Beyond PCSP(**1-in-3**, **NAE**)*

Alex Brandts
University of Oxford
alex.brandts@cs.ox.ac.uk

Stanislav Živný
University of Oxford
standa.zivny@cs.ox.ac.uk

August 28, 2022

Abstract

The promise constraint satisfaction problem (PCSP) is a recently introduced vast generalisation of the constraint satisfaction problem (CSP) that captures approximability of satisfiable instances. A PCSP instance comes with two forms of each constraint: a strict one and a weak one. Given the promise that a solution exists using the strict constraints, the task is to find a solution using the weak constraints. While there are by now several dichotomy results for fragments of PCSPs, they all consider (in some way) symmetric PCSPs.

1-in-3-SAT and Not-All-Equal-3-SAT are classic examples of Boolean symmetric (non-promise) CSPs. While both problems are NP-hard, Brakensiek and Guruswami showed [SICOMP'21] that given a satisfiable instance of 1-in-3-SAT one can find a solution to the corresponding instance of (weaker) Not-All-Equal-3-SAT. In other words, the PCSP template (**1-in-3**, **NAE**) is tractable.

We focus on *non-symmetric* PCSPs. In particular, we study PCSP templates obtained from the Boolean template (**t-in-k**, **NAE**) by either adding tuples to **t-in-k** or removing tuples from **NAE**. For the former, we classify all templates as either tractable or not solvable by one of the strongest known algorithm for PCSPs, the combined basic LP and affine IP relaxation of Brakensiek, Guruswami, Wrochna, and Živný [SICOMP'20]. For the latter, we classify all templates as either tractable or NP-hard.

1 Introduction

How hard is it to find a 6-colouring of a graph if it is promised to be 3-colourable? We do not know but believe it to be NP-hard. Despite sustained effort, this so-called *approximate graph colouring* problem has been elusive since it was considered by Garey and Johnson almost 50 years ago [21]. The current state of the art is NP-hardness of finding a 5-colouring of a 3-colourable graph [6]. Approximate graph colouring is an example of the very general promise constraint satisfaction problem, which is the focus of this paper. We start with (non-promise) constraint satisfaction problems to set the stage.

*An extended abstract of this work appeared in the *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP'21)* [13]. Alex Brandts was supported by a Royal Society Enhancement Award and an NSERC PGS Doctoral Award. Stanislav Živný was supported by a Royal Society University Research Fellowship. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

Constraint satisfaction While deciding whether a graph is 2-colourable is solvable in polynomial time, deciding 3-colourability is NP-complete [25]. The *constraint satisfaction problem* (CSP) is a general framework that captures graph colourings and many other fundamental computational problems. Feder and Vardi initiated a systematic study of so-called fixed-template decision CSPs. Let $\mathbf{A}$ be a fixed finite relational structure, called the *template* or constraint language; i.e., $\mathbf{A}$ consists of a finite universe A and finitely many relations on A, each of possibly different arity. The fixed-template CSP over $\mathbf{A}$, denoted by $\mathrm{CSP}(\mathbf{A})$, is the class of CSPs in which all constraint relations come from $\mathbf{A}$. In more detail, $\mathrm{CSP}(\mathbf{A})$ denotes the following computational problem: Given a structure $\mathbf{X}$ over the same signature as $\mathbf{A}$, is there a homomorphism from $\mathbf{X}$ to $\mathbf{A}$, denoted by $\mathbf{X} \to \mathbf{A}$? (Formal definitions can be found in Section 2.) If $\mathbf{A} = K_3$ is a clique on 3 vertices then $\mathrm{CSP}(\mathbf{A})$ is precisely the standard graph 3-colouring problem.

A classic result of Schaefer shows that, for any $\mathbf{A}$ on a 2-element set, $\mathrm{CSP}(\mathbf{A})$ is either solvable in polynomial time or NP-complete. The non-trivial tractable cases from Schaefer's classification are taught in undergraduate algorithms courses: 2-SAT, (dual) Horn-SAT, and linear equations over $\{0, 1\}$. Two concrete CSPs that are NP-hard by Schaefer's result are the (positive) 1-in-3-SAT and (positive) Not-All-Equal-3-SAT. For both problems, the instance is a list of triples of variables. In 1-in-3-SAT, the task is to find a mapping from the variables to $\{0, 1\}$ so that in each specified triple exactly one variable is set to 1. Formally, 1-in-3-SAT is $\mathrm{CSP}(\textbf{1-in-3})$, where $\textbf{1-in-3} = (\{0, 1\}; \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\})$. In Not-All-Equal-3-SAT, the task is to find a mapping from the variables to $\{0, 1\}$ so that in each triple not all variables are assigned the same value. Formally, Not-All-Equal-3-SAT is $\mathrm{CSP}(\mathbf{NAE})$, where $\mathbf{NAE} = (\{0, 1\}; \{0, 1\}^3 \setminus \{(0, 0, 0), (1, 1, 1)\})$.

If $\mathbf{A}$ is a graph (i.e., a single symmetric binary relation) then, as shown by Hell and Nešetřil [23], $\mathrm{CSP}(\mathbf{A})$ is either solvable in polynomial time or NP-complete.

Based on these two examples and a connection to logic, Feder and Vardi famously conjectured [19] that, for any finite $\mathbf{A}$, $\mathrm{CSP}(\mathbf{A})$ is either solvable in polynomial time or NP-complete. Bulatov [15], and independently Zhuk [30], proved the conjecture in the affirmative, both relying on the algebraic approach to CSPs [24, 14, 7].

Promise constraint satisfaction Austrin, Guruswami, and Håstad [4] and Brakensiek and Guruswami [10] initiated the investigation of the *promise constraint satisfaction problem* (PCSP), which is a vast generalisation of the CSP. Let $\mathbf{A}$ and $\mathbf{B}$ be two relational structures such that $\mathbf{A} \to \mathbf{B}$. The fixed-template PCSP over $\mathbf{A}$ and $\mathbf{B}$, denoted by $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$, is the following computational problem: Given $\mathbf{X}$ such that $\mathbf{X} \to \mathbf{A}$, find a homomorphism from $\mathbf{X}$ to $\mathbf{B}$ (which exists by the composition of the promised homomorphism from $\mathbf{X}$ to $\mathbf{A}$ and the homomorphism from $\mathbf{A}$ to $\mathbf{B}$). If we take $\mathbf{A} = K_3$ to be a clique on 3 vertices and $\mathbf{B} = K_6$ to be a clique on 6 vertices, then $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is an instance of the approximate graph colouring problem mentioned at the beginning of this article.

Actually, what we described is the *search* version of the PCSP. The *decision* version is as follows: Given $\mathbf{X}$, return Yes if $\mathbf{X} \to \mathbf{A}$ and return No if $\mathbf{X} \not\to \mathbf{B}$. (The promise in the decision version is that it does not happen that $\mathbf{X} \not\to \mathbf{A}$ but $\mathbf{X} \to \mathbf{B}$.) It is well known that the decision version reduces to the search version but it is not known whether there is a reduction the other way [6]. In most results (including ours), hardness is established for the decision version and tractability for the search version.

If $\mathbf{A} = \mathbf{B}$ then $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is the same as $\mathrm{CSP}(\mathbf{A})$ and thus PCSPs indeed generalise

CSPs. For CSPs, the decision and search versions are known to be equivalent [14].

Building on the result of Barto, Opršal, and Pinsker [8] that the complexity of CSP($\mathbf{A}$) is captured by certain types of identities of higher-order symmetries (called polymorphisms) of $\mathbf{A}$, Barto, Bulín, Krokhin, and Opršal showed that the basics of the algebraic approach developed for CSPs [8] can be generalised to PCSPs [6], thus introducing a general methodology for investigating the computational complexity of PCSPs. In particular, among other things, they showed that finding a 5-colouring of a 3-colourable graph is NP-hard.

Related work Motivated by the goal to understand the computational complexity of all fixed-template PCSPs, a recent line of research has focused on restricted classes of templates, with the main directions being Boolean templates (i.e., templates on a two-element set) and symmetric templates (i.e., all relations in the template satisfy that if a tuple belongs to a relation then so do all its permutations).

Austrin, Guruswami, and Håstad [4] considered the $(1, g, k)$-SAT problem: Given an instance of k-SAT with the promise that there is an assignment satisfying at least g literals in each clause, find an assignment that satisfies at least one literal in each clause. They showed that this problem is NP-hard if $\frac{g}{k} < \frac{1}{2}$, and polynomial-time solvable otherwise. $(1, g, k)$-SAT is a Boolean PCSP with a (symmetric) template that includes the binary disequality relation and a relation containing all tuples of particular Hamming weights. The NP-hardness in [4] was proved via reduction from the label cover problem using the idea of polymorphisms lifted from CSPs to PCSPs. Building on the algebraic theory from [6], Brandts, Wrochna, and Živný [12] extended the classification of $(1, g, k)$-SAT to arbitrary finite domains.

Brakensiek and Guruswami [10] managed to classify all PCSPs over symmetric Boolean templates with the disequality relation as NP-hard or solvable in polynomial time. Ficak, Kozik, Olšák, and Stankiewicz [20] extended this result to all symmetric Boolean templates.

In very recent work, Barto, Battistelli, and Berg [5] explored symmetric PCSPs on three- and four-element domains.

