Information and Computation 220–221 (2012) 1–14

Contents lists available at SciVerse ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

The complexity of approximating bounded-degree Boolean #CSP ☆

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

[a] School of Computing, University of Leeds, Leeds, LS2 9JT, UK
[b] Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK
[c] Department of Computer Science, University of Bristol, Merchant Venturers Building, Bristol, BS8 1UB, UK

ARTICLE INFO

Article history:
Received 7 February 2011
Revised 16 September 2011
Available online 5 October 2012

Keywords:
Counting constraint satisfaction problem
CSP
Approximation algorithm
Complexity

ABSTRACT

The degree of a CSP instance is the maximum number of times that any variable appears in the scopes of constraints. We consider the approximate counting problem for Boolean CSP with bounded-degree instances, for constraint languages containing the two unary constant relations $\{0\}$ and $\{1\}$. When the maximum allowed degree is large enough (at least 6) we obtain a complete classification of the complexity of this problem. It is exactly solvable in polynomial time if every relation in the constraint language is affine. It is equivalent to the problem of approximately counting independent sets in bipartite graphs if every relation can be expressed as conjunctions of $\{0\}$, $\{1\}$ and binary implication. Otherwise, there is no FPRAS unless $\mathbf{NP} = \mathbf{RP}$. For lower degree bounds, additional cases arise, where the complexity is related to the complexity of approximately counting independent sets in hypergraphs.

1. Introduction

In the constraint satisfaction problem (CSP), we seek to assign values from some domain to a set of variables, while satisfying given constraints on the combinations of values that certain tuples of the variables may take. Constraint satisfaction problems are ubiquitous in computer science, with close connections to graph theory, database query evaluation, type inference, satisfiability, scheduling and artificial intelligence [32,33,36]. CSP can also be reformulated in terms of homomorphisms between relational structures [26] and conjunctive query containment in database theory [32]. Weighted versions of CSP appear in statistical physics, where they correspond to partition functions of spin systems [44].

We give formal definitions in Section 2 but, for now, consider an undirected graph G and the CSP where the domain is $\{\text{red}, \text{green}, \text{blue}\}$, the variables are the vertices of G and the constraints specify that, for every edge $xy \in G$, x and y must be assigned different values. Thus, in a satisfying assignment, no two adjacent vertices are given the same colour: the CSP is satisfiable if, and only if, the graph is 3-colourable. As a second example, given a formula in 3-CNF, we can write a system of constraints over the variables, with domain $\{\text{true}, \text{false}\}$, that requires the assignment to each clause of the formula to satisfy at least one literal. Clearly, the resulting CSP is directly equivalent to the original satisfiability problem.

1.1. Decision CSP

In the *uniform constraint satisfaction problem*, we are given the set of constraints explicitly, as lists of allowable combinations for given tuples of the variables; these lists can be considered as relations over the domain. Since it includes problems

☆ The work described in this paper was partly supported by EPSRC Research Grant (refs EP/I011528/1 and EP/I012087/1) "Computational Counting".
* Corresponding author.
E-mail address: m.e.dyer@leeds.ac.uk (M. Dyer).

0890-5401/$ – see front matter
http://dx.doi.org/10.1016/j.ic.2011.12.007

such as 3-SAT and 3-COLOURABILITY, uniform CSP is **NP**-complete. However, uniform CSP also includes problems in **P**, such as 2-SAT and 2-COLOURABILITY, raising the natural question of what restrictions lead to tractable problems. It is natural to restrict either the form of the constraints or of the instances.

The most common restriction is to allow only certain fixed relations in the constraints. The list of allowed relations is known as the *constraint language* and we write CSP(Γ) for the so-called *non-uniform* CSP in which each constraint states that the values assigned to some tuple of variables must be a tuple in a specified relation in Γ.

The classic example of this is due to Schaefer [37]. Restricting to Boolean constraint languages (i.e., those with domain $\{0,1\}$), he showed that CSP(Γ) is in **P** if Γ is included in one of six classes and is **NP**-complete, otherwise. The Boolean case of CSP is often referred to as "generalized satisfiability" in the literature. More recently, Bulatov has produced a corresponding dichotomy for three-element domains [3].

Restricting to relations of fixed arity over arbitrary finite domains has also been studied in depth. In particular, requiring Γ to be a single binary relation gives the directed graph homomorphism problem, and the undirected graph homomorphism problem if the relation is also required to be symmetric. Hell and Nešetřil have shown that, for every symmetric binary relation E, CSP(E) is either in **P** or is **NP**-complete [28]. They conjecture that this holds for all binary relations.

In all the above cases, CSP(Γ) has been either in **P** or **NP**-complete and Feder and Vardi have conjectured that this holds for all Γ [26]. No such dichotomy can exist for the whole of **NP** because Ladner has shown that either **P** = **NP** or there is an infinite, strict hierarchy between the two [34]. However, a dichotomy for CSP is possible as there are problems in **NP**, such as graph Hamiltonicity and even connectedness, that cannot be expressed as CSP(Γ)[1] and Ladner's diagonalization does not seem to be expressible in CSP [26]. Resolving Hell and Nešetřil's conjecture for a class of simple acyclic digraphs would immediately resolve the CSP dichotomy [26], though recent work on the dichotomy has focused on methods from universal algebra—see, for example, [3,8] and the references there.

Allowing arbitrary constraint languages but restricting the form of the instances has also been studied. Dechter and Pearl [15] and Freuder [27] have shown that even uniform CSP is in **P** on instances of bounded tree width; see also [31]. Bounded tree width and other similar restrictions are generalized by the "guarded decompositions" of Cohen, Jeavons and Gyssens [9]. Restricting the degree of instances (the maximum number of times that each variable may appear in the scopes of constraints) is incomparable but not much is known in this case. In the non-uniform Boolean case, Dalmau and Ford have shown that, as long as Γ contains the relations $R_{\text{zero}} = \{0\}$ and $R_{\text{one}} = \{1\}$, CSP(Γ) for instances of degree at most three has the same complexity as the case with no degree restrictions [14]. The degree-2 case has not yet been completely classified, though it is known that degree-2 CSP(Γ) is as hard as general CSP(Γ) whenever Γ contains R_{zero} and R_{one} and some relation that is not a Δ-matroid [14,25].

1.2. Counting CSP

A generalization of the classical constraint satisfaction problem is to ask how many satisfying solutions there are, rather than just whether the constraints are satisfiable. This is referred to as the counting CSP problem, #CSP. Clearly, the decision problem is reducible to counting: if we can efficiently count the solutions, we can efficiently determine whether there is at least one. However, the converse does not hold: for example, there are well-known polynomial-time algorithms that determine whether a graph admits a perfect matching but it is #**P**-complete to count the perfect matchings, even in a bipartite graph [42].

The class #**P** can be considered to be the counting analogue of **NP**: it is defined as the class of functions f for which there is a nondeterministic, polynomial-time Turing machine that has exactly $f(x)$ accepting paths for every input x [41]. The counting version of any **NP** decision problem is easily seen to be in #**P**. Note that, although #**P** plays a similar role in the complexity of function problems to that of **NP** in decision problems, problems that are complete for #**P** under appropriate reductions are, under standard complexity-theoretic assumptions, considerably harder than **NP**-complete problems. Toda has shown that $\mathbf{P}^{\#\mathbf{P}}$ includes the whole of the polynomial hierarchy [40], but $\mathbf{P}^{\mathbf{NP}}$ is generally thought not to.

Although it is not known if there is a dichotomy for CSP, Bulatov has recently shown that, for every Γ, #CSP(Γ) is either computable in polynomial time or #**P**-complete [4]. Two of the present authors have since given an elementary proof of this result and also shown the dichotomy to be decidable [24]. However, it is not obvious how the methods of these results could be applied to bounded-degree #CSP.

So, although there is a full dichotomy for #CSP(Γ), results for restricted forms of constraint language are still of interest. For Boolean constraint languages, Creignou and Hermann have shown that only one of Schaefer's polynomial-time cases survives the transition to counting: #CSP(Γ) has a polynomial-time algorithm if every relation in Γ is affine (i.e., the solution set of a system of linear equations over GF_2) and is #**P**-complete, otherwise [12]. It is not surprising that there are fewer tractable cases—it is easy to arrange that every instance of CSP(Γ) be trivially satisfiable (say, by making the all-zeroes assignment satisfying), but the number of non-trivial solutions might be difficult to compute. Dyer, Goldberg and Jerrum [19] extended Creignou and Hermann's result to weighted Boolean #CSP. Cai, Lu and Xia [6,7] extended further to

[1] This follows from the observation that any set S of structures (e.g., graphs) that is definable in CSP has the property that, if $A \in S$ and there is a homomorphism $B \to A$, then $B \in S$; neither the set of Hamiltonian nor connected graphs has this property.

the case of complex weights and show that the dichotomy holds for the restriction of the problem in which instances have degree 3. Their result implies that the degree-3 problem $\#\mathrm{CSP}_3(\Gamma)$ ($\#\mathrm{CSP}(\Gamma)$ restricted to instances of degree 3) has a polynomial-time algorithm if every relation in Γ is affine and is #**P**-complete, otherwise.

The case where Γ contains a single symmetric, binary relation E corresponds exactly to the problem of counting the homomorphisms from an input graph to some fixed undirected graph H, also known as the counting H-colouring problem. Dyer and Greenhill have shown that $\#\mathrm{CSP}(\{E\})$ is in polynomial time if E is a complete relation or defines a complete bipartite graph and is #**P**-complete otherwise [23]. The dichotomy for directed acyclic graphs has been characterized by Dyer, Goldberg and Paterson [21] and, more recently, Cai and Chen have shown a dichotomy for all directed graphs, even in the presence of non-negative algebraic weights [5]. In contrast to the decision problem, it is not known whether a direct proof of the dichotomy for general directed graphs would yield an alternative proof of the dichotomy for arbitrary constraint languages.

