed signs.

Lemma 30. *Suppose $f \in \mathcal{F}_{\mathrm{B}}$ is not pure affine of degree at most two and $g \in \mathcal{F}_{\mathrm{B}}$ is not of product type of degree at most one. Then* #CSP$(\{f, g, \delta_0, \delta_1\})$ *is* #P*-hard.*

Proof. Suppose f is not pure affine of degree at most two. If f does not even have affine support, we are done by Lemma 28 and, if f is pure affine of degree three or higher, we are done by Lemma 22. So we may assume that f is not pure affine. By Lemma 18, f^2 is also not pure affine and, by Lemma 17, it suffices to show that #CSP$(\{f^2, g, \delta_0, \delta_1\})$ is #P-hard.

Since f^2 has affine support but is not pure affine, there must be at least two positive values in its range. The proof now proceeds exactly as that of Lemma 16 in [8]. By using pinning and projection, we extract from f^2 a unary function Θ_λ for some positive rational $\lambda \neq 1$. The function Θ_λ is simulated by f^2 and we show hardness of #CSP$(\{f^2, g, \delta_0, \delta_1\})$ by reduction from #CSP$(\{g, \delta_0, \delta_1, \Theta_\lambda\})$, which is #P-hard by Lemma 29. We do not repeat the details here; refer to the proof in [8], starting with the second paragraph and noting that the function g referred to there is the function f^2 here. □

References

[1] A.A. Bulatov, M.E. Dyer, L.A. Goldberg, M. Jerrum, Personal communication.
[2] A.A. Bulatov, M. Grohe, The complexity of partition functions, Theoretical Computer Science 348 (2–3) (2005) 148–186.
[3] A.A. Bulatov, The complexity of the counting constraint satisfaction problem., in: 35th International Colloquium on Automata, Languages and Programming, ICALP 2008, Part 1, in: Lecture Notes in Computer Science, vol. 5125, Springer, 2008, pp. 646–661.
[4] J.-Y. Cai, P. Lu, M. Xia, The complexity of complex weighted boolean #CSP, Upcoming Journal submission (2009).
[5] N. Creignou, M. Hermann, Complexity of generalized satisfiability counting problems, Information and Computation 125 (1) (1996) 1–12.

[6] N. Creignou, S. Khanna, M. Sudan, Complexity Classifications of Boolean Constraint Satisfaction Problems, SIAM Press, 2001.
[7] M.E. Dyer, L.A. Goldberg, M. Jerrum, An approximation trichotomy for Boolean #CSP, 2007. http://arxiv.org/abs/0710.4272.
[8] M.E. Dyer, L.A. Goldberg, M. Jerrum, The complexity of weighted Boolean #CSP, SIAM Journal on Computing 38 (5) (2009) 1970–1986.
[9] M.E. Dyer, C. Greenhill, The complexity of counting graph homomorphisms, Random Structures and Algorithms 17 (3–4) (2000) 260–289.
[10] A. Ehrenfeucht, M. Karpinski, The computational complexity of (XOR, AND)-counting problems, Technical Report 8543-CS, University of Bonn, 1990. Available at http://citeseer.ist.psu.edu/ehrenfeucht90computational.html.
[11] T. Feder, M.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) (1999) 57–104.
[12] L.A. Goldberg, M. Grohe, M. Jerrum, M. Thurley, A complexity dichotomy for partition functions with mixed signs, in: 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, in: Dagstuhl Seminar Series, 2009, pp. 493–501.
[13] L.A. Goldberg, M. Jerrum, Inapproximability of the Tutte polynomial, Information and Computation 207 (7) (2008) 908–929.
[14] C. Greenhill, The complexity of counting colourings and independent sets in sparse graphs and hypergraphs, Computational Complexity 9 (1) (2000) 52–72.
[15] P. Hell, J. Nešetřil, Graph Homomorphisms, Oxford University Press, 2004.
[16] R. Ladner, On the structure of polynomial time reducibility, Journal of the ACM 22 (1) (1975) 155–171.
[17] R. Lide, H. Niederreiter, Finite Fields, 2nd ed., in: Encyclopedia of Mathematics and its Applications, vol. 20, Cambridge University Press, 1997.
[18] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[19] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006.
[20] T.J. Schaefer, The complexity of satisfaction problems, in: 10th ACM Symposium on Theory of Computing, ACM Press, 1978, pp. 216–226.
[21] L. Valiant, The complexity of enumeration and reliability problems, SIAM Journal on Computing 8 (3) (1979) 410–421.
[22] D. Welsh, Complexity: Knots, Colourings and Counting, Cambridge University Press, 1993.