While the approximate graph colouring problem remains open, hardness was proved under stronger assumptions (namely Khot's 2-to-1 Conjecture [26] for k-colourings with $k \geq 4$ and its non-standard variant for 3-colourings) by Dinur, Mossel, and Regev [17]. Guruswami and Sandeep [22] recently established this result under a weaker assumption, the so-called d-to-1 conjecture for any fixed $d \geq 2$. For approximate *hypergraph* colouring, another important PCSP, NP-hardness was established by Dinur, Regev, and Smyth [18]. There has been some recent progress on approximate graph colourings [29] and related PCSPs, e.g. approximate graph homomorphism problems [27, 29], and rainbow vs. normal hypergraph colourings [3].

Contributions Unlike most previous works, which focused on symmetric PCSPs, we investigate *non-symmetric* PCSPs. Our first motivation is that a classification of more concrete PCSP templates is needed to improve and extend the general algebraic theory from [6], for example by identifying new hardness and tractability criteria. At the moment, even an analogue of Schaefer's result, i.e., classifying all Boolean PCSPs, seems out of reach. Our second motivation is the pure beauty of the template $(\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE})$. While PCSP$(\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE})$ admits a polynomial-time algorithm [10, 9], tractability cannot be obtained via a "gadget reduction" to tractable finite-domain CSPs [6] or via a "local consistency checking" [2].

Let **t-in-k** denote the Boolean structure with a single relation of arity k that contains tuples with exactly t 1's and let **NAE** denote the Boolean structure with a single relation of arity k, which is always clear from the context, that contains all tuples except for the two all-equal tuples. (Previously in this section, we used **NAE** only with $k = 3$.) Consider the Boolean PCSP($\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE}$), which is a natural generalisation of PCSP($\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE}$). Similarly to PCSP($\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE}$), we have that PCSP($\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE}$) is a symmetric tractable PCSP.

We study the following two questions: Firstly, when can we add tuples to **t-in-k** (i.e., how can we weaken the promise) to keep the PCSP tractable? Secondly, when can we remove tuples from **NAE** (i.e., how can we strengthen the relation **NAE**) to keep the PCSP tractable? Note that both of these changes generally do not result in symmetric templates.

For the second question, we give a complete answer in Theorem 14: If t is odd, k is even, and tuples of only even Hamming weight are removed from **NAE**, the resulting PCSP is solvable in polynomial time. In all other cases, the resulting PCSP is NP-hard. Put differently, PCSP($\mathbf{t\text{-}in\text{-}k}, \mathbf{T}$) is tractable if only if $\mathbf{T} = \mathbf{NAE}$ or CSP($\mathbf{T}$) is tractable (assuming P$\neq$NP).

For the first question, we give a second-best possible answer in Theorem 11: If t is odd, k is even, and tuples of only odd Hamming weight are added to **t-in-k**, the resulting PCSP is tractable. In all other cases, the resulting PCSP is not solved by the combined basic LP and affine IP relaxation (BLP + AIP) of Brakensiek, Guruswami, Wrochna, and Živný [11], one of the currently strongest known algorithm for PCSPs. The power of this relaxation, both in terms of minions and polymorphism identities, is known [11]. It is consistent with the current knowledge that BLP + AIP could solve all tractable Boolean PCSPs. The only stronger algorithm than BLP + AIP studied in the context of PCSPs is CLAP [16], but its power is currently only known via a minion-theoretic characterisation (and not via a polymorphism characterisation). Similarly to PCSP($\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE}$), the PCSPs that we prove to be BLP + AIP-hard are not solvable by "local consistency checking" and via a "gadget reduction" to tractable finite-domain CSPs (cf. Remark 12).

One take-away message from our results is that the tractability of PCSP($\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE}$) is very fragile, which gives more support for its importance. Another message is that the PCSP templates obtained from the template ($\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE}$) by adding a single tuple are good candidates for testing and/or improving NP-hardness criteria for PCSPs. Finally, Proposition 16, while with a very simple proof, shows that the classification of Boolean symmetric PCSP templates ($\mathbf{A}, \mathbf{B}$) from [20] holds more generally and requires that only $\mathbf{A}$ should be symmetric.

2 Preliminaries

We denote by $[n]$ the set $\{1, 2, \ldots, n\}$. For a k-tuple $\mathbf{x}$, we write $\mathbf{x} = (x_1, \ldots, x_k)$. We denote by $\leq_p$ a polynomial-time many-one reduction and by $\equiv_p$ a polynomial-time many-one equivalence.

A *relational structure* is a tuple $\mathbf{A} = (A; R_1, \ldots, R_p)$, where A is a finite set called the *domain* of $\mathbf{A}$, and each R_i is a relation of arity $\mathrm{ar}(R_i) \geq 1$, that is, R_i is a non-empty subset of $A^{\mathrm{ar}(R_i)}$. A relational structure is *symmetric* if each relation in it is invariant under any permutation of coordinates. Two relational structures $\mathbf{A} = (A; R_1, \ldots, R_p)$ and $\mathbf{B} = (B; S_1, \ldots, S_q)$ have the same *signature* if $p = q$ and $\mathrm{ar}(R_i) = \mathrm{ar}(S_i)$ for every $i \in [p]$. In this case, a mapping $\phi : A \to B$ is called a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$, denoted by $\phi : \mathbf{A} \to \mathbf{B}$, if ϕ preserves all relations; that is, for every $i \in [p]$ and every tuple $\mathbf{x} \in R_i$, we have $\phi(\mathbf{x}) \in S_i$, where ϕ is applied component-wise. The existence of a homomorphism from

$\mathbf{A}$ to $\mathbf{B}$ is denoted by $\mathbf{A} \to \mathbf{B}$. A PCSP *template* is a pair $(\mathbf{A}, \mathbf{B})$ of relational structures over the same signature such that $\mathbf{A} \to \mathbf{B}$.

Definition 1. Let $(\mathbf{A}, \mathbf{B})$ be a PCSP template. The *decision version* of $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is the following problem: Given as input a relational structure $\mathbf{X}$ over the same signature as $\mathbf{A}$ and $\mathbf{B}$, output YES if $\mathbf{X} \to \mathbf{A}$ and NO if $\mathbf{X} \not\to \mathbf{B}$. The *search version* of $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is the following problem: Given as input a relational structure $\mathbf{X}$ over the same signature as $\mathbf{A}$ and $\mathbf{B}$ such that $\mathbf{X} \to \mathbf{A}$, find a homomorphism from $\mathbf{X}$ to $\mathbf{B}$.

We call $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *tractable* if any instance of $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ can be solved in polynomial time in the size of the input structure $\mathbf{X}$. It is easy to show that the decision version reduces to the search version [6]. Our hardness results will be for the decision version and our tractability results for the search version. For a relational structure $\mathbf{A}$, the constraint satisfaction problem with the template $\mathbf{A}$, denoted by $\mathrm{CSP}(\mathbf{A})$, is $\mathrm{PCSP}(\mathbf{A}, \mathbf{A})$.

The following notion of polymorphisms is at the heart of the algebraic approach to (P)CSPs.

Definition 2. Let $(\mathbf{A}, \mathbf{B})$ be a PCSP template. A function $f : A^m \to B$ is a *polymorphism* of arity m of $(\mathbf{A}, \mathbf{B})$ if for each pair of corresponding relations R_i and S_i from $\mathbf{A}$ and $\mathbf{B}$, respectively, the following holds: For any $(\mathrm{ar}(R_i) \times m)$ matrix M whose columns are tuples in R_i, the application of f to rows of M gives a tuple in S_i. In other words, an arity m polymorphism is a homomorphism from the m-th Cartesian power of $\mathbf{A}$ to $\mathbf{B}$. We denote by $\mathrm{Pol}(\mathbf{A}, \mathbf{B})$ the set of all polymorphisms of $(\mathbf{A}, \mathbf{B})$.

In a PCSP template $(\mathbf{A}, \mathbf{B})$ we view tuples from $\mathbf{A}$ and $\mathbf{B}$ as columns. When writing tuples in text we may write them as rows to simplify notation but they should still be understood as columns. For a k-ary relation R on the set A, we denote by $R^c = A^k \setminus R$ the complement of R. For a relational structure $\mathbf{A}$, we denote by $\mathbf{A}^c$ the structure with relations R^c for each relation R in $\mathbf{A}$. Most of our relational structures will be on the Boolean domain $\{0, 1\}$ and contain a single relation of arity k. The (Hamming) weight of a tuple $\mathbf{x} \in \{0, 1\}^k$, denoted throughout by d, is the number of 1's in $\mathbf{x}$. For $1 \leq t < k$, the Boolean relational structure **t-in-k** consists (of one relation consisting) of all k-tuples with weight t. The Boolean relational structure **NAE** contains all k-tuples except 0^k and 1^k.

We need a definition and some notation to state existing results on Boolean (P)CSPs.

Definition 3. A function $f : \{0, 1\}^m \to \{0, 1\}$ is

- an OR_m (AND_m) if it returns the logical OR (respectively logical AND) of its arguments;
- an alternating threshold AT_m if m is odd and
$$f(x_1, \ldots, x_m) = 1 \text{ if and only if } x_1 - x_2 + x_3 - \cdots + x_m > 0;$$
- a parity function XOR_m if $f(x_1, \ldots, x_m) = x_1 + \cdots + x_m \mod 2$;
- a q-threshold $\mathrm{THR}_{q,m}$ (for q a rational between 0 and 1 and mq not an integer) if $f(x_1, \ldots, x_m) = 0$ if $\sum_{i=1}^{m} x_i < mq$ and 1 otherwise;
- a majority MAJ_m if f is a $\frac{1}{2}$-threshold and m is odd.