Restricting the tree-width of instances has a dramatic effect. In the case of counting H-colourings, restricting the instance to be a graph of tree-width at most k makes the problem solvable in linear time for any graph H, a result due to Díaz, Serna and Thilikos [16]. This result follows immediately from Courcelle's theorem, which says that, if a decision problem is definable in monadic second-order logic (which H-colouring is, for any fixed H), then both it and the corresponding counting problem are computable in linear time [10,11]. However, invocations of Courcelle's theorem hide enormous constants in the notation $\mathcal{O}(n)$ (in this case, a tower of twos of height $|H|$), while the work of Díaz et al. not only yields practical constants but can also be applied to classes of instances where the tree-width is allowed to grow logarithmically with the order of the graph, rather than being constant.

1.3. Approximate counting

Since $\#\mathrm{CSP}(\Gamma)$ is very often #**P**-complete, approximation algorithms play an important role. The key concept is that of a *fully polynomial randomized approximation scheme* (FPRAS). This is a randomized algorithm for computing some function $f(x)$, which takes as input x, along with a constant $\epsilon > 0$, and computes a value Y such that $e^{-\epsilon} \leqslant Y/f(x) \leqslant e^{\epsilon}$ with probability at least $\frac{3}{4}$, in time polynomial in both $|x|$ and ϵ^{-1}. See Section 2.4 for details.

Dyer, Goldberg and Jerrum have classified the complexity of approximately computing $\#\mathrm{CSP}(\Gamma)$ for Boolean constraint languages [20]. When all relations in Γ are affine, $\#\mathrm{CSP}(\Gamma)$ can be computed exactly in polynomial time by the result of Creignou and Hermann discussed above [12]. Otherwise, if every relation in Γ can be defined by a conjunction of Boolean implications and pins (i.e., assertions of the form $v = 0$ or $v = 1$), then $\#\mathrm{CSP}(\Gamma)$ is as hard to approximate as the problem #BIS of counting independent sets in a bipartite graph; otherwise, $\#\mathrm{CSP}(\Gamma)$ is as hard to approximate as the problem #SAT of counting the satisfying truth assignments of a Boolean formula. Dyer, Goldberg, Greenhill and Jerrum have shown that the latter problem is complete for #**P** under appropriate approximation-preserving reductions (see Section 2.4) and has no FPRAS unless **NP** = **RP** [18], which is thought to be unlikely. The complexity of #BIS is currently open: there is no known FPRAS but it is not known to be #**P**-complete, either. #BIS is known to be complete with respect to approximation-preserving reductions in a logically-defined subclass of #**P** [18].

1.4. Our result

In this paper we consider the complexity of approximately solving Boolean #CSP problems when instances have bounded degree. Following Dalmau and Ford [14] and Feder [25] we consider the case in which $R_{\mathrm{zero}} = \{0\}$ and $R_{\mathrm{one}} = \{1\}$ are available. We proceed by showing that any Boolean relation that is not definable as a conjunction of ORs or NANDs can be used in low-degree instances to assert equalities between variables. Thus, we can side-step degree restrictions by replacing high-degree variables with distinct variables that are constrained to be equal, reducing to Dyer, Goldberg and Jerrum's trichotomy for Boolean #CSP without degree restrictions [20].

Our main result, Theorem 24, is a trichotomy for the case in which instances have maximum degree d for any $d \geqslant 6$. If every relation in Γ is affine then $\#\mathrm{CSP}_d(\Gamma \cup \{R_{\mathrm{zero}}, R_{\mathrm{one}}\})$ is solvable in polynomial time. Otherwise, if every relation in Γ can be defined as a conjunction of R_{zero}, R_{one} and binary implications, then $\#\mathrm{CSP}_d(\Gamma \cup \{R_{\mathrm{zero}}, R_{\mathrm{one}}\})$ is equivalent in approximation complexity to #BIS. Otherwise, it has no FPRAS unless **NP** = **RP**. Theorem 23 gives a partial classification of the complexity when $d < 6$. In the new cases that arise here, the complexity is given in terms of $\#w\text{-HIS}_d$, the complexity of counting independent sets in hypergraphs of degree at most d with hyper-edges of size at most w. The complexity of this problem is not fully understood. We explain what is known about it in Section 6.

1.5. Organization

The remainder of the paper is organized as follows. In Section 2, we define the basic notation, relational operations and hypergraph properties that we use, and formally define bounded-degree CSPs. In Section 3, we introduce the classes of relations that we will use throughout the paper and give some of their basic properties. A key tool in this type of work [7,25] is characterizing the ability of certain relations or sets of relations to assert equalities between variables: we show when this can be done in Section 4. The last piece of preparatory work is to show that every Boolean relation that cannot simulate equality in this way is definable by a conjunction of pins and either ORs or NANDs, which is done in Section 5. Our classification of the approximation complexity of bounded-degree Boolean counting CSPs follows, in Section 6.

2. Preliminaries

2.1. Basic notation

We write $\bar{a}$ for the tuple $\langle a_1, \ldots, a_r \rangle$, which we often shorten to $a_1 \ldots a_r$. We write a^r for the r-tuple $a \ldots a$ and $\bar{a}\bar{b}$ for the tuple formed from the elements of $\bar{a}$ followed by those of $\bar{b}$.

The *bit-wise complement* of a relation $R \subseteq \{0,1\}^r$ is the relation

$$\widetilde{R} = \{\langle a_1 \oplus 1, \ldots, a_r \oplus 1 \rangle \mid \bar{a} \in R\},$$

where $\oplus$ denotes addition modulo 2.

We say that a relation R is *ppp-definable*[2] in a relation R' and write $R \leqslant_{\mathrm{ppp}} R'$ if R can be obtained from R' by some sequence of the following operations:

- permutation of columns;
- pinning (taking sub-relations of the form $R_{i \mapsto c} = \{\bar{a} \in R \mid a_i = c\}$ for some i and some $c \in \{0,1\}$); and
- projection ("deleting the ith column" to give the relation $\{a_1 \ldots a_{i-1} a_{i+1} \ldots a_r \mid a_1 \ldots a_r \in R\}$).

The three p's in "ppp-definable" refer to the initial letters of the words permutation, pinning and projection. Allowing permutation of columns is just a notational convenience: it clearly adds no expressive power.

It is easy to see that $\leqslant_{\mathrm{ppp}}$ is a partial order on Boolean relations and that, if $R \leqslant_{\mathrm{ppp}} R'$, then R can be obtained from R' by first permuting the columns, then making some pins and then projecting.

We write $R_{\mathrm{zero}} = \{0\}$, $R_{\mathrm{one}} = \{1\}$, $R_{=} = \{00, 11\}$, $R_{\neq} = \{01, 10\}$, $R_{\mathrm{OR}} = \{01, 10, 11\}$, $R_{\mathrm{NAND}} = \{00, 01, 10\}$, $R_{\rightarrow} = \{00, 01, 11\}$ and $R_{\leftarrow} = \{00, 10, 11\}$. For $k \geqslant 2$, we write $R_{=,k} = \{0^k, 1^k\}$, $R_{\mathrm{OR},k} = \{0,1\}^k \setminus \{0^k\}$ and $R_{\mathrm{NAND},k} = \{0,1\}^k \setminus \{1^k\}$ (i.e., k-ary equality, OR and NAND, respectively).

We write $\mathrm{proj}_i\, R$ for the projection of R onto its ith column and $\mathrm{proj}_{i,j}\, R$ for the projection onto columns i and j.

2.2. Boolean constraint satisfaction problems

A *constraint language* is a set $\Gamma = \{R_1, \ldots, R_m\}$ of named Boolean relations. Given a set V of variables, a *constraint* over Γ is an expression $R(\bar{v})$ where $R \in \Gamma$ has arity r and $\bar{v} \in V^r$. Note that, if v and v' are variables, neither $v = v'$ nor $v \neq v'$ is a constraint, though of course $R_{=}(v, v')$ is a constraint if $R_{=} \in \Gamma$ and similarly for $R_{\neq}$. The *scope* of a constraint $R(\bar{v})$ is the tuple $\bar{v}$. Note that the variables in the scope of a constraint need not all be distinct.

An *instance* of the constraint satisfaction problem (CSP) over Γ is a set V of variables and a set C of constraints over Γ in the variables in V.

An *assignment* to a set V of variables is a function $\sigma : V \to \{0,1\}$ and it *satisfies* an instance (V, C) if $\langle \sigma(v_1), \ldots, \sigma(v_r) \rangle \in R$ for every constraint of the form $R(v_1, \ldots, v_r)$. Given an instance I of some CSP, we write $Z(I)$ for the number of satisfying assignments.

We are interested in the counting CSP problem #CSP(Γ) (parameterized by Γ), defined as follows:

Input: an instance $I = (V, C)$ of CSP over Γ.
Output: $Z(I)$.

The *degree* of an instance is the greatest number of times any variable appears among its constraints. Note that the variable v appears twice in the constraint $R(v, v)$. Our specific interest in this paper is in classifying the complexity of bounded-degree counting CSPs. For a constraint language Γ and a positive integer d, define $\#\mathrm{CSP}_d(\Gamma)$ to be the restriction of #CSP(Γ) to instances of degree at most d.

We can deal with instances of degree 1 immediately.

Theorem 1. *For any Γ, $\#\mathrm{CSP}_1(\Gamma) \in$* **FP**.

Proof. Because each variable appears at most once, the constraints are independent. Each constraint $R(v_1, \ldots, v_r)$ can be satisfied in $|R|$ ways and any variable that does not appear in a constraint can take value either 0 or 1. The total number of assignments is the product of the number of ways each constraint can be satisfied, times 2^k, where k is the number of unconstrained variables. $\square$

A key technique in proving hardness results for #CSP and related problems is *pinning* [12,14,19,20,23,25]. We write $R_{\mathrm{zero}} = \{0\}$ and $R_{\mathrm{one}} = \{1\}$ for the two unary relations that contain only zero and one, respectively. We refer to constraints in R_{zero} and R_{one} as *pins* and we say that the single variable in the scope of a pin is *pinned*. To make notation easier,

[2] This should not be confused with the concept of primitive positive definability (pp-definability) which appears in algebraic treatments of CSP and #CSP, for example in the work of Bulatov [4].

we will sometimes write constraints using constants instead of explicit pins. That is, we will write constraints of the form $R(x_1,\ldots,x_r)$ where each x_i is either a variable from V or a constant 0 or 1 (again, the x_i need not be distinct). Such a constraint can always be rewritten as a set of "proper" constraints by replacing each instance of a constant 0 or 1 with a fresh variable v and introducing the appropriate constraint $R_{\mathrm{zero}}(v)$ or $R_{\mathrm{one}}(v)$. Note that every variable introduced in this way appears exactly twice in the resulting instance so if the degree of the CSP instance is at least two, the transformation does not increase the instance's degree. We let Γ_{pin} denote the constraint language $\{R_{\mathrm{zero}}, R_{\mathrm{one}}\}$.