We denote by OR and AND the set of all OR_m and AND_m functions, respectively, for all $m \geq 2$. We denote by AT and XOR the set of all AT_m and XOR_m functions, respectively, for odd $m \geq 1$. Finally, THR_q denotes the set of all $\mathrm{THR}_{q,m}$ functions for $qm \notin \mathbb{Z}$.

Define $\overline{f}$, the *negation of* f, as the function $x \mapsto 1 - f(x)$, and for a family of functions F, define the *negation of* F by $\overline{F} = \{\overline{f} | f \in F\}$.

Schaefer's dichotomy theorem [28] classified all Boolean CSP templates and can be stated in various forms (see e.g. [7] for further discussion). Here we give a modern formulation in terms of polymorphisms.

Theorem 4. *Let* $\mathbf{B}$ *be a Boolean CSP template. If* $\mathrm{Pol}(\mathbf{B})$ *contains a constant,* AND_2*,* OR_2*,* MAJ_3*, or* XOR_3*, then* $\mathrm{CSP}(\mathbf{B})$ *is tractable. Otherwise,* $\mathrm{CSP}(\mathbf{B})$ *is NP-hard.*

Ficak et al. classified all symmetric Boolean PCSP templates [20].

Theorem 5 ([20])**.** *Let* $(\mathbf{A}, \mathbf{B})$ *be a symmetric Boolean PCSP template. If* $\mathrm{Pol}(\mathbf{A}, \mathbf{B})$ *contains a constant or at least one of* OR*,* AND*,* XOR*,* AT*,* THR_q *(for some* q*) or their negations, then* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is tractable. Otherwise,* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is NP-hard.*

The only possibly unresolved promise templates are those with NP-hard CSP templates.

Proposition 6. *Let* $(\mathbf{A}, \mathbf{B})$ *be a promise template such that at least one of* $\mathrm{CSP}(\mathbf{A})$*,* $\mathrm{CSP}(\mathbf{B})$ *is tractable. Then* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is tractable.*

Proposition 6 is a direct consequence of the important concept of homomorphic relaxation, which we now define. Let $(\mathbf{A}, \mathbf{B})$ and $(\mathbf{A}', \mathbf{B}')$ be two PCSP templates over the same signature. We call $(\mathbf{A}', \mathbf{B}')$ a *homomorphic relaxation* of $(\mathbf{A}, \mathbf{B})$ if $\mathbf{A}' \to \mathbf{A}$ and $\mathbf{B} \to \mathbf{B}'$. It is easy to show [6] that $\mathrm{PCSP}(\mathbf{A}', \mathbf{B}') \leq_p \mathrm{PCSP}(\mathbf{A}, \mathbf{B})$.[1]

Proof of Proposition 6. We have $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}) \leq_p \mathrm{PCSP}(\mathbf{A}, \mathbf{A}) = \mathrm{CSP}(\mathbf{A})$, since $(\mathbf{A}, \mathbf{B})$ is a homomorphic relaxation of $(\mathbf{A}, \mathbf{A})$ as $\mathbf{A} \to \mathbf{B}$ by assumption. Similarly, $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}) \leq_p \mathrm{PCSP}(\mathbf{B}, \mathbf{B}) = \mathrm{CSP}(\mathbf{B})$, since $(\mathbf{A}, \mathbf{B})$ is a homomorphic relaxation of $(\mathbf{B}, \mathbf{B})$ as $\mathbf{A} \to \mathbf{B}$ by assumption. ☐

Theorem 4 established NP-hardness of two natural CSPs: CSP(**1-in-3**) and CSP(**NAE**). Interestingly, PCSP(**1-in-3**, **NAE**) is solvable in polynomial-time, as first shown by Brakensiek and Guruswami [10]. (This shows that the converse of Proposition 6 is false.) A natural generalisation of **1-in-3** is **t-in-k**. Theorem 4 implies that CSP(**t-in-k**) is NP-hard, which also follows from Proposition 25. Theorem 5 implies that the tractability of PCSP(**1-in-3**, **NAE**) also holds for PCSP(**t-in-k**, **NAE**).

Proposition 7. *For* $k \geq 3$ *and* $1 \leq t < k$, CSP(**t-*in*-k**) *is NP-hard.*

Proposition 8. *For* $k \geq 2$ *and* $1 \leq t < k$, PCSP(**t-*in*-k**, **NAE**) *is tractable.*

[1]In fact, more is known: The trivial (identity) reduction from $\mathrm{PCSP}(\mathbf{A}', \mathbf{B}')$ to $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is correct if and only if $(\mathbf{A}', \mathbf{B}')$ is a homomorphic relaxation of $(\mathbf{A}, \mathbf{B})$.

2.1 Algorithms

We now present three relaxations for PCSPs: BLP, AIP, and BLP + AIP. The first one, BLP, is needed for the description of the third one. The second one, AIP, solves all tractable cases in our classification results. Finally, the third one, BLP + AIP, is one of the strongest known algorithms for PCSPs and the strongest one with a characterisation of its power in terms of polymorphism identities. Our "algorithmic dichotomy" result (Theorem 11) shows AIP solvability vs. BLP + AIP-hardness.

In the rest of this section, let $(\mathbf{A}, \mathbf{B})$ be a PCSP template, where $\mathbf{A} = (A; R_1, \dots, R_p)$ and $\mathbf{B} = (B; S_1, \dots, S_p)$. Let $\mathbf{X} = (X; T_1, \dots, T_p)$ be an instance of PCSP$(\mathbf{A}, \mathbf{B})$. We assume without loss of generality that all three structures contain a unary relation equal to X in $\mathbf{X}$, equal to A in $\mathbf{A}$, and equal to B in $\mathbf{B}$; the relation is called R_u in $\mathbf{A}$. If this is not the case, the template and the instance can be extended without changing the set of solutions.

The *basic linear programming relaxation* (BLP) of $\mathbf{X}$, denoted by BLP$(\mathbf{X}, \mathbf{A})$, is defined as follows. The variables are $\lambda_{\mathbf{x},i}(\mathbf{a})$ for every $i \in [p]$, $\mathbf{x} \in T_i$, and $\mathbf{a} \in R_i$, and the constraints are given in Figure 1. (Note that BLP$(\mathbf{X}, \mathbf{A})$ does not depend on $\mathbf{B}$.)

$$0 \leq \lambda_{\mathbf{x},i}(\mathbf{a}) \leq 1 \qquad \forall i \in [p], \forall \mathbf{x} \in T_i, \forall \mathbf{a} \in R_i \tag{1}$$

$$\sum_{\mathbf{a} \in R_i} \lambda_{\mathbf{x},i}(\mathbf{a}) = 1 \qquad \forall i \in [p], \forall \mathbf{x} \in T_i \tag{2}$$

$$\sum_{\mathbf{a} \in R_i, a_j = a} \lambda_{\mathbf{x},i}(\mathbf{a}) = \lambda_{x_j, R_u}(a) \qquad \forall i \in [p], \forall \mathbf{x} \in T_i, \forall a \in A, \forall j \in [\mathrm{ar}(R_i)] \tag{3}$$

Figure 1: Definition of BLP$(\mathbf{A}, \mathbf{X})$.

The *basic affine integer programming relaxation* (AIP) of $\mathbf{X}$, denoted by AIP$(\mathbf{X}, \mathbf{A})$, is defined as follows. The variables are $\tau_{\mathbf{x},i}(\mathbf{a})$ for every $i \in [p]$, $\mathbf{x} \in T_i$, and $\mathbf{a} \in R_i$, and the constraints are given in Figure 2.

$$\tau_{\mathbf{x},i}(\mathbf{a}) \in \mathbb{Z} \qquad \forall i \in [p], \forall \mathbf{x} \in T_i, \forall \mathbf{a} \in R_i \tag{4}$$

$$\sum_{\mathbf{a} \in R_i} \tau_{\mathbf{x},i}(\mathbf{a}) = 1 \qquad \forall i \in [p], \forall \mathbf{x} \in T_i \tag{5}$$

$$\sum_{\mathbf{a} \in R_i, a_j = a} \tau_{\mathbf{x},i}(\mathbf{a}) = \tau_{x_j, R_u}(a) \qquad \forall i \in [p], \forall \mathbf{x} \in T_i, \forall a \in A, \forall j \in [\mathrm{ar}(R_i)] \tag{6}$$

Figure 2: Definition of AIP$(\mathbf{A}, \mathbf{X})$.

We say that AIP$(\mathbf{X}, \mathbf{A})$ accepts if the affine program in Figure 2 is feasible, and rejects otherwise. By construction, if $\mathbf{X} \to \mathbf{A}$ then AIP$(\mathbf{X}, \mathbf{A})$ accepts. We say that AIP *solves* PCSP$(\mathbf{A}, \mathbf{B})$ if for every instance $\mathbf{X}$ accepted by AIP$(\mathbf{X}, \mathbf{A})$ we have $\mathbf{X} \to \mathbf{B}$.

A $(2m+1)$-ary function $f : A^{2m+1} \to B$ is called *alternating* if $f(a_1, \dots, a_{2m+1}) = f(a_{\pi(1)}, \dots, a_{\pi(2m+1)})$ for every $a_1, \dots, a_{2m+1} \in A$ and every permutation $\pi : [2m+1] \to [2m+1]$ that preserves parity, and $f(a_1, \dots, a_{2m-1}, a, a) = f(a_1, \dots, a_{2m-1}, a', a')$ for every

$a_1, \ldots, a_{2m-1}, a, a' \in A$. Intuitively, an alternating function is invariant under permutations of its odd and even coordinates and has the property that adjacent coordinates cancel each other out. The power of AIP for PCSPs is characterised by the following result.[2]

Theorem 9 ([6]). *Let* $(\mathbf{A}, \mathbf{B})$ *be a PCSP template. Then (the decision version of)* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is tractable via* AIP *if and only if* $\mathrm{Pol}(\mathbf{A}, \mathbf{B})$ *contains alternating functions of all odd arities.*

The *combined basic LP and affine IP algorithm* (BLP + AIP) [11] is presented in Algorithm 1. If $\mathbf{X} \to \mathbf{A}$ then BLP + AIP accepts $\mathbf{X}$ [11]. We say that BLP + AIP *solves* PCSP$(\mathbf{A}, \mathbf{B})$ if for every instance $\mathbf{X}$ accepted by BLP + AIP we have $\mathbf{X} \to \mathbf{B}$.

Algorithm 1: The BLP + AIP algorithm

```
Input:   an instance $\mathbf{X}$ of PCSP$(\mathbf{A}, \mathbf{B})$
Output:  YES if $\mathbf{X} \to \mathbf{A}$ and NO if $\mathbf{X} \not\to \mathbf{B}$
1 find a relative interior point $(\lambda_{\mathbf{x},i}(\mathbf{a}))_{i \in [p], \mathbf{x} \in T_i, \mathbf{a} \in R_i}$ of BLP$(\mathbf{X}, \mathbf{A})$;
2 if no relative interior point exists then
3     return NO;
4 end
5 refine AIP$(\mathbf{X}, \mathbf{A})$ by setting $\tau_{\mathbf{x},i}(\mathbf{a}) = 0$ if $\lambda_{\mathbf{x},i}(\mathbf{a}) = 0$;
6 if the refined AIP$(\mathbf{X}, \mathbf{A})$ accepts then
7     return YES;
8 end
9 return NO;
```

A $(2m+1)$-ary function $f : A^{2m+1} \to B$ is called 2*-block-symmetric* if $f(a_1, \ldots, a_{2m+1}) = f(a_{\pi(1)}, \ldots, a_{\pi(2m+1)})$ for every $a_1, \ldots, a_{2m+1} \in A$ and every permutation $\pi : [2m+1] \to [2m+1]$ that preserves parity. In other words, f is 2-block-symmetric if its $2m+1$ coordinates can be partitioned into two blocks of size $m+1$ and m in such a way that the value of f is invariant under any permutation of coordinates within each block. Without loss of generality, we will assume that the two blocks are the odd and even coordinates of f.

The power of BLP + AIP for PCSPs is characterised by the following result.[3]

Theorem 10 ([11]). *Let* $(\mathbf{A}, \mathbf{B})$ *be a PCSP template. Then (the decision version of)* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is tractable via* BLP + AIP *if and only if* $\mathrm{Pol}(\mathbf{A}, \mathbf{B})$ *contains 2-block-symmetric functions of all odd arities.*

3 Results

Our results are concerned with templates that arise from $(\textbf{t-in-k}, \mathbf{NAE})$ either by adding tuples to **t-in-k** or removing tuples from **NAE**. For a set of tuples $S \subseteq \{0,1\}^k$, we write $\textbf{t-in-k} \cup \mathbf{S}$ for the relational structure whose (only) relation contains all k-tuples of weight t and the tuples from S, and similarly for $\mathbf{NAE} \setminus \mathbf{S}$.

Our first result is an algorithmic dichotomy for templates constructed by adding tuples to **t-in-k**.

[2]We note that [6] proves several other equivalent statements in Theorem 9.

[3]We note that [11] proves several other equivalent statements in Theorem 10.

Theorem 11 (**Main #1**). *Let $k \geq 3$ and $\emptyset \neq S \subseteq (\mathbf{t\text{-}\textit{in}\text{-}k})^c \cap \mathbf{NAE}$. If t is odd, k is even, and S contains tuples of only odd weight, then* $\mathrm{PCSP}(\mathbf{t\text{-}\textit{in}\text{-}k} \cup \mathbf{S}, \mathbf{NAE})$ *is tractable via* AIP. *Otherwise,* $\mathrm{PCSP}(\mathbf{t\text{-}\textit{in}\text{-}k} \cup \mathbf{S}, \mathbf{NAE})$ *is not solved by* $\mathrm{BLP} + \mathrm{AIP}$.

Ruling out the applicability of $\mathrm{BLP} + \mathrm{AIP}$ as stated in Theorem 11 is done, via Theorem 10, in Section 4.

Remark 12. Barto et al. [6] showed that $\mathrm{PCSP}(\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE})$ is not "finitely tractable", meaning that there is no finite $\mathbf{C}$ such that $\mathbf{1\text{-}in\text{-}3} \to \mathbf{C} \to \mathbf{NAE}$ and $\mathrm{CSP}(\mathbf{C})$ is tractable. In other words, the tractability of $\mathrm{PCSP}(\mathbf{1\text{-}in\text{-}3}, \mathbf{NAE})$ cannot be achieved via a "gadget reduction" to tractable finite-domain CSPs. This result was then extended by Asimi and Barto [1] to $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE})$ for $k \geq 3$, $t < k$ when t is even or k is odd. Since the $\mathrm{BLP} + \mathrm{AIP}$-hard cases in Theorem 11 are homomorphically sandwiched by templates proved finitely intractable in [1], they are also finitely intractable.

A recent result of Atserias and Dalmau that gives a necessary condition for PCSPs to be solvable by a "local consistency checking" algorithm [2] implies that all templates from Theorem 11 (and in particular those not solved by $\mathrm{BLP} + \mathrm{AIP}$) are *not* solved by a "local consistency checking" algorithm. By [2, Corollary 4.2], such an algorithm does not solve $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE})$ for any $k \geq 3$ and $t < k$, and since $(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE})$ is a homomorphic relaxation of the templates from Theorem 11, our claim follows from [6, Lemma 7.5].

Our second result is a complexity dichotomy for templates constructed by removing tuples from $\mathbf{NAE}$. The key result here is the following.

Theorem 13. *Let $k \geq 3$ and let $\mathbf{T} \subseteq \{0,1\}^k$ be a relation such that $\mathbf{t\text{-}\textit{in}\text{-}k} \to \mathbf{T}$ and* $\mathrm{CSP}(\mathbf{T})$ *is NP-hard. Then* $\mathrm{PCSP}(\mathbf{t\text{-}\textit{in}\text{-}k}, \mathbf{T})$ *is tractable if and only if $\mathbf{T} = \mathbf{NAE}$, unless P=NP.*

In other words, $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{T})$ is tractable if $\mathrm{CSP}(\mathbf{T})$ is tractable or $\mathbf{T} = \mathbf{NAE}$, and is NP-hard otherwise. Theorem 13 then easily implies the following.

Theorem 14 (**Main #2**). *Let $k \geq 3$ and $\emptyset \neq S \subseteq (\mathbf{t\text{-}\textit{in}\text{-}k})^c \cap \mathbf{NAE}$. If t is odd, k is even, and S contains tuples of only even weight, then* $\mathrm{PCSP}(\mathbf{t\text{-}\textit{in}\text{-}k}, \mathbf{NAE} \setminus \mathbf{S})$ *is tractable. Otherwise,* $\mathrm{PCSP}(\mathbf{t\text{-}\textit{in}\text{-}k}, \mathbf{NAE} \setminus \mathbf{S})$ *is NP-hard.*

Theorem 13 is proved in Section 5 and relies on Theorem 5, as well as a symmetrisation trick (Proposition 16, observed independently in [5]) and the following observation.

Proposition 15. *Let R be a symmetric relation on a set A. For any function $f : A \to B$, the component-wise image of R under f, denoted $f(R)$, is a symmetric relation on B.*

Proof. Suppose that $\mathbf{y} \in f(R)$, so $\mathbf{y} = f(\mathbf{x})$ for some $\mathbf{x} \in R$. We must show that $\pi(\mathbf{y}) \in f(R)$ for an arbitrary permutation π. But since R is symmetric, we have $\pi(\mathbf{x}) \in R$, and so $f(\pi(\mathbf{x})) = \pi(\mathbf{y})$ since f is applied component-wise. □

Proposition 16. *Let $(\mathbf{A}, \mathbf{B})$ be a PCSP template with $\mathbf{A}$ symmetric. For each relation $R \in \mathbf{B}$, let R' be the largest symmetric relation contained in R. Let $\mathbf{B}'$ be the relational structure with the same domain as $\mathbf{B}$ but with relations R' instead of R. Then* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is polynomial-time equivalent to* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}')$.