When there are no degree bounds, adding pinning does not affect complexity results for either the exact or approximate version of #CSP. In the exact case, the addition of pinning does not affect the structural properties that determine the complexity of #CSP(Γ) [24] whereas, for approximation on the Boolean domain, there are reductions of the appropriate kind from #CSP($\Gamma \cup \Gamma_{\mathrm{pin}}$) to #CSP($\Gamma$) [19,20]. However, these reductions increase the degree of variables so are not applicable in our setting. In order to make progress, we follow earlier work on degree-bounded CSP [14,25] and assume that pinning is available in constraint languages. This plays a significant role in Section 4.

2.3. Hypergraphs

A *hypergraph* $H = (V, E)$ consists of a set $V = V(H)$ of vertices and a set $E = E(H) \subset \mathcal{P}(V)$ of non-empty *hyper-edges*. The *degree* of a vertex $v \in V(H)$ is the number $d(v)$ of hyper-edges it participates in: $d(v) = |\{e \in E(H) \mid v \in e\}|$. The degree of a hypergraph is the maximum degree of its vertices. If $w = \max\{|e| \mid e \in E(H)\}$, we say that H has *width* w.

An *independent set* in a hypergraph H is a set $S \subseteq V(H)$ such that $e \nsubseteq S$ for every $e \in E(H)$. Notice that we may have more than one vertex of a hyper-edge in an independent set, so long as at least one vertex of each hyper-edge is omitted.

We write #w-HIS for the following problem:

Input: a width-w hypergraph H
Output: the number of independent sets in H

and #w-HIS$_d$ for the following problem:

Input: a width-w hypergraph H of degree at most d
Output: the number of independent sets in H.

2.4. Approximation complexity

A *randomized approximation scheme* (RAS) for a function $f : \Sigma^* \to \mathbb{N}$ is a probabilistic Turing machine that takes as input a pair $(x, \epsilon) \in \Sigma^* \times (0,1)$, and produces, on an output tape, an integer random variable Y satisfying the condition $\Pr(e^{-\epsilon} \leqslant Y/f(x) \leqslant e^{\epsilon}) \geqslant \frac{3}{4}$.[3] A *fully polynomial randomized approximation scheme* (*FPRAS*) is a RAS that runs in time polynomial in both $|x|$ and ϵ^{-1}.

To compare the complexity of approximate counting problems, we use the AP-reductions of [18]. Suppose that f and g are functions from some input domain Σ^* to the natural numbers and we wish to compare the complexity of approximately computing them. An *approximation-preserving* reduction from f to g is a probabilistic oracle Turing machine M whose input is a pair $(x, \epsilon) \in \Sigma^* \times (0,1)$, and which satisfies the following three conditions: (i) every oracle call made by M is of the form (w, δ) where $w \in \Sigma^*$ is an instance of g and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \leqslant \mathrm{poly}(|x|, \epsilon^{-1})$; (ii) M is a randomized approximation scheme for f whenever the oracle is a randomized approximation scheme for g; and (iii) the running time of M is polynomial in $|x|$ and ϵ^{-1}.

If there is an approximation-preserving reduction from f to g, we write $f \leqslant_{\mathrm{AP}} g$ and say that f is *AP-reducible* to g. If g has an FPRAS then so does f. If $f \leqslant_{\mathrm{AP}} g$ and $g \leqslant_{\mathrm{AP}} f$ then we say that f and g are *AP-interreducible* and write $f \equiv_{\mathrm{AP}} g$.

AP-reductions are well-suited to approximate counting problems. The class of problems admitting an FPRAS is closed under these reductions and a Ladner-like hierarchy of AP-interreducible approximation problems has been shown to exist [1]. Further, the intuition that the counting version of an **NP**-complete problem should be #**P**-complete is a theorem if #**P**-completeness is defined with respect to AP-reductions [18] but is not known to hold for other classes of reduction, such as Simon's parsimonious reductions [38] and polynomial-time Turing reductions.

3. Classes of relations

A relation $R \subseteq \{0,1\}^r$ is *affine* if it is the set of solutions to some system of linear equations over GF_2. That is, there is a set Σ of equations in variables $x_1,\ldots,x_r$ where each equation has the form $\bigoplus_{i\in I} x_i = c$, where $\oplus$ denotes addition modulo 2, $I \subseteq [1,r]$ and $c \in \{0,1\}$, and we have $\bar{a} \in R$ if, and only if, the assignment $x_1 \mapsto a_1, \ldots, x_r \mapsto a_r$ satisfies every equation in Σ. Note that the empty relation is defined by the equation $0 = 1$ (or, more formally, $\bigoplus_{i\in\emptyset} = 1$) and the complete relation $\{0,1\}^r$ is defined by the empty set of equations. If a variable x_i occurs in an equation of the form $x_i = c$, we say that it is *pinned to c*.

[3] The choice of the value $\frac{3}{4}$ is inconsequential: the same class of problems has an FPRAS if we choose any probability $\frac{1}{2} < p < 1$ [29].

3.1. OR-*conj,* NAND-*conj, IM-conj and normalized formulae*

Let OR-conj be the set of Boolean relations that are defined by conjunctions of pins and ORs of any arity and let NAND-conj be the set of Boolean relations definable by conjunctions of pins and NANDs (i.e., negated conjunctions) of any arity. For example, the 8-ary relation defined by the formula

$$(x_1 = 0) \wedge (x_2 = 1) \wedge \mathrm{OR}(x_3, x_4, x_5, x_6) \wedge \mathrm{OR}(x_5, x_8)$$

is in OR-conj. (Note, also, that it does not constrain the variable x_7.) We say that one of the defining formulae of these relations is *normalized* if

- no pinned variable appears in any OR or NAND,
- the arguments of each individual OR and NAND are distinct,
- every OR or NAND has at least two arguments, and
- no OR or NAND's arguments are a subset of any other's.

Note that the formula in the example above is normalized.

Lemma 2. *Every* OR-*conj* (*respectively,* NAND-*conj*) *relation is defined by a unique normalized formula.*

Proof. We show the result for OR-conj relations; the case for NAND-conj is similar.

Let R be an OR-conj relation defined by the formula ϕ. The second and subsequent occurrences of any variable within a single clause can be deleted. Any clause that contains a variable pinned to one can be deleted; any variable that is pinned to zero can be deleted from any clause in which it appears. The disjunction $\mathrm{OR}(x)$ is equivalent to pinning x to one. If ϕ contains a clause that is a subset of another, any assignment that satisfies the smaller clause necessarily satisfies the latter, which can, therefore, be deleted. This establishes that every OR-conj relation is defined by at least one normalized formula.

To prove uniqueness, suppose that $R \subseteq \{0,1\}^r$ is defined by the normalized formulae ϕ and ψ. The two formulae must obviously pin the same variables and we may assume that none are pinned. Consider any clause in ϕ, which we may assume, without loss of generality, to be $\mathrm{OR}(x_1, \ldots, x_k)$. Since no clause of ϕ is a subset of $\{x_1, \ldots, x_k\}$, every other clause must include at least one variable from $x_{k+1}, \ldots, x_r$ and, therefore, $0^{k-1}1^{r-k+1}$ satisfies ϕ and 0^k1^{r-k} does not.

Now, suppose that this clause does not appear in ψ. There are two cases. If ψ contains a clause whose variables are a subset of $\{x_1, \ldots, x_k\}$, which we may assume, without loss of generality, to be $\mathrm{OR}(x_1, \ldots, x_\ell)$ for some $\ell < k$, then ψ is not satisfied by $0^{k-1}1^{r-k+1}$. Otherwise, every clause of ψ contains at least one variable from $x_{k+1}, \ldots, x_r$, so 0^k1^{r-k} satisfies ψ. In either case, ϕ and ψ define different relations. It follows that every clause that appears in ϕ must also appear in ψ. By symmetry, every clause that appears in ψ must appear in ϕ so the two formulae are identical. $\square$

Given the uniqueness of defining normalized formulae, we define the *width* of an OR-conj or NAND-conj relation R to be $\mathrm{width}(R)$, the greatest number of arguments to any of the ORs or NANDs in the normalized formula that defines it. Note that, from the definition of normalized formulae, there are no relations of width 1. However, a conjunction of pins can be seen as an OR-conj formula with no ORs, i.e., of width 0: such a formula defines the complete relation, possibly padded with some constant columns. A conjunction of pins is also a NAND-conj formula with no NANDs so we will usually just refer to these relations as "relations of width 0." We define the width of an OR-conj or NAND-conj constraint language to be the greatest width of the relations within it.

We define IM-conj to be the class of relations defined by a conjunction of pins and (binary) implications. This class is called IM_2 in [20]. Say that a conjunction of pins and implications is *normalized* if no pinned variable appears in an implication and every implication has distinct arguments.

Lemma 3. *Every relation in IM-conj is defined by a normalized formula.*

Proof. Let $R \in$ IM-conj be defined by the formula ϕ. Any implication $x \to x$ can be deleted as it does not constrain the value of x. If the variable y is pinned to zero then any implication $y \to z$ can be deleted and any implication $z \to y$ can be replaced by pinning z to zero. If y is pinned to one, $y \to z$ can be replaced by pinning z to one and $z \to y$ can be deleted. Iterating, we can remove all implications involving pinned variables. $\square$

Note that, in contrast to normalized OR-conj and NAND-conj formulae, normalized IM-conj formulae are not necessarily unique. For example, the following three normalized formulae all define the same relation:

$$x \to y \wedge y \to z \wedge z \to x,$$

$$x \to z \wedge z \to y \wedge y \to x,$$

$$x \to y \wedge y \to x \wedge x \to z \wedge z \to x.$$

3.2. ppp-Defining Boolean connectives

Lemma 4. *If $R \in$ IM-conj is not affine, then $R_\rightarrow \leqslant_{\mathrm{ppp}} R$.*

Proof. Let $R \in$ IM-conj be defined by the normalized formula ϕ. If there are variables $x_1, \ldots, x_r$ such that ϕ contains the implications $x_1 \rightarrow x_2, \ldots, x_{r-1} \rightarrow x_r$ and $x_r \rightarrow x_1$ then, in any satisfying assignment for ϕ, the variables $x_1, \ldots, x_r$ must take the same value. Hence, we may assume that, if ϕ contains such a cycle of implications, it also contains $x_i \rightarrow x_j$ for every distinct pair $x_i, x_j \in \{x_1, \ldots, x_r\}$.

There are two cases. First, if ϕ is symmetric (in the sense that, for every implication $x \rightarrow y$ in ϕ, the formula also contains $y \rightarrow x$) then ϕ is equivalent to a conjunction of pins and equalities between variables, so R is affine. Otherwise, there must be at least one pair of variables such that $x \rightarrow y$ is a conjunct of ϕ but $y \rightarrow x$ is not. We ppp-define implication by pinning to zero every unpinned variable v_1 such that there is a chain of implications $v_1 \rightarrow v_2, \ldots, v_{r-1} \rightarrow v_r$, $v_r \rightarrow x$ and pinning to one every other unpinned variable apart from x and y. Finally, project out the $r-2$ constant columns. □

Lemma 5. *If $R \in$ OR-conj has width w, then $R_{\mathrm{OR},2}, \ldots, R_{\mathrm{OR},w} \leqslant_{\mathrm{ppp}} R$. Similarly, if $R \in$ NAND-conj has width w, then $R_{\mathrm{NAND},2}, \ldots, R_{\mathrm{NAND},w} \leqslant_{\mathrm{ppp}} R$.*

Proof. Let $R \in$ OR-conj have arity r and width w. Let R be defined by the normalized formula ϕ which, without loss of generality, we may assume to contain the clause $\mathrm{OR}(x_1, \ldots, x_w)$. Since ϕ is normalized, every other clause must contain at least one variable from $x_{w+1}, \ldots, x_r$. For any k with $2 \leqslant k \leqslant w$, we can ppp-define $R_{\mathrm{OR},k}$ by pinning $x_{k+1}, \ldots, x_w$ to zero and pinning $x_{w+1}, \ldots, x_r$ to one.

The proof for $R \in$ NAND-conj is similar. □

3.3. Characterizations

The following proposition establishes a duality between OR-conj and NAND-conj relations. Whenever we say that R is OR-conj or NAND-conj, it is equivalent to say that R or $\widetilde{R}$ is OR-conj, where $\widetilde{R}$ is the bit-wise complement of R, as defined in Section 2.1. Of course, it is also equivalent to say that R or $\widetilde{R}$ is NAND-conj.

Proposition 6. *A relation $R \subseteq \{0,1\}^r$ is in OR-conj if, and only if, $\widetilde{R} \in$ NAND-conj.*

Proof. Suppose R is defined by the normalized formula

$$P \wedge \bigwedge_{1 \leqslant j \leqslant m} \bigvee_{i \in I_j} x_i,$$

where P is a conjunction of pins and $I_1, \ldots, I_m \subseteq [1, r]$. Then $\widetilde{R}$ is defined by the formula

$$P' \wedge \bigwedge_{1 \leqslant j \leqslant m} \bigvee_{i \in I_j} \neg x_i,$$

where P' is the conjunction of pins with the opposite values to those in P. This formula is equivalent to

$$P' \wedge \bigwedge_{1 \leqslant j \leqslant m} \neg \bigwedge_{i \in I_j} x_i,$$

which is a NAND-conj formula, as required. The argument is reversible. □

Given tuples $\bar{a}, \bar{b} \in \{0,1\}^r$, we write $\bar{a} \leqslant \bar{b}$ if $a_i \leqslant b_i$ for all $i \in [1, r]$. If $\bar{a} \leqslant \bar{b}$ and $\bar{a} \neq \bar{b}$, we write $\bar{a} < \bar{b}$. We say that a relation $R \subseteq \{0,1\}^r$ is *monotone* if, whenever $\bar{a} \in R$ and $\bar{a} \leqslant \bar{b}$, then $\bar{b} \in R$. We say that R is *antitone* if, whenever $\bar{a} \in R$ and $\bar{b} \leqslant \bar{a}$, then $\bar{b} \in R$. That is, changing zeroes to ones in a tuple in a monotone relation gives another tuple in the relation; similarly, antitone relations are preserved by changing ones to zeroes. It is easy to see that R is monotone if, and only if, $\widetilde{R}$ is antitone. We say that a relation is *pseudo-monotone* (respectively, *pseudo-antitone*) if its restriction to non-constant columns is monotone (respectively, antitone). The following is a simple consequence of results in [30, Section 7.1.1].

Proposition 7. *A relation $R \subseteq \{0,1\}^r$ is in OR-conj (respectively, NAND-conj) if, and only if, it is pseudo-monotone (respectively, pseudo-antitone).*

4. Simulating equality

An important ingredient in bounded-degree dichotomy theorems [7,25] is showing how to express equality using constraints from a constraint language that does not necessarily include the equality relation. In this section, we give the definitions that we need and some results about when equality can be expressed in our setting.

Recall that, for all integers $k \geqslant 2$, $R_{=,k}$ is the k-ary equality relation $\{0^k, 1^k\}$. We say that a constraint language Γ *simulates* $R_{=,k}$ if, for some $\ell \geqslant k$ there is an integer $m \geqslant 1$ and a $(\Gamma \cup \Gamma_{\text{pin}})$-CSP instance I with variables $x_1, \dots, x_\ell$ and such that I has exactly m satisfying assignments σ with $\sigma(x_1) = \cdots = \sigma(x_k) = 0$, exactly m with $\sigma(x_1) = \cdots = \sigma(x_k) = 1$ and no other satisfying assignments. If, further, the degree of I is d and the degree of each variable $x_1, \dots, x_k$ is at most $d-1$, we say that Γ *simulates* $R_{=,k}$ *with* d *variable repetitions* or, for brevity, that Γ d-*simulates* $R_{=,k}$. We say that Γ d-*simulates equality* if it d-simulates $R_{=,k}$ for all $k \geqslant 2$. If only one relation R is involved in the simulation, we drop the curly brackets and say that R, rather than $\{R\}$, d-simulates equality.

The point of this slightly strange definition is that, if Γ d-simulates equality, we can express the constraint $y_1 = \cdots = y_k$ in $\Gamma \cup \Gamma_{\text{pin}}$ and then use each y_i in one further constraint, while still having an instance of degree d. The variables $x_{k+1}, \dots, x_\ell$ in the definition function as auxiliary variables and do not appear in any other constraint. This means that, if the variable y occurs $k > d$ times in some instance, we can replace the successive occurrences with distinct variables $y_1, \dots, y_k$ that are constrained to be equal, giving an equivalent instance of degree at most d.

Concepts similar to simulation have been used before, such as "perfect implementation" [13] and "implementation" [20]. The difference is that our setting demands degree bounds on the constraints used in simulation and, for counting, we need to preserve the number of satisfying assignments (at least, up to some constant multiple) not just the existence of satisfying assignments.

Proposition 8. *If Γ d-simulates equality, then* $\#\text{CSP}(\Gamma) \leqslant_{\text{AP}} \#\text{CSP}_d(\Gamma \cup \Gamma_{\text{pin}})$.

Proof. Let I be an instance of $\#\text{CSP}(\Gamma)$. We produce a new CSP instance I' over the constraint language Γ augmented with $R_{=,i}$ constraints for certain values of i as follows. For each variable x that appears $k > d$ times in I, replace the occurrences with new variables $x_1, \dots, x_k$ and add the constraint $R_{=,k}(x_1, \dots, x_k)$. Clearly, $Z(I') = Z(I)$.

Note that every variable in I' either occurs exactly once in an equality constraint (one of the form $R_{=,i}(\bar{x})$) and exactly once in a Γ-constraint or occurs in no equality constraints and at most d times in Γ-constraints. Since Γ d-simulates equality, we can replace the equality constraints with $(\Gamma \cup \Gamma_{\text{pin}})$-constraints, using fresh auxiliary variables for each equality, to give an instance I'' of $\#\text{CSP}(\Gamma \cup \Gamma_{\text{pin}})$ with degree d. There is some constant m, depending only on the number and arities of the equality constraints in I', such that $Z(I'') = mZ(I')$. Since m can be computed in polynomial time, we have an AP-reduction. $\square$

Lemma 9. *Let $R \subseteq \{0,1\}^r$. If $R_= \leqslant_{\text{ppp}} R$, $R_{\neq} \leqslant_{\text{ppp}} R$ or $R_{\rightarrow} \leqslant_{\text{ppp}} R$, then R 3-simulates equality.*

Note that, if $R_{\leftarrow} \leqslant_{\text{ppp}} R$ then $R_{\rightarrow} \leqslant_{\text{ppp}} R$, also.

Proof of Lemma 9. For each $k \geqslant 2$, we show how to 3-simulate $R_{=,k}$. We may assume without loss of generality that the ppp definition of $R_=$, $R_{\neq}$ or $R_{\rightarrow}$ from R involves applying the identity permutation to the columns, pinning columns 3 to $3+p-1$ inclusive to zero, pinning columns $3+p$ to $3+p+q-1$ inclusive to one (that is, pinning $p \geqslant 0$ columns to zero and $q \geqslant 0$ to one) and then projecting away all but the first two columns.

Suppose first that $R_= \leqslant_{\text{ppp}} R$ or $R_{\rightarrow} \leqslant_{\text{ppp}} R$. R must contain $\alpha \geqslant 1$ tuples that begin 000^p1^q, $\beta \geqslant 0$ that begin 010^p1^q and $\gamma \geqslant 1$ that begin 110^p1^q, with $\beta = 0$ unless we are ppp-defining $R_{\rightarrow}$.