Proof. We first check that $(\mathbf{A}, \mathbf{B}')$ is a valid PCSP template, i.e., that there is a homomorphism $\mathbf{A} \to \mathbf{B}'$. Let ϕ be a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. By Proposition 15, $\phi(\mathbf{A})$ is symmetric, and since $\mathbf{B}'$ is the largest symmetric relational structure contained in $\mathbf{B}$, we have $\phi(\mathbf{A}) \subseteq \mathbf{B}'$. Therefore $(\mathbf{A}, \mathbf{B}')$ is a valid PCSP template. For $f \in \mathrm{Pol}(\mathbf{A}, \mathbf{B})$, $f(\mathbf{A})$ is symmetric and is contained in $\mathbf{B}$, so $f(\mathbf{A}) \subseteq \mathbf{B}'$ and $\mathrm{Pol}(\mathbf{A}, \mathbf{B}) \subseteq \mathrm{Pol}(\mathbf{A}, \mathbf{B}')$. The reverse inclusion follows from $\mathbf{B}' \subseteq \mathbf{B}$ and gives $\mathrm{Pol}(\mathbf{A}, \mathbf{B}) = \mathrm{Pol}(\mathbf{A}, \mathbf{B}')$, which implies by [6, Theorem 3.1] that $\mathrm{PCSP}(\mathbf{A}, \mathbf{B}) \equiv_p \mathrm{PCSP}(\mathbf{A}, \mathbf{B}')$. $\square$

The tractability parts in Theorem 11 and Theorem 14 follow easily from existing work, as we now show. Using (the sufficiency of) Theorem 9, it is easy to establish Proposition 8, i.e., tractability of $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE})$. Indeed, by [10, Claim 4.6], the AT family maps collections of $\mathbf{t\text{-}in\text{-}k}$ tuples into $\mathbf{NAE}$, and so $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE})$ is tractable.

Proposition 17. *For k even, (the search version of)* $\mathrm{CSP}(\mathbf{odd\text{-}in\text{-}k})$ *is tractable.*

Proof. We claim that $\mathrm{XOR}_3 \in \mathrm{Pol}(\mathbf{odd\text{-}in\text{-}k})$ so that tractability will follow from Theorem 4. To see that $\mathrm{XOR}_3 \in \mathrm{Pol}(\mathbf{T})$, suppose that XOR_3 returns a tuple of even weight d. Then in the $k \times 3$ matrix of inputs with three odd weight tuples as columns, there are d rows with an odd number of 1's and $k - d$ rows with an even number of 1's. Together these give an even total number of 1's in the matrix. But since the three input columns have odd weight, the total number of 1's in the matrix is odd. Contradiction. $\square$

Proof of the tractability part of Theorems 11 and 14. Under the tractability criterion of Theorem 11, $\mathbf{t\text{-}in\text{-}k} \cup \mathbf{S} \subseteq \mathbf{odd\text{-}in\text{-}k} \subseteq \mathbf{NAE}$ and thus $(\mathbf{t\text{-}in\text{-}k} \cup \mathbf{S}, \mathbf{NAE})$ is a homomorphic relaxation of $(\mathbf{odd\text{-}in\text{-}k}, \mathbf{odd\text{-}in\text{-}k})$. As discussed in Section 2 (and proved in [6]), this implies that $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k} \cup \mathbf{S}, \mathbf{NAE}) \leq_p \mathrm{PCSP}(\mathbf{odd\text{-}in\text{-}k}, \mathbf{odd\text{-}in\text{-}k}) = \mathrm{CSP}(\mathbf{odd\text{-}in\text{-}k})$, where $\mathrm{CSP}(\mathbf{odd\text{-}in\text{-}k})$ is tractable by Proposition 17 (for even k). Similarly, under the tractability criterion of Theorem 14, we have $\mathbf{t\text{-}in\text{-}k} \subseteq \mathbf{odd\text{-}in\text{-}k} \subseteq \mathbf{NAE} \setminus \mathbf{S}$ and thus $\mathrm{PCSP}(\mathbf{t\text{-}in\text{-}k}, \mathbf{NAE} \setminus \mathbf{S}) \leq_p \mathrm{CSP}(\mathbf{odd\text{-}in\text{-}k})$. By composing XOR_3 functions from the proof of Proposition 17, or by observing that $\mathbf{odd\text{-}in\text{-}k}$ is an affine subspace, we have $\mathrm{XOR} \subseteq \mathrm{Pol}(\mathbf{odd\text{-}in\text{-}k})$, which implies via the inclusion that our PCSP templates have the XOR family of polymorphisms and thus are solvable by AIP. $\square$

4 Adding tuples

The following result implies, by Theorem 10, the non-tractability part of Theorem 11.

Theorem 18. *Let $k \geq 3$, $1 \leq t < k$, and $\mathbf{x}$ be a k-tuple of weight $1 \leq d < k$ with $d \neq t$. Then,* $(\mathbf{t\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ *does not have 2-block-symmetric polymorphisms of all odd arities, unless t is odd, k is even, and d is odd.*

The implication is as follows: In the non-tractability case of Theorem 11, $\mathbf{S}$ contains a tuple $\mathbf{x}$ of weight d such that if d is odd, then t is even, k is odd, or both. Therefore $\mathrm{Pol}(\mathbf{t\text{-}in\text{-}k} \cup \mathbf{S}, \mathbf{NAE}) \subseteq \mathrm{Pol}(\mathbf{t\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$, so it suffices to rule out 2-block-symmetric polymorphisms for templates of the form $(\mathbf{t\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$.

We start with two simple observations which reduce the number of cases to deal with. Since permuting the rows of a matrix of inputs to a polymorphism permutes the values of the output tuple and does not affect membership in the symmetric $\mathbf{NAE}$ relation, we have the following.

Observation 19. *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be two* k*-tuples of weight* d*. Then,* $\mathrm{Pol}(\mathbf{t}\text{-}\boldsymbol{in}\text{-}\mathbf{k} \cup \{\mathbf{x}\}, \mathbf{NAE}) = \mathrm{Pol}(\mathbf{t}\text{-}\boldsymbol{in}\text{-}\mathbf{k} \cup \{\mathbf{y}\}, \mathbf{NAE})$*.*

By Observation 19, it suffices to prove Theorem 18 for $\mathbf{x}$ of the form $\mathbf{x} = 1^d 0^{k-d}$.

Observation 20. *There is a bijection between* $\mathrm{Pol}(\mathbf{t}\text{-}\boldsymbol{in}\text{-}\mathbf{k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ *and* $\mathrm{Pol}((\mathbf{k}-\mathbf{t})\text{-}\boldsymbol{in}\text{-}\mathbf{k} \cup \{\overline{\mathbf{x}}\}, \mathbf{NAE})$ *given by* $f(x_1, \ldots, x_m) \mapsto f(1 - x_1, \ldots, 1 - x_m)$*, where* $\overline{\mathbf{x}}$ *is the negation of* $\mathbf{x}$*.*

Observation 20 implies that 2-block-symmetry of polymorphisms is preserved when swapping 0's and 1's.

There are eight combinations of the parities of k, t, and d. The case $(k, t, d) \equiv (0, 1, 1)$ is out of the scope of Theorem 18 and is covered by the tractable case of Theorem 11. The case $(k, t, d) \equiv (1, 0, 0)$ is covered for $d > t$ in Proposition 23 and $d < t$ in Proposition 24, and all other cases are covered for $d > t$ in Proposition 21 and $d < t$ in Proposition 22. By applying Observation 20, we may assume that $d + t \leq k$, which allows a single construction to work in each of these propositions. We start with a brief account of the idea behind the proofs.

Let C_k^t be the $k \times k$ matrix containing the k cyclic shifts of the column $1^t 0^{k-t}$. The matrix C_k^t can be used to fill one of the coordinate blocks of a 2-block-symmetric function f of arity $2k \pm 1$. For example, suppose that C_k^t is used to fill the "first" coordinate block. It does not matter whether the first block contains the odd or even coordinates. Then f depends only on the weights in each row of the "second" block, since the first block has the same weight in every row. This allows f to be analysed as a symmetric (1-block-symmetric) function.

For each k, t, and d, and for any f such that one of its blocks can be filled by C_k^t, we exhibit a set of tableaux for the other block that prevents f from being a polymorphism. Suppose we have filled one block with C_k^t, so that f can now be represented as a unary function of the weight on its other block. For any weights w_1, w_2, w_3, we have at least one of $f(w_1) = f(w_2)$, $f(w_1) = f(w_3)$, and $f(w_2) = f(w_3)$. For each pair of weights, we construct a tableau where each row of the second block is one of the two weights. Thus we are guaranteed that f will return an all-equal tuple and hence not be a polymorphism.

Proposition 21. *Let* $k \geq 3$*,* $1 \leq t < k$*, and* $t < d < k$ *be such that* $d + t \leq k$*, and* $t \equiv k$ *or* $d \not\equiv t \pmod 2$*. Let* $\mathbf{x}$ *be a tuple of weight* d*. Then* $\mathrm{Pol}(\mathbf{t}\text{-}\boldsymbol{in}\text{-}\mathbf{k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ *does not have 2-block-symmetric functions of arity* $2k - 1$*.*

Proof. Let f be a 2-block-symmetric function of arity $2k - 1$. We will show that f is not a polymorphism of $(\mathbf{t}\text{-}\mathbf{in}\text{-}\mathbf{k} \cup \{\mathbf{x}\}, \mathbf{NAE})$. Let the odd block contain the tableau C_k^t and denote by C_k^{t-} the tableau obtained from C_k^t by removing its last column. We describe how to construct the even block with $k - 1$ columns in the cases below. We use $t - 1$, t, and $t + 1$ as our three weights.

Case 1: weights $t - 1$ and t.

The even tableau is C_k^{t-}. Thus each row in the even block has weight either $t - 1$ or t. If $f(t - 1) = f(t)$ then f returns an all-equal tuple 0^k or 1^k, so $f \notin \mathrm{Pol}(\mathbf{t}\text{-}\mathbf{in}\text{-}\mathbf{k} \cup \{\mathbf{x}\}, \mathbf{NAE})$.