We consider, first, the case where $\alpha = \gamma$, and show that we can 3-simulate $R_{=,k}$, expressing the constraint $R_{=,k}(x_1, \dots, x_k)$ with the constraints

$$R\big(x_1x_20^p1^q*\big), R\big(x_2x_30^p1^q*\big), \dots, R\big(x_{k-1}x_k0^p1^q*\big), R\big(x_kx_10^p1^q*\big),$$

where $*$ denotes a fresh $(r-2-p-q)$-tuple of variables in each constraint. This set of constraints is equivalent to either $x_1 = \cdots = x_k = x_1$ or $x_1 \rightarrow \cdots \rightarrow x_k \rightarrow x_1$ so, in either case, constrains the variables $x_1, \dots, x_k$ to have the same value, as required. Every variable appears at most twice and there are α^k solutions to these constraints that put $x_1 = \cdots = x_k = 0$, the same number with $x_1 = \cdots = x_k = 1$ and no other solutions. Therefore, R 3-simulates $R_{=,k}$, as required.

We now show, by induction on r, the arity of R, that we can 3-simulate $R_{=,k}$ even if α is not necessarily equal to γ. For the base case, $r = 2$, we have $\alpha = \gamma = 1$ and we are done. For the inductive step, let $r > 2$ and assume, without loss of generality that $\alpha > \gamma$ (we are already done if $\alpha = \gamma$ and the case $\alpha < \gamma$ is symmetric). In particular, we have $\alpha \geqslant 2$, so there are distinct tuples $000^p1^q\bar{a}$ and $000^p1^q\bar{b}$ in R. R also contains a tuple $110^p1^q\bar{c}$. Choose j such that $a_j \neq b_j$. Pinning the $(2+p+q+j)$th column of R to c_j and projecting out the resulting constant column gives a relation of arity $r-1$ that still contains at least one tuple beginning 000^p1^q and at least one beginning 110^p1^q: by the inductive hypothesis, this relation 3-simulates $R_{=,k}$.

Finally, we consider the case that $R_{\neq} \leqslant_{\mathrm{ppp}} R$. R contains $\alpha \geqslant 1$ tuples beginning 010^p1^q and $\beta \geqslant 1$ beginning 100^p1^q and no other tuples. We express the constraint $R_{=,k}(x_1, \ldots, x_k)$ by introducing fresh variables $y_1, \ldots, y_k$ and using the constraints

$$\begin{aligned}
&R\big(x_1 y_1 0^p 1^q *\big), R\big(y_1 x_2 0^p 1^q *\big),\\
&R\big(x_2 y_2 0^p 1^q *\big), R\big(y_2 x_3 0^p 1^q *\big),\\
&\quad\vdots\\
&R\big(x_{k-1} y_{k-1} 0^p 1^q *\big), R\big(y_{k-1} x_k 0^p 1^q *\big),\\
&R\big(x_k y_k 0^p 1^q *\big), R\big(y_k x_1 0^p 1^q *\big),
\end{aligned}$$

where $*$ denotes a fresh $(r-2-p-q)$-tuple of variables in each constraint, as before. There are $\alpha^k\beta^k$ solutions when $x_1 = \cdots = x_k = 0$ (and $y_1 = \cdots = y_k = 1$) and $\beta^k\alpha^k$ solutions when the x's are 1 and the y's are 0. There are no other solutions and no variable is used more than twice. $\square$

The following technical lemma and the definitions that support it are used only to prove Lemma 11. For $c \in \{0,1\}$, an r-ary relation is *c-valid* if it contains the tuple c^r. Given a relation $R \subseteq \{0,1\}^r$, a tuple $\bar{a} \in R$ that contains both zeroes and ones and a constant $c \in \{0,1\}$, let $R_{\bar{a},c}$ be the result of pinning the set of columns $\{i \mid a_i = c\}$ to c and then projecting out those columns. Observe that $R_{\bar{a},c}$ is always $(1-c)$-valid (because it contains the projection of $\bar{a}$) and is c-valid if R is (because then it contains the projection of c^r).

Lemma 10. *Let $r \geqslant 3$ and let $R_{=,r} \subsetneq R \subsetneq \{0,1\}^r$. There are $\bar{a} \in R$ and $c \in \{0,1\}$ such that $R_{\bar{a},c}$ is not complete.*

Proof. Suppose there is a tuple $\bar{a} \in R \setminus \{0^r\}$ such that changing some zero in $\bar{a}$ to a one gives a tuple $\bar{a}' \notin R$. Then $R_{\bar{a},1}$ does not contain the relevant projection of $\bar{a}'$ and we are done. Similarly, if there is a tuple $\bar{b} \in R \setminus \{1^r\}$ that leaves R by changing some one to a zero, then $R_{\bar{b},0}$ is not complete. If no such tuple exists, then either $R = \{0,1\}^r$ or $R = R_{=,r}$, violating our assumptions. $\square$

Lemma 11. *Let $r \geqslant 2$ and let $R \subset \{0,1\}^r$ be 0- and 1-valid but not complete. Then R 3-simulates equality.*

Proof. We show by induction on r that either $R_=$ or $R_\rightarrow$ is ppp-definable in R, and the result follows by Lemma 9.

In the case $r = 2$, R is either $R_=$, $R_\rightarrow$ or $R_\leftarrow$. For $r \geqslant 3$, if $R = R_{=,r}$ then $\mathrm{proj}_{1,2} R = R_=$. Otherwise, by Lemma 10, there is some $\bar{a} \in R$ and $c \in \{0,1\}$ such that $R_{\bar{a},c}$ is not complete. Since $R_{\bar{a},c} \leqslant_{\mathrm{ppp}} R$ and is 0- and 1-valid, we are done by the inductive hypothesis. $\square$

We will next show that, if binary OR is ppp-definable in R and binary NAND in R', then the constraint language $\{R, R'\}$ 3-simulates equality (R and R' need not be distinct). To do this, we will use the following sets of constraints, ξ_k, for $k \geqslant 2$:

$$\xi_k = \big\{R_{\mathrm{OR}}(x_i, y_i) \bigm| 1 \leqslant i \leqslant k\big\} \cup \big\{R_{\mathrm{NAND}}(y_i, x_{i+1}) \bigm| 1 \leqslant i < k\big\} \cup \big\{R_{\mathrm{NAND}}(y_k, x_1)\big\}.$$

The key point about these constraints is that they show that any language that contains R_{OR} and R_{NAND} 3-simulates equality.

Lemma 12. *An assignment σ to $\{x_1, \ldots, x_k, y_1, \ldots, y_k\}$ satisfies all constraints in ξ_k if, and only if, $\sigma(x_1) = \cdots = \sigma(x_k) \neq \sigma(y_1) = \cdots = \sigma(y_k)$.*

Proof. It is easy to check that assignments of the given type satisfy ξ_k. Conversely, suppose that σ satisfies ξ_k.

If $\sigma(x_1) = 0$, we have $\sigma(y_1) = 1$ because $R_{\mathrm{OR}}(x_1, y_1)$ is satisfied and we must have $\sigma(x_2) = 0$ because $R_{\mathrm{NAND}}(y_1, x_2)$ is satisfied. By a trivial induction, $\sigma(x_i) = 0$ and $\sigma(y_i) = 1$ for all i.

Otherwise, $\sigma(x_1) = 1$. If $\sigma(x_i) = 0$ for any $i > 1$ then, by the same argument as above, $\sigma(x_i) = 0$ for all $i \in [1,k]$, contradicting the assumption that $\sigma(x_1) = 1$. Therefore, $\sigma(x_i) = 1$ for all i. To satisfy the constraints $R_{\mathrm{NAND}}(y_i, x_{i+1})$, we must have $\sigma(y_i) = 0$ for all i. $\square$

We now show that, in fact, we do not need to have R_{OR} and R_{NAND} in our constraint language Γ: it suffices to be able to ppp-define them from relations in Γ.

Lemma 13. *If $R_{\mathrm{OR}} \leqslant_{\mathrm{ppp}} R$ and $R_{\mathrm{NAND}} \leqslant_{\mathrm{ppp}} R'$ then $\{R, R'\}$ 3-simulates equality.*

Proof. Suppose first that R and R' are two distinct relations. We may assume, as in the proof of Lemma 9, that the ppp definition of R_{OR} from R involves performing some permutation and projecting to the first two columns after pinning the next p columns to zero and the q columns after that to one. We may suppose further that it is not possible to pin any more columns of R and still ppp-define R_{OR}. Without loss of generality, we may assume the permutation to be the identity.

Under these assumptions, R contains $\alpha \geqslant 1$ tuples beginning 010^p1^q, $\beta \geqslant 1$ tuples beginning 100^p1^q and $\gamma \geqslant 1$ tuples beginning 110^p1^q, but none beginning 000^p1^q. We first show that, if $\alpha \neq \beta$, then we are done because $R_{\neq} \leqslant_{\mathrm{ppp}} R$, which means that R 3-simulates equality by Lemma 9

To this end, suppose $\alpha > \beta$ so, in particular, $\alpha \geqslant 2$ and there are distinct tuples $010^p1^q\bar{a}$ and $010^p1^q\bar{b}$ in R. We may assume, without loss of generality, that $a_1 \neq b_1$. Since $\beta \geqslant 1$, there is at least one tuple $100^p1^q\bar{c} \in R$. Suppose, now that we pin the $(2+p+q+1)$th column of R to c_1. R cannot contain any tuple $110^p1^q\bar{d}$ with $d_1 = c_1$ because it is not possible to pin more columns and still ppp-define R_{OR}. But then R contains tuples beginning with each of $010^p1^qc_1$ and $100^p1^qc_1$ and none beginning $000^p1^qc_1$ or $110^p1^qc_1$, so $R_{\neq} \leqslant_{\mathrm{ppp}} R$. We similarly have $R_{\neq} \leqslant_{\mathrm{ppp}} R$ if $\alpha < \beta$. From this point, we may assume that $\alpha = \beta$.

Similarly, either $R_{\neq} \leqslant_{\mathrm{ppp}} R'$, so we are done, or R' contains α' tuples beginning with each of $010^{p'}1^{q'}$ and $100^{p'}1^{q'}$, γ' tuples beginning $000^{p'}1^{q'}$ and no tuples beginning $110^{p'}1^{q'}$.