Case 2: weights $t - 1$ and $t + 1$.

We take C_k^{t-} and replace some of the $\mathbf{t}\text{-}\mathbf{in}\text{-}\mathbf{k}$ tuples with $\mathbf{x}$ as necessary.

Case 2a: $t \equiv k \pmod 2$.

The tableau C_k^{t-} has an even number $k - t$ of rows with weight t. These can be paired up and 1's exchanged so that each row has weight either $t - 1$ or $t + 1$. In particular, in the columns $t{+}1, t{+}3, \ldots, k{-}1$, we swap the values in the pairs of rows $(t, t{+}1), (t{+}2, t{+}3), \ldots, (k{-}2, k{-}1)$, respectively. An illustration is given in Figure (3a).

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & \mathbf{0} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & \mathbf{0} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

(a) Case 2a with $t = 3$.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) Case 2b with $t = 2$ and $d = 5$.

Figure 3: Example with $k = 9$ for $f(t-1) = f(t+1)$. Swapped values in bold.

Case 2b: $(k, t, d) \equiv (0, 1, 0)$ or $(1, 0, 1) \pmod 2$.
The tableau C_k^{t-} has an odd number $k - t$ of rows with weight t. We replace the first column with $\mathbf{x}$. This adds $d - t$ 1's to the first column, and the $d - t$ rows with the added 1's now have weight $t + 1$. There remains an even number $k - d$ of rows of weight t, which can be paired up to exchange 1's and achieve weight $t - 1$ or $t + 1$ in each row. In particular, we swap the values at positions $(t, 2)$ and $(d + 1, 2)$, and then in the columns $d + 3, d + 5, \ldots, k - 1$, we swap the values in the pairs of rows $(d+2, d+3), (d+4, d+5), \ldots, (k-2, k-1)$, respectively. An illustration is given in Figure (3b).

Cases 2a and 2b cover all possible parities of k, t, and d under the proposition's assumptions. In both cases, each row in the even block has weight either $t - 1$ or $t + 1$. If $f(t-1) = f(t+1)$ then f returns an all-equal tuple, so $f \notin \mathrm{Pol}(\textbf{t-in-k} \cup \{\mathbf{x}\}, \mathbf{NAE})$.
Case 3: weights t and $t + 1$.

We first give a general description of the tableau, and then derive values for its parameters. We place the tuple $\mathbf{x}$ in the first r columns and fill the remaining $k - 1 - r$ columns with **t-in-k** tuples that have specific behaviours in the upper d rows and lower $k - d$ rows. Our goal is to distribute the weight of the **t-in-k** tuples between these two blocks of rows so that every row in the full tableau has weight t or $t + 1$.

Denoting by a the average column weight within the upper group of rows, we place either b or $b + 1$ 1's from each **t-in-k** tuple in the upper block, where b is an integer close to a. More precisely, to fill the upper d rows, we use $0 \leq s \leq k - 1 - r$ columns of weight b and $k - 1 - r - s$ columns of weight $b + 1$, and in the lower block of rows, we use s columns of weight $t - b$ and $k - 1 - r - s$ columns of weight $t - (b + 1)$, respectively, so that the full columns are **t-in-k** tuples.

We now describe the position of the 1's in the upper group of rows; the construction for the lower group is analogous. We fill columns with 1's from top to bottom, starting at the left-most column, and moving to the right after placing a column's quota of 1's (either b or $b + 1$). The order of the weight b and $b + 1$ tuples does not matter. When moving right to the next column, we continue placing 1's in the row immediately below the lowest row containing a 1 in the previous column. Once we reach the bottom of the group of rows, we wrap around to the top and continue in this way.

A $k \times (k - 1)$ tableau containing only **t-in-k** tuples needs at least t more 1's to achieve weight $\geq t$ in each row. Each time we replace a **t-in-k** tuple with $\mathbf{x}$, the tableau gains $d-t$ 1's, and therefore at least $r = \left\lceil \frac{t}{d-t} \right\rceil$ occurrences of $\mathbf{x}$ are necessary. This turns out to be sufficient. The remainder of the proof is devoted to showing that there exist a, b, and s which allow our construction to work.

A crucial observation connecting the column and row weights in our tableau is that within each block of rows, the weight between rows varies by at most one. It therefore suffices for the *average* row weight in each block to be between t and $t+1$.

To achieve average row weight at least t, a must be such that the total weights in the upper and lower blocks are at least $d(t-r)$ and $t(k-d)$, respectively. This is guaranteed when both $a(k-1-r) \geq d(t-r)$ and $(t-a)(k-1-r) \geq t(k-d)$, or equivalently,

$$\frac{d(t-r)}{k-1-r} \;\leq\; a \;\leq\; \frac{t(d-r-1)}{k-1-r}. \tag{7}$$

To achieve average row weight at most $t+1$, a must be such that the total weights in the upper and lower blocks are at most $d(t+1-r)$ and $(k-d)(t+1)$, respectively. This is guaranteed when both $a(k-1-r) \leq d(t+1-r)$ and $(t-a)(k-1-r) \leq (k-d)(t+1)$, or equivalently,

$$\frac{t(d-r-1)-k+d}{k-1-r} \;\leq\; a \;\leq\; \frac{d(t-r+1)}{k-1-r}. \tag{8}$$

Finally, to ensure that each block of rows is tall enough to accommodate the 1's specified in the construction, it suffices that $0 \leq a \leq d$ and $0 \leq t-a \leq k-d$, which together are equivalent to $\max(0, d+t-k) \leq a \leq \min(d,t)$. By our assumptions, this reduces to $0 \leq a \leq t$.

We now show that these inequalities can all be simultaneously satisfied. In 7, the upper bound is at least the lower bound if and only if $r \geq \frac{t}{d-t}$, which holds for our choice of r, and in 8, the upper bound is at least the lower bound if and only if $r \leq \frac{t}{d-t} + \frac{k}{d-t}$, which holds since $\frac{k}{d-t} \geq 1$. Exchanging the upper/lower bound pairs in 7 and 8 results in two pairs of inequalities on a that are always satisfied. Finally, for $0 \leq a \leq t$, it suffices that at least one of the lower bounds is nonnegative, and at least one of the upper bounds is at most t. The lower bound in 7 is nonnegative if and only if $t \geq r$, which always holds, and the upper bound is at most t if and only if $d \leq k$, which also always holds.

Therefore there exists $0 \leq a \leq t$ satisfying 7 and 8, and since these inequalities are not strict, we can take a to be rational with denominator $k-1-r$. Let $b = \lfloor a \rfloor$. Recalling that s is the number of columns of weight b in the upper block, computing the total weight in the upper block gives $sb + (k-1-r-s)(b+1) = a(k-1-r)$, so that $s = (k-1-r)(b+1-a)$. This is an integer since a is a fraction with denominator $k-1-r$. As a sanity check, note that if $a = b$ or $a = b+1$, then $s = k-1-r$ or $s = 0$, respectively.

We have shown that there exist a, b, and s which permit us to construct the tableau with weight t or $t+1$ in each row. Thus if $f(t) = f(t+1)$, then f returns the all-equal tuple 0^k or 1^k, so $f \notin \mathrm{Pol}(\textbf{t-in-k} \cup \{\mathbf{x}\}, \mathbf{NAE})$. This ends the proof of Case 3.

Since we must have at least one of $f(t-1) = f(t)$, $f(t-1) = f(t+1)$, and $f(t) = f(t+1)$, the three cases complete the proof.

An example with $t = 7$, $k = 15$, and $d = 10$ is illustrated in Figure 4. In this case, we have $r = \left\lceil \frac{t}{d-t} \right\rceil = 3$ and we get the inequalities $\frac{40}{11} \leq a \leq \frac{42}{11}$ and $\frac{37}{11} \leq a \leq \frac{50}{11}$. We take $a = \frac{41}{11}$; the values $\frac{40}{11}$ and $\frac{42}{11}$ would also work. Then $b = 3$ and $s = 3$, so in the upper group we have 3 columns of weight $b = 3$ and 8 columns of weight $b + 1 = 4$. The columns containing $\mathbf{x}$ are shown in addition to the construction on $k - 1 - r$ columns.

$$\left(\begin{array}{ccc|ccccccccccc}
1 & 1 & 1 & 1 & & & 1 & & 1 & & & 1 & & 1 \\
1 & 1 & 1 & 1 & & & 1 & & & 1 & & 1 & & \\
1 & 1 & 1 & 1 & & & 1 & & & 1 & & 1 & & \\
1 & 1 & 1 & & 1 & & & 1 & & 1 & & & 1 & \\
1 & 1 & 1 & & 1 & & & 1 & & 1 & & & 1 & \\
1 & 1 & 1 & & 1 & & & 1 & & & 1 & & 1 & \\
1 & 1 & 1 & & & 1 & & 1 & & & 1 & & 1 & \\
1 & 1 & 1 & & & 1 & & & 1 & & 1 & & & 1 \\
1 & 1 & 1 & & & 1 & & & 1 & & 1 & & & 1 \\
1 & 1 & 1 & & & & 1 & & 1 & & & 1 & & 1 \\
\hline
 & & & 1 & 1 & 1 & & 1 & 1 & & 1 & & 1 & 1 \\
 & & & 1 & 1 & 1 & & 1 & & 1 & 1 & & 1 & \\
 & & & 1 & 1 & & 1 & 1 & & 1 & & 1 & 1 & \\
 & & & 1 & & 1 & 1 & & 1 & 1 & & 1 & & 1 \\
 & & & & 1 & 1 & 1 & & 1 & & 1 & 1 & & 1
\end{array}\right)$$

Figure 4: Example with $t = 7$, $k = 15$, and $d = 10$ from Case 3, for $f(t) = f(t + 1)$.

$\square$

Proposition 22. *Let $k \geq 3$, $1 < t < k$, and $1 \leq d < t$ be such that $d + t \leq k$, and $t \equiv k$ or $d \not\equiv t \pmod 2$. Let $\mathbf{x}$ be a tuple of weight d. Then $\mathrm{Pol}(\mathbf{t\text{-}in\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ does not have 2-block-symmetric functions of arity $2k + 1$.*

Proof. The proof is similar to the case $d > t$ established in Proposition 21, except that now we can reduce the number of 1's in the tableaux by replacing **t-in-k** tuples with $\mathbf{x}$. We place the tableau C_k^t in the even coordinates, so that there are $k + 1$ columns to be filled in the odd coordinates. As before, we give tableaux for the three pairs of weights from $t - 1$, t, and $t + 1$.

Let C_k^{t+} be the $k \times (k + 1)$ matrix C_k^t with an extra column $1^t 0^{k-t}$.

Case 1: weights t and $t + 1$.
The odd tableau is C_k^{t+}, so each row in the odd block has weight either t or $t + 1$.

Case 2: weights $t - 1$ and $t + 1$.
The tableaux are similar to Case 2 in the proof of Proposition 21. When $t \equiv k$, we modify C_k^{t+} in the columns $t+2, t+4, \ldots, k$ by swapping the values in the pairs of rows $(t+1, t+2), (t+3, t+4), \ldots, (k-1, k)$, respectively. When $(k, t, d) \equiv (0, 1, 0)$ or $(1, 0, 1)$, we replace the first column of C_k^{t+} with $\mathbf{x}$, which leaves an even number $k - d$ of rows of weight t. Therefore in columns $d+2, d+4, \ldots, k$ we swap the values in the pairs of rows $(d+1, d+2), (d+3, d+4), \ldots, (k-1, k)$, respectively, to get weight $t - 1$ or $t + 1$ in each row.

Case 3: weights $t - 1$ and t.
This case is similar to Case 3 in the proof of Proposition 21. The tableau C_k^{t+} has t rows with weight $t + 1$ and $k - t$ rows with weight t, so we must reduce the total weight by at least t. Replacing a **t-in-k** tuple with $\mathbf{x}$ reduces the weight by $t - d$, which suggests $r = \left\lceil \frac{t}{t-d} \right\rceil$ such replacements.

Let a be the average column weight in the upper d rows. To achieve average row weight at most t, a must be such that the total weights in the upper and lower blocks are at most $d(t-r)$ and $t(k-d)$, respectively. This is guaranteed when both $a(k+1-r) \le d(t-r)$ and $(t-a)(k+1-r) \le t(k-d)$, or equivalently,

$$\frac{t(d-r+1)}{k+1-r} \le a \le \frac{d(t-r)}{k+1-r}. \tag{9}$$

To achieve average row weight at least $t-1$, a must be such that the total weights in the upper and lower blocks are at least $d(t-1-r)$ and $(k-d)(t-1)$, respectively. This is guaranteed when both $a(k+1-r) \ge d(t-1-r)$ and $(t-a)(k+1-r) \ge (k-d)(t-1)$, or equivalently,

$$\frac{d(t-1-r)}{k+1-r} \quad \le \quad a \quad \le \quad \frac{t(d-r+1)+k-d}{k+1-r}. \tag{10}$$

Finally, to ensure that each block of rows is tall enough to accommodate the 1's specified by the construction, it suffices that $0 \le a \le d$ and $0 \le t-a \le k-d$, which together are equivalent to $\max(0, d+t-k) \le a \le \min(d,t)$. By our assumptions, this reduces to $0 \le a \le d$.

We now show that these inequalities can all be simultaneously satisfied. In 9, the upper bound is at least the lower bound if and only if $r \ge \frac{t}{t-d}$, which holds for our choice of r. In 10, the upper bound is at least the lower bound if and only if $r \le \frac{t}{t-d} + \frac{k}{t-d}$, which holds since $\frac{k}{t-d} \ge 1$. Exchanging the upper/lower bound pairs in 9 and 10 results in two pairs of inequalities on a that are always satisfied. Finally, for $0 \le a \le d$, it suffices that at least one of the lower bounds is nonnegative, and at least one of the upper bounds is at most d. The lower bound in 9 is nonnegative if and only if $t \ge d+1$, and the upper bound is at most d if and only if $t \le k+1$, both of which always hold. The rest of the proof follows the same reasoning as in Proposition 21. □

Proposition 23. *Let $k \ge 3$, $1 \le t < k$, and $t < d < k$ be such that $d + t \le k$ and $(k,t,d) \equiv (1,0,0) \pmod 2$. Let $\mathbf{x}$ be a tuple of weight d. Then* $\mathrm{Pol}(\mathbf{t\text{-}\textit{in}\text{-}k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ *does not have 2-block-symmetric functions of all odd arities.*

Proof. The parities of k, t, and d prevent us from using the weights $t-1$ and $t+1$ in Case 2, which necessitates a different choice of weights and a slightly more complicated construction for Case 3. We take $L \ge 1$ copies of C_k^t in the even block of the tableau, and leave $Lk+1$ columns to be filled in the odd block, for a total arity of $2Lk+1$. The three weights we use are Lt, $Lt+1$, and $Lt+2$, with L determined later.

Case 1: weights Lt and $Lt+1$.

We take $L-1$ copies of C_k^t and one copy of C_k^{t+}, so that each row has weight Lt or $Lt+1$.

Case 2: weights Lt and $Lt+2$.

We take $L-1$ copies of C_k^t and one copy of C_k^{t+}, leaving t rows of weight $Lt+1$, and since t is even, these rows can be paired and values swapped so that each row has weight Lt or $Lt+2$. In particular, in columns $2, 4, \ldots, t$, we swap the values in the pairs of rows $(1,2), (3,4), \ldots, (t-1,t)$, respectively.

Case 3: weights $Lt+1$ and $Lt+2$.

Let $r = \left\lceil \frac{k-t}{d-t} \right\rceil$ and let a be the average column weight in the upper d rows. To achieve average row weight at least $Lt+1$, a must be such that the total weights in the upper and lower blocks are at least $d(Lt+1-r)$ and $(Lt+1)(k-d)$, respectively. This is guaranteed when both $a(Lk+1-r) \geq d(Lt+1-r)$ and $(t-a)(Lk+1-r) \geq (Lt+1)(k-d)$, or equivalently,

$$\frac{d(Lt+1-r)}{Lk+1-r} \leq a \leq \frac{t(Ld+1-r)-k+d}{Lk+1-r}. \tag{11}$$

To achieve average row weight at most $Lt+2$, a must be such that the total weights in the upper and lower blocks are at most $d(Lt+2-r)$ and $(k-d)(Lt+2)$, respectively. This is guaranteed when both $a(Lk+1-r) \leq d(Lt+2-r)$ and $(t-a)(Lk+1-r) \leq (k-d)(Lt+2)$, or equivalently,

$$\frac{t(Ld+1-r)-2(k+d)}{Lk+1-r} \leq a \leq \frac{d(Lt+2-r)}{Lk+1-r}. \tag{12}$$

Finally, to ensure that each block of rows is tall enough to accommodate the 1's specified in the construction, it suffices that $0 \leq a \leq d$ and $0 \leq t-a \leq k-d$, which together are equivalent to $\max(0, d+t-k) \leq a \leq \min(d,t)$. By our assumptions, this reduces to $0 \leq a \leq t$.

We now show that these inequalities can all be simultaneously satisfied. In 11, the upper bound is at least the lower bound if and only if $r \geq \frac{k-t}{d-t}$, which holds for our choice of r, and in 12, the upper bound is at least the lower bound if and only if $r \leq \frac{k-t}{d-t} + \frac{k}{d-t}$, which holds since $\frac{k}{d-t} \geq 1$. Exchanging the upper/lower bound pairs in 11 and 12 results in two pairs of inequalities on a that are always satisfied. Finally, for $0 \leq a \leq t$, it suffices that at least one of the lower bounds is nonnegative, and at least one of the upper bounds is at most t. The lower bound in 11 is nonnegative if and only if $L \geq \frac{r-1}{t}$, and the upper bound is at most t if and only if $L \geq -\frac{1}{t}$, so it suffices to take $L \geq \frac{r-1}{t}$. The rest of the proof follows the same reasoning as in Proposition 21. $\square$

Proposition 24. *Let $k \geq 3$, $1 < t < k$, and $1 \leq d < t$ be such that $t+d \leq k$ and $(k,t,d) \equiv (1,0,0) \pmod 2$. Let $\mathbf{x}$ be a tuple of weight d. Then $\mathrm{Pol}(\mathbf{t}\text{-}\boldsymbol{in}\text{-}\mathbf{k} \cup \{\mathbf{x}\}, \mathbf{NAE})$ does not have 2-block-symmetric functions of all odd arities.*

Proof. We place $L \geq 1$ copies of C_k^t in the odd block of our tableau, leaving $Lk-1$ columns to be filled in the even block for a total arity of $2Lk-1$. The three weights used are Lt, $Lt-1$, and $Lt-2$, with L determined later.
Case 1: weights Lt and $Lt-1$.
We take $L-1$ copies of C_k^t and one copy of C_k^{t-}, so that each row has weight Lt or $Lt-1$.
Case 2: weights Lt and $Lt-2$.
We take $L-1$ copies of C_k^t and one copy of C_k^{t-}, leaving t rows of weight $Lt-1$, and since t is even, these rows can be paired and values swapped so that each row has weight Lt or $Lt-2$. In particular, in columns $2,4,\ldots,t-2$, we swap the values in the pairs of rows $(1,2),(3,4),\ldots,(t-3,t-2)$, respectively. Finally, in column $k-1$, we swap the values in rows k and $t-1$.
Case 3: weights $Lt-1$ and $Lt-2$.