We now show how to simulate equality. We can 3-simulate $R_{=,k}$ by replacing the constraint $R_{=,k}(x_1,\dots,x_k)$ with the following set of constraints, modelled on ξ_k:

$$\Xi_k = \{R(x_iy_i0^p1^q*) \mid 1 \leqslant i \leqslant k\} \cup \{R'(y_ix_{i+1}0^{p'}1^{q'}*) \mid 1 \leqslant i < k\} \cup \{R'(y_kx_10^{p'}1^{q'}*)\},$$

where the y_i are fresh variables and, as before, $*$ denotes a fresh tuple of variables for each constraint, of the appropriate length. By Lemma 12, an assignment σ satisfies Ξ_k if, and only if, $\sigma(x_1) = \dots = \sigma(x_k) \neq \sigma(y_1) = \dots = \sigma(y_k)$.

Further, there are α ways to satisfy the variables denoted by $*$ in each R constraint and α' ways in each R' constraint. Therefore, there are $(\alpha\alpha')^k$ satisfying assignments for Ξ_k corresponding to each satisfying assignment for $R_{=,k}$ and we are done.

Notice that our assumption that the ppp definitions of R_{OR} in R and R_{NAND} in R' involve the identity permutation, pinning sequential columns to zero and one and projecting to the first two columns was made only for the notational convenience of referring to "tuples beginning 010^p1^q" and so on. This being the case, there is no requirement that R and R' be distinct, so the proof is complete. □

Note that there are relations, such as $R_{=,3}$, that 2-simulate equality, though we do not require this, here, so we omit the proof.

5. Classifying relations

We are now ready to prove that every Boolean relation R is in OR-conj, in NAND-conj or 3-simulates equality. Given r-ary relations R_0 and R_1, we write $R_0 + R_1$ for the relation $\{0\bar{a} \mid \bar{a} \in R_0\} \cup \{1\bar{a} \mid \bar{a} \in R_1\}$. The proof of the classification is by induction on the arity of R and proceeds by decomposing R as $R_0 + R_1$.

Recall that a width-zero OR-conj (or, equivalently, NAND-conj) relation is a complete relation, possibly padded with some constant columns.

Lemma 14. *Let $R_0, R_1 \subseteq$ OR-conj have arity r and width zero and let $R = R_0 + R_1$. Then, $R \in$ OR-conj, $R \in$ NAND-conj or R 3-simulates equality.*

Proof. We may assume that R has no constant columns, since adding or removing such columns does not affect whether a relation is OR-conj or NAND-conj or whether it 3-simulates equality.

For $i \in [2, r+1]$, let $R'_i = \mathrm{proj}_{1,i} R$, so each $R'_i \leqslant_{\mathrm{ppp}} R$. If any R'_i is $R_=$, $R_{\neq}$, $R_{\to}$ or $R_{\leftarrow}$ then R 3-simulates equality by Lemma 9. Otherwise, each R'_i is either $\{0,1\}^2$, R_{OR} or R_{NAND}. If $R'_j = R_{\mathrm{OR}}$ and $R'_k = R_{\mathrm{NAND}}$ for some j and k, then R 3-simulates equality by Lemma 13. Otherwise, if no $R'_i = R_{\mathrm{NAND}}$, let $I = \{i \mid R'_i = R_{\mathrm{OR}}\}$. Then,

$$R = \bigwedge_{i \in I} \mathrm{OR}(x_1, x_i),$$

so $R \in$ OR-conj. If no $R'_i = R_{\mathrm{OR}}$, then $R \in$ NAND-conj, by a similar argument. □

Lemma 15. *Let $R_0, R_1 \subseteq \{0,1\}^r$ be OR-conj and let $R = R_0 + R_1$. Then, $R \in$ OR-conj, $R \in$ NAND-conj or R 3-simulates equality.*

Proof. We may assume, as before, that R has no constant columns. We may also assume that at least one of R_0 and R_1 has positive width: otherwise, the result is immediate from the previous lemma. We split the remaining work into two cases.

Case 1: $R_0 \subseteq R_1$. Note that R_1 cannot have any constant columns in this case, since the same column would also have to be constant in R_0, giving a constant column in R.

Suppose R_i is defined by the normalized OR-conj formula ϕ_i in variables $x_2, \ldots, x_{r+1}$. Then R is defined by the formula

$$\begin{aligned}\phi_0 \vee (x_1 = 1 \wedge \phi_1) &\equiv (\phi_0 \vee x_1 = 1) \wedge (\phi_0 \vee \phi_1) \\ &\equiv (\phi_0 \vee x_1 = 1) \wedge \phi_1, \end{aligned} \tag{1}$$

where the first equivalence is the distribution law and the second is because ϕ_0 implies ϕ_1 (because $R_0 \subseteq R_1$). We consider the following two cases.

Case 1.1: *R_0 has no constant columns.* ϕ_0 contains no pins and $x_1 = 1$ is equivalent to $\mathrm{OR}(x_1)$ so we can rewrite $\phi_0 \vee x_1 = 1$ in CNF. Therefore, the formula (1) defines an OR-conj relation.

Case 1.2: *R_0 has a constant column.* R_1 has no constant columns so, if $\mathrm{proj}_k R_0 = \{0\}$ for some k, then $\mathrm{proj}_{1,k+1} R = R_{\leftarrow}$, and R 3-simulates equality by Lemma 9. If every constant column of R_0 is all ones, then ϕ_0 is in CNF since every pinning $x_i = 1$ in ϕ_0 can be written $\mathrm{OR}(x_i)$. We can therefore rewrite $\phi_0 \vee x_1 = 1$ in CNF, as in Case 1.1.

Case 2: $R_0 \nsubseteq R_1$. We will show that R 3-simulates equality or is in NAND-conj. We consider two cases.

Case 2.1: *R_1 has a constant column*, say the kth. If the kth column of R_0 is also constant, it must have the opposite value (or R would have a constant column). Therefore, $\mathrm{proj}_{1,k+1} R$ is either $R_=$ or $R_{\neq}$, so R 3-simulates equality by Lemma 9.

Otherwise, the kth column of R_0 is not constant, so $\mathrm{proj}_{1,k+1} R$ is either $R_{\rightarrow}$ or R_{NAND}. In the first case, R 3-simulates equality by Lemma 9. In the second case, R_{OR} is ppp-definable in at least one of R_0 and R_1 by Lemma 5 so R 3-simulates equality by Lemma 13.

Case 2.2: *R_1 has no constant columns.* By Proposition 7, R_1 is monotone. Let $\bar{a} \in R_0 \setminus R_1$: by applying the same permutation to the columns of R_0 and R_1, we may assume that $\bar{a} = 0^\ell 1^{r-\ell}$. We must have $\ell \geqslant 1$ as every non-empty r-ary monotone relation contains the tuple 1^r. Let $\bar{b} \in R_1$ be a tuple such that $a_i = b_i$ for all i in a maximal initial segment of $[1, r]$. By monotonicity of R_1, we may assume that $\bar{b} = 0^k 1^{r-k}$. Further, we must have $k < \ell$, since, otherwise, we would have $\bar{b} < \bar{a}$, and this contradicts our choice of $\bar{a} \notin R_1$.

Now, consider the relation

$$R' = \left\{a_0 a_1 \ldots a_{\ell-k} \;\middle|\; a_0 0^k a_1 \ldots a_{\ell-k} 1^{r-\ell} \in R\right\},$$

which is the result of pinning columns 2 to $(k+1)$ of R to zero and columns $(r-\ell+1)$ to $(r+1)$ to one and discarding the resulting constant columns. R' contains $0^{\ell-k+1}$ and $1^{\ell-k+1}$ but is not complete, since it does not contain $10^{\ell-k}$. By Lemma 11, R' 3-simulates equality, so R does, too. $\square$

The following corollary follows from Proposition 6 and the facts that $\widetilde{R_0 + R_1} = \widetilde{R_1} + \widetilde{R_0}$ and that, if $\widetilde{R}$ 3-defines equality, then so does R (since the equality relation is its own bit-wise complement).

Corollary 16. *Let $R_0, R_1 \in$ NAND-conj and let $R = R_0 + R_1$. Then $R \in$ OR-conj, $R \in$ NAND-conj or R 3-simulates equality.*

Theorem 17. *Every Boolean relation is in OR-conj, is in NAND-conj or 3-simulates equality.*

Proof. Let R be a Boolean relation. We proceed by induction on its arity, r. If $r \leqslant 2$, then, if R is neither OR-conj nor NAND-conj then it can only be $R_=$, $R_{\neq}$, $R_{\rightarrow}$ or $R_{\leftarrow}$; all of these 3-simulate equality by Lemma 9.

For the inductive step, let R have arity $r+1 > 2$ and let R_0 and R_1 be such that $R = R_0 + R_1$. By the inductive hypothesis, each of R_0 and R_1 is in OR-conj, in NAND-conj or 3-simulates equality.

If either of R_0 and R_1 3-simulates equality, then so does R. Otherwise, either both are in OR-conj, both are in NAND-conj or one is in OR-conj and the other in NAND-conj. In the first two cases, R is in OR-conj or in NAND-conj or 3-simulates equality by Lemma 15 or Corollary 16. In the third case, if R_0 and R_1 have positive width, then R 3-simulates equality by Lemma 13; otherwise, we are in one of the first two cases. $\square$

6. Complexity

The complexity of approximating #CSP(Γ) where the degree of instances is unbounded is given by Dyer, Goldberg and Jerrum [20, Theorem 3].

Theorem 18. *Let Γ be a Boolean constraint language.*

- *If every $R \in \Gamma$ is affine, then #CSP(Γ) $\in$* **FP**.

- *Otherwise, if $\Gamma \subseteq$ IM-conj, then* $\#\mathrm{CSP}(\Gamma) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$.
- *Otherwise,* $\#\mathrm{CSP}(\Gamma) \equiv_{\mathrm{AP}} \#\mathrm{SAT}$.

Working towards our classification of the approximation complexity of bounded-degree $\#\mathrm{CSP}(\Gamma)$, we first deal with sub-cases. Recall that #BIS is the problem of counting independent sets in bipartite graphs and $\#w\text{-HIS}_d$ is that of counting independent sets in hypergraphs where every vertex has degree at most d and every hyper-edge contains at most w vertices.

Proposition 19. *If $\Gamma \subseteq$ IM-conj contains a non-affine relation, then, for all $d \geqslant 3$, $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$.*

Proof. It is immediate from [20, Lemma 9] that $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \leqslant_{\mathrm{AP}} \#\mathrm{BIS}$.

For the converse, first observe that, by [20, Lemma 8], $\#\mathrm{BIS} \leqslant_{\mathrm{AP}} \#\mathrm{CSP}(\{R_{\to}\})$ and, since $R_{\to}$ 3-simulates equality by Lemma 9, we have $\#\mathrm{CSP}(\{R_{\to}\}) \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\{R_{\to}\} \cup \Gamma_{\mathrm{pin}})$ for all $d \geqslant 3$ by Proposition 8. We must show that $\#\mathrm{CSP}_d(\{R_{\to}\} \cup \Gamma_{\mathrm{pin}}) \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$.

To this end, let R be any non-affine relation in Γ. By Lemma 4, $R_{\to} \leqslant_{\mathrm{ppp}} R$ and the ppp definition involves projecting only pinned columns. Therefore, we can express the constraint $R_{\to}(x, y)$ by a constraint of the form $R(v_1, \dots, v_r)$, where, for some i and j, $v_i = x$ and $v_j = y$ and the other variables are pinned to zero or one. $\square$

Lemma 20. *For $d \geqslant 2$ and $w \geqslant 2$,*

$$\#w\text{-HIS}_d \equiv_{\mathrm{AP}} \#\mathrm{CSP}_d\big(\{R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}}\big) \equiv_{\mathrm{AP}} \#\mathrm{CSP}_d\big(\{R_{\mathrm{NAND},w}\} \cup \Gamma_{\mathrm{pin}}\big).$$

Proof. The second equivalence is trivial, since $R_{\mathrm{OR},w}$ and $R_{\mathrm{NAND},w}$ are bit-wise complements of each other.

For the first equivalence, let H be an instance of $\#w\text{-HIS}_d$. We create an instance of $\#\mathrm{CSP}_d(\{R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}})$ as follows. The variables are $\{x_v \mid v \in V(H)\}$ and, for each hyper-edge $\{v_1, \dots, v_s\}$, there is a constraint $R_{\mathrm{OR},w}(x_{v_1}, \dots, x_{v_s}, 0, \dots, 0)$. Each vertex appears in at most d hyper-edges so each variable appears in at most d constraints. It is easy to see that a configuration σ of the resulting $\#\mathrm{CSP}_d(\{R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}})$ instance is satisfying if, and only if, $\{v \mid \sigma(x_v) = 0\}$ is an independent set in H.

Conversely, if we are given an instance of $\#\mathrm{CSP}_d(\{R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}})$, we create an instance H of $\#w\text{-HIS}_d$ as follows. There is a vertex v_x for every variable x. For every constraint $R_{\mathrm{OR},w}(x_1, \dots, x_w)$ (where the x_i are not necessarily distinct), add the hyper-edge $\{v_{x_1}, \dots, v_{x_w}\}$. Now, for every constraint $R_{\mathrm{zero}}(x)$, delete the vertex v_x and remove it from every hyper-edge that contains it. For every constraint $R_{\mathrm{one}}(x)$, delete v_x and delete every hyper-edge that contains it. It is easy to see that a configuration σ is satisfying if, and only if, it satisfies the pins and the set $\{v_x \mid \sigma(x) = 0\} \cap V(H)$ is independent in H. $\square$

In the following two propositions, we just prove the OR-conj cases; the NAND-conj cases are equivalent.

Proposition 21. *Let R be an* OR-*conj or* NAND-*conj relation of width $w > 0$. Then, for $d \geqslant 2$, $\#w\text{-HIS}_d \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\{R\} \cup \Gamma_{\mathrm{pin}})$.*

Proof. By Lemma 5, $R_{\mathrm{OR},w} \leqslant_{\mathrm{ppp}} R$ and the ppp definition involves pinning and then projecting away all but w of the columns. Thus, an $R_{\mathrm{OR},w}$-constraint can be simulated by an R-constraint in which some elements of the scope are constants. The result follows from Lemma 20. $\square$

We define the *variable rank* of an OR-conj or NAND-conj relation R to be $\mathrm{vrank}(R)$, the greatest number of times that any variable appears in the (unique) normalized formula that defines R. We similarly define the variable rank of an OR-conj or NAND-conj constraint language to be the maximum variable rank of the relations within it.

Proposition 22. *Let R be an* OR-*conj or* NAND-*conj relation of width $w > 0$ and variable rank k. Then, for $d \geqslant 2$, $\#\mathrm{CSP}_d(\{R\} \cup \Gamma_{\mathrm{pin}}) \leqslant_{\mathrm{AP}} \#w\text{-HIS}_{kd}$.*

Proof. Given an instance I of $\#\mathrm{CSP}_d(\{R\} \cup \Gamma_{\mathrm{pin}})$, we produce an instance I' of the problem $\#\mathrm{CSP}(\{R_{\mathrm{OR},2}, \dots, R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}})$ with the same variables by replacing every R-constraint with the $R_{\mathrm{OR},i}$-constraints and pins corresponding to the normalized formula that defines R. Clearly, $Z(I) = Z(I')$ but a variable that appeared d times in I appears up to kd times in I', so we have established that

$$\begin{aligned} \#\mathrm{CSP}_d\big(\{R\} \cup \Gamma_{\mathrm{pin}}\big) &\leqslant_{\mathrm{AP}} \#\mathrm{CSP}_{kd}\big(\{R_{\mathrm{OR},2}, \dots, R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}}\big) \\ &\leqslant_{\mathrm{AP}} \#\mathrm{CSP}_{kd}\big(\{R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}}\big), \end{aligned}$$

where the last reduction holds because the constraint $R_{\mathrm{OR},s}(x_1, \dots, x_s)$ is equivalent to $R_{\mathrm{OR},w}(x_1, \dots, x_s, 0, \dots, 0)$ for any $s < w$. By Lemma 20, $\#\mathrm{CSP}_{kd}(\{R_{\mathrm{OR},w}\} \cup \Gamma_{\mathrm{pin}})$ is AP-equivalent to $\#w\text{-HIS}_{kd}$. $\square$

Table 1
A summary of known approximability of $\#w\text{-HIS}_d$. For values of d and w not covered by the table, the approximability is still unknown.

Degree d	Width w	Approximability of $\#w\text{-HIS}_d$
1	$\geqslant 2$	Exact counting in **FP**
2	2	Exact counting in **FP**
2	$\geqslant 3$	FPRAS [22]
3	2, 3	FPRAS [22]
3, 4, 5	2	PTAS [43]
$\geqslant 6$	$\geqslant 2$	No FPRAS unless $\mathbf{NP} = \mathbf{RP}$ [39]

We now give the complexity of approximating $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$ for $d \geqslant 3$.

Theorem 23. *Let Γ be a Boolean constraint language and let $d \geqslant 3$.*

- *If every $R \in \Gamma$ is affine, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \in \mathbf{FP}$.*
- *Otherwise, if $\Gamma \subseteq$ IM-conj, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$.*
- *Otherwise, if $\Gamma \subseteq \mathrm{OR\text{-}conj}$ or $\Gamma \subseteq \mathrm{NAND\text{-}conj}$, then $\#w\text{-HIS}_d \leqslant_{\mathrm{AP}} \#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \leqslant_{\mathrm{AP}} \#w\text{-HIS}_{kd}$, where $w = \mathrm{width}(\Gamma)$ and $k = \mathrm{vrank}(\Gamma)$.*
- *Otherwise, $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{SAT}$.*

Proof. The first three cases are immediate from Theorem 18 and Propositions 19, 21 and 22. Note that $\Gamma \cup \Gamma_{\mathrm{pin}}$ is affine if, and only if, Γ is.

For the remaining case, suppose that Γ is not affine, $\Gamma \nsubseteq$ IM-conj, $\Gamma \nsubseteq$ OR-conj and $\Gamma \nsubseteq$ NAND-conj. Since $\Gamma \cup \Gamma_{\mathrm{pin}}$ is neither affine nor a subset of IM-conj, we have $\#\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{SAT}$ by Theorem 18 so, if we can show that Γ d-simulates equality, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ by Proposition 8 and we are done. If Γ contains a relation R that is neither OR-conj nor NAND-conj, then R 3-simulates equality by Theorem 17. Otherwise, Γ must contain distinct relations $R_1 \in$ OR-conj and $R_2 \in$ NAND-conj that are non-affine so have width at least two, so Γ 3-simulates equality by Lemma 13. □

Sly has shown that there can be no FPRAS for the problem of counting independent sets in graphs of maximum degree at least 6, unless $\mathbf{NP} = \mathbf{RP}$ [39]. Clearly, if there is no FPRAS for counting independent sets in such graphs, there can be no FPRAS for $\#w\text{-HIS}_d$ with $w \geqslant 2$ and $d \geqslant 6$. Further, since #SAT is complete for #**P** with respect to AP-reducibility [18], #SAT cannot have an FPRAS unless $\mathbf{NP} = \mathbf{RP}$. Thus, Theorem 24 below is an immediate corollary of Theorem 23.

Theorem 24. *Let Γ be a Boolean constraint language and let $d \geqslant 6$.*

- *If every $R \in \Gamma$ is affine, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \in \mathbf{FP}$.*
- *Otherwise, if $\Gamma \subseteq$ IM-conj, then $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}}) \equiv_{\mathrm{AP}} \#\mathrm{BIS}$.*
- *Otherwise, there is no FPRAS for $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$, unless $\mathbf{NP} = \mathbf{RP}$.*

Note that $\Gamma \cup \Gamma_{\mathrm{pin}}$ is affine (respectively, in OR-conj or in NAND-conj) if, and only if, Γ is. Therefore, the case for large-degree instances ($d \geqslant 6$) corresponds exactly in complexity to the unbounded case [20].

For lower degree bounds, the picture is more complex. To put Theorem 23 in context, we give a summary of what is known about the approximability of $\#w\text{-HIS}_d$ for various values of d and w.

The case $d = 1$ is clearly in **FP** (Theorem 1) and so is the case $d = w = 2$, which corresponds to counting independent sets in graphs of maximum degree two. For $d = 2$ and width $w \geqslant 3$, Dyer and Greenhill have shown that there is an FPRAS for $\#w\text{-HIS}_d$ [22]. For $d = 3$, they have shown that there is an FPRAS if the width w is at most 3. For larger width, the approximability of $\#w\text{-HIS}_3$ is still not known. With the width restricted to $w = 2$ (ordinary graphs), Weitz has shown that, for degree $d \in \{3, 4, 5\}$, there is a deterministic approximation scheme that runs in polynomial time (a PTAS) [43]. This extends a result of Luby and Vigoda, who gave an FPRAS for $d \leqslant 4$ [35]. For $d > 5$, approximating $\#w\text{-HIS}_d$ becomes considerably harder. Dyer, Frieze and Jerrum showed that, for $d = 6$, the Monte Carlo Markov chain technique is likely to fail, in the sense that a certain class of Markov chains are provably slowly mixing [17]. They also showed that, for $d = 25$, there can be no polynomial-time algorithm for approximate counting, unless $\mathbf{NP} = \mathbf{RP}$. As mentioned above, Sly has recently improved on this, showing that there can be no FPRAS for $d \geqslant 6$ unless $\mathbf{NP} = \mathbf{RP}$. Table 1 summarizes the results.

Returning to bounded-degree #CSP, the case $d = 2$ seems to have a rather different flavour to degree bounds three and higher. This is also the case for decision CSP—recall that the complexity of degree-d $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ is the same as unbounded-degree $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ for all $d \geqslant 3$ [14], while degree-2 $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ is often easier than the unbounded-degree case [14,25] but there are still constraint languages Γ for which the complexity of degree-2 $\mathrm{CSP}(\Gamma \cup \Gamma_{\mathrm{pin}})$ is open.

Our key techniques for determining the complexity of $\#\mathrm{CSP}_d(\Gamma \cup \Gamma_{\mathrm{pin}})$ for $d \geqslant 3$ are the 3-simulation of equality and Theorem 17, which says that every Boolean relation is in OR-conj, in NAND-conj or 3-simulates equality. However, it seems that not all relations that 3-simulate equality also 2-simulate equality so the corresponding classification of relations does not appear to hold. It seems that different techniques will be required for the degree-2 case. For example, it is possible that there is no FPRAS for #BIS and, therefore, no FPRAS for $\#\mathrm{CSP}_3(\Gamma \cup \Gamma_{\mathrm{pin}})$ except when Γ is affine. However, Bubley and Dyer have shown that there is an FPRAS for the restriction of #SAT in which each variable appears at most twice, even though the exact counting problem is #**P**-complete [2]; the corresponding constraint language is not affine. This also shows that there is a class $\mathcal{C}$ of constraint languages for which $\#\mathrm{CSP}_2(\Gamma \cup \Gamma_{\mathrm{pin}})$ has an FPRAS for every $\Gamma \in \mathcal{C}$ but for which no exact polynomial-time algorithm exists, unless $\mathbf{FP} = \#\mathbf{P}$.

We leave the complexity of degree-2 #CSP and of #BIS and the various parameterized versions of the counting hypergraph independent sets problem as open questions.

References

[1] M. Bordewich, On the approximation complexity hierarchy, in: 8th Int. Workshop on Approximation and Online Algorithms (WOAO 2010), in: Lecture Notes in Comput. Sci., vol. 6534, 2011, pp. 37–46.

[2] R. Bubley, M.E. Dyer, Graph orientations with no sink and an approximation for a hard case of #SAT, in: 8th ACM–SIAM Symp. on Discrete Algorithms (SODA 1997), 1997, pp. 248–257.

[3] A.A. Bulatov, A dichotomy theorem for constraint satisfaction problems on a 3-element domain, J. ACM 53 (1) (2006) 66–120.

[4] A.A. Bulatov, The complexity of the counting constraint satisfaction problem, in: 35th Int. Colloq. on Automata, Languages and Programming (ICALP 2008), Part I, in: Lecture Notes in Comput. Sci., vol. 5125, Springer, 2008, pp. 646–661.

[5] J.-Y. Cai, X. Chen, A decidable dichotomy theorem on directed graph homomorphisms with non-negative weights, in: 51st IEEE Symp. on Foundations of Computer Science (FOCS 2010), IEEE Computer Society, 2010, pp. 437–446.

[6] J.-Y. Cai, P. Lu, M. Xia, Holant problems and counting CSP, in: 41st Annual ACM Symp. on Theory of Computing (STOC 2009), 2009, pp. 715–724.

[7] J.-Y. Cai, P. Lu, M. Xia, The complexity of complex weighted Boolean #CSP, preprint, 2009.

[8] D.A. Cohen, P.G. Jeavons, The complexity of constraint languages, in: Handbook of Constraint Programming, Elsevier, 2006 (Ch. 8).

[9] D.A. Cohen, P.G. Jeavons, M. Gyssens, A unified theory of structural tractability for constraint satisfaction problems, J. Comput. System Sci. 74 (5) (2008) 721–743.

[10] B. Courcelle, Graph rewriting: An algebraic and logic approach, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, 1990, pp. 194–242.

[11] B. Courcelle, J.A. Makowski, U. Rotics, On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic, Discrete Appl. Math. 108 (1–2) (2001) 23–52.

[12] N. Creignou, M. Hermann, Complexity of generalized satisfiability counting problems, Inform. and Comput. 125 (1) (1996) 1–12.

[13] N. Creignou, S. Khanna, M. Sudan, Complexity Classification of Boolean Constraint Satisfaction Problems, SIAM, 2001.

[14] V. Dalmau, D.K. Ford, Generalized satisfiability with limited occurrences per variable: A study through Delta-matroid parity, in: Mathematical Foundations of Computer Science (MFCS 2003), in: Lecture Notes in Comput. Sci., vol. 2747, Springer, 2003, pp. 358–367.

[15] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence 38 (3) (1989) 353–366.

[16] J. Díaz, M. Serna, D.M. Thilikos, Counting h-colorings of partial k-trees, Theoret. Comput. Sci. 281 (1–2) (2002) 291–309.

[17] M.E. Dyer, A. Frieze, M.R. Jerrum, On counting independent sets in sparse graphs, SIAM J. Comput. 31 (5) (2002) 1527–1541.

[18] M.E. Dyer, L.A. Goldberg, C.S. Greenhill, M.R. Jerrum, The relative complexity of approximate counting problems, Algorithmica 38 (3) (2003) 471–500.

[19] M.E. Dyer, L.A. Goldberg, M.R. Jerrum, The complexity of weighted Boolean #CSP, SIAM J. Comput. 38 (5) (2009) 1970–1986.

[20] M.E. Dyer, L.A. Goldberg, M.R. Jerrum, An approximation trichotomy for Boolean #CSP, J. Comput. System Sci. 76 (3–4) (2010) 267–277.

[21] M.E. Dyer, L.A. Goldberg, M. Paterson, On counting homomorphisms to directed acyclic graphs, J. ACM 54 (6) (2007), Article No. 27.

[22] M.E. Dyer, C.S. Greenhill, On Markov chains for independent sets, J. Algorithms 35 (1) (2000) 17–49.

[23] M.E. Dyer, C.S. Greenhill, The complexity of counting graph homomorphisms, Random Structures Algorithms 17 (3–4) (2000) 260–289; Random Structures Algorithms 25 (3) (2004) 346–352 (Corrigendum).

[24] M.E. Dyer, D.M. Richerby, An effective dichotomy for the counting constraint satisfaction problem, SIAM J. Comput., in press.

[25] T. Feder, Fanout limitations on constraint systems, Theoret. Comput. Sci. 255 (1–2) (2001) 281–293.

[26] T. Feder, M.Y. Vardi, The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory, SIAM J. Comput. 28 (1) (1998) 57–104.

[27] E.C. Freuder, Complexity of k-tree structured constraint satisfaction problems, in: 8th Conf. of American Association for Artificial Intelligence, AAAI Press/MIT Press, 1990, pp. 4–9.

[28] P. Hell, J. Nešetřil, On the complexity of h-coloring, J. Combin. Theory Ser. B 48 (1) (1990) 92–110.

[29] M.R. Jerrum, L.G. Valiant, V.V. Vazirani, Random generation of combinatorial structures from a uniform distribution, Theoret. Comput. Sci. 43 (1986) 169–188.

[30] D.E. Knuth, The Art of Computer Programming, vol. 4A: Combinatorial Algorithms, Addison–Wesley, Reading, MA, 2011.

[31] P.G. Kolaitis, M.Y. Vardi, A game-theoretic approach to constraint satisfaction, in: 17th Conf. of American Association for Artificial Intelligence, AAAI Press/MIT Press, 2000, pp. 175–181.

[32] P.G. Kolaitis, M.Y. Vardi, Conjunctive query containment and constraint satisfaction, J. Comput. System Sci. 61 (2) (2000) 302–332.

[33] V. Kumar, Algorithms for constraint satisfaction problems: A survey, AI Mag. 13 (1) (1992) 33–42.

[34] R.E. Ladner, On the structure of polynomial time reducibility, J. ACM 22 (1) (1975) 155–171.

[35] M. Luby, E. Vigoda, Fast convergence of the Glauber dynamics for sampling independent sets, Random Structures Algorithms 15 (3–4) (1999) 229–241.

[36] U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, Inform. Sci. 7 (1974) 95–135.

[37] T.J. Schaefer, The complexity of satisfiability problems, in: 10th ACM Symp. on Theory of Computing, 1978, pp. 216–226.

[38] J. Simon, On the difference between one and many, in: 4th Int. Colloq. on Automata, Languages and Programming (ICALP 1977), in: Lecture Notes in Comput. Sci., vol. 52, Springer, 1977, pp. 480–491 (preliminary version).

[39] A. Sly, Computational transition at the uniqueness threshold, in: 51st IEEE Symp. on Foundations of Computer Science (FOCS 2010), IEEE Computer Society, 2010, pp. 287–296.

[40] S. Toda, On the computational power of PP and $\bigoplus$P, in: 30th Annual Symp. on Foundations of Computer Science (FOCS 1989), IEEE Computer Society, 1989, pp. 514–519.

[41] L.G. Valiant, The complexity of computing the permanent, Theoret. Comput. Sci. 8 (1979) 189–201.

[42] L.G. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (3) (1979) 410–421.

[43] D. Weitz, Counting independent sets up to the tree threshold, in: 38th ACM Symp. on Theory of Computing (STOC 2006), 2006, pp. 140–149.

[44] D. Welsh, Complexity: Knots, Colourings and Counting, Cambridge University Press, 1993.