Let $r = \left\lceil \frac{k-t}{t-d} \right\rceil$ and let a be the average column weight in the upper d rows. To achieve average row weight at most $Lt-1$, a must be such that the total weights in the upper and lower blocks are at most $d(Lt-1-r)$ and $(Lt-1)(k-d)$, respectively. This is guaranteed when both $a(Lk-1-r) \leq d(Lt-1-r)$ and $(t-a)(Lk-1-r) \leq (Lt-1)(k-d)$, or equivalently,

$$\frac{t(Ld-1-r)+k-d}{Lk-1-r} \;\leq\; a \;\leq\; \frac{d(Lt-1-r)}{Lk-1-r}. \tag{13}$$

To achieve average row weight at least $Lt-2$, a must be such that the total weights in the upper and lower blocks are at least $d(Lt-2-r)$ and $(k-d)(Lt-2)$, respectively. This is guaranteed when both $a(Lk-1-r) \geq d(Lt-2-r)$ and $(t-a)(Lk-1-r) \geq (k-d)(Lt-2)$, or equivalently,

$$\frac{d(Lt-2-r)}{Lk-1-r} \;\leq\; a \;\leq\; \frac{t(Ld-1-r)+2(k-d)}{Lk-1-r}. \tag{14}$$

Finally, to ensure that each block of rows is tall enough to accommodate the 1's specified in the construction, it suffices that $0 \leq a \leq d$ and $0 \leq t-a \leq k-d$, which together are equivalent to $\max(0, d+t-k) \leq a \leq \min(d,t)$. By our assumptions, this reduces to $0 \leq a \leq d$.

We now show that these inequalities can all be simultaneously satisfied. In 13, the upper bound is at least the lower bound if and only if $r \geq \frac{k-t}{t-d}$, which holds for our choice of r, and in 14, the upper bound is at least the lower bound if and only if $r \leq \frac{k-t}{t-d} + \frac{k}{t-d}$, which holds since $\frac{k}{t-d} \geq 1$. Exchanging the upper/lower bound pairs in 13 and 14 results in two pairs of inequalities on a that are always satisfied. Finally, for $0 \leq a \leq d$, it suffices that at least one of the lower bounds is nonnegative, and at least one of the upper bounds is at most d. The lower bound in 14 is nonnegative if and only if $L \geq \frac{r+2}{t}$, and the upper bound in 13 is at most d if and only if $t \leq k$, so it suffices to take $L \geq \frac{r+2}{t}$. The rest of the proof follows the same reasoning as in Proposition 21. □

5 Removing tuples

In this section we prove Theorem 13 and show how it implies Theorem 14.

Schaefer's dichotomy theorem (Theorem 4) allows us to obtain a simple description of all $\mathbf{T}$ with CSP($\mathbf{T}$) tractable and **t-in-k** $\to \mathbf{T}$.

Proposition 25. *Let $k \geq 3$, $1 \leq t < k$, and suppose that* **t-*in*-k** $\to \mathbf{T}$*. Then* CSP($\mathbf{T}$) *is tractable if and only if*

1. $0^k \in \mathbf{T}$ *or* $1^k \in \mathbf{T}$*, or*
2. *t is odd, k is even, and* $\mathbf{T} =$ **odd-*in*-k***.*

Observe that Proposition 25 in particular implies Proposition 7, NP-hardness of CSP(**t-in-k**).

Proof. In Case 1, Pol($\mathbf{T}$) contains a constant function so CSP($\mathbf{T}$) is tractable by Theorem 4, and in Case 2, tractability is given by Proposition 17.

We now turn to hardness. For the rest of the proof, assume that neither (1) nor (2) of the proposition statement applies.

Suppose that **t-in-k** $\to \mathbf{T}$ by the function $\phi : \{0,1\} \to \{0,1\}$. If ϕ is constant, then either $\mathbf{T} = \{0^k\}$ or $\mathbf{T} = \{1^k\}$ and we are in case (1), a contradiction. If $\phi(x) = 1-x$, note that ϕ(**t-in-k**) satisfies conditions (1) and (2) precisely when **t-in-k** does, so it suffices to consider only the case where ϕ is the identity and **t-in-k** $\subseteq \mathbf{T}$.

We show that Pol($\mathbf{T}$) contains none of the functions AND$_2$, OR$_2$, MAJ$_3$, and XOR$_3$ from Theorem 4. To accomplish this, we assume that one of these functions f is present in Pol($\mathbf{T}$), and then show that repeated application of f to a certain set of tuples leads to (1) or (2) of the proposition, a contradiction. Recall that we denote by $f(R)$ the image of the relation R under f. We make seven claims below, which together exclude the four functions as polymorphisms: case (i) covers AND_2, case (ii) covers OR_2, (overlapping) cases (iii) and (iv) cover MAJ_3, and cases (v), (vi), and (vii) cover XOR_3. Case (vii) contradicts (2) of the proposition; all others contradict (1).[4]

We give more details for case (i) for illustration. Taking AND_2 of the two tuples, we get $\mathrm{AND}_2(1^t0^{k-t}, 01^t0^{k-t-1}) = 01^{t-1}0^{k-t}$, a tuple of weight $t-1$. By symmetry, we obtain all tuples of weight $t-1$, which is the statement of case (i). Continuing this way, we obtain all tuples of weight $t-2$, $t-3$, etc. until we eventually obtain 0^k, which gives a contradiction as we assume that (1) of the proposition does not apply, so $0^k \notin \mathbf{T}$.

i $(\mathbf{t-1})$**-in-k** $\subseteq \mathrm{AND}_2($**t-in-k**$)$

tuples: 1^t0^{k-t} and 01^t0^{k-t-1}

eventual output: 0^k

ii $(\mathbf{t+1})$**-in-k** $\subseteq \mathrm{OR}_2($**t-in-k**$)$

tuples: 1^t0^{k-t} and 01^t0^{k-t-1}

eventual output: 1^k

iii if $t \geq 2$ then $(\mathbf{t+1})$**-in-k** $\subseteq \mathrm{MAJ}_3($**t-in-k**$)$

tuples: $1^{t-2}0^{k-t-1}110, 1^{t-2}0^{k-t-1}101$, and $1^{t-2}0^{k-t-1}011$

eventual output: 1^k

iv if $t \leq k-2$ then $(\mathbf{t-1})$**-in-k** $\subseteq \mathrm{MAJ}_3($**t-in-k**$)$

tuples: $1^{t-1}0^{k-t-2}100, 1^{t-1}0^{k-t-2}010$, and $1^{t-1}0^{k-t-2}001$

eventual output: 0^k

v if t is even, then $(\mathbf{t-2})$**-in-k** $\subseteq \mathrm{XOR}_3($**t-in-k**$)$

tuples: $1^{t-2}0^{k-t-1}110, 1^{t-2}0^{k-t-1}101$, and $1^{t-2}0^{k-t-1}011$

eventual output: 0^k

vi if t and k are odd, then $(\mathbf{t+2})$**-in-k** $\subseteq \mathrm{XOR}_3($**t-in-k**$)$

tuples: $1^{t-1}0^{k-t-2}100, 1^{t-1}0^{k-t-2}010$, and $1^{t-1}0^{k-t-2}001$

eventual output: 1^k

vii if t is odd and k is even,
then $(\mathbf{t+2})$**-in-k** $\subseteq \mathrm{XOR}_3($**t-in-k**$)$ if $t < k-2$,
and $(\mathbf{t-2})$**-in-k** $\subseteq \mathrm{XOR}_3($**t-in-k**$)$ if $t > 2$.

tuples:

$1^{t-1}0^{k-t-2}100, 1^{t-1}0^{k-t-2}010$, and $1^{t-1}0^{k-t-2}001$

[4]Another way of establishing this result for XOR_3 is via linear algebra (being closed under XOR_3 is the same as being an affine subspace) and for MAJ_3 from the fact that relations closed under MAJ_3 are determined by their binary projections.

$1^{t-2}0^{k-t-1}110, 1^{t-2}0^{k-t-1}101$, and $1^{t-2}0^{k-t-1}011$

eventual output: all odd weight tuples

$\square$

With Proposition 25 in hand, we can prove Theorems 13 and 14.

Theorem (Theorem 13 restated)**.** Let $k \geq 3$ and let $\mathbf{T} \subseteq \{0,1\}^k$ be a relation such that $\textbf{t-in-k} \to \mathbf{T}$ and $\mathrm{CSP}(\mathbf{T})$ is NP-hard. Then $\mathrm{PCSP}(\textbf{t-in-k}, \mathbf{T})$ is tractable if and only if $\mathbf{T} = \mathbf{NAE}$.

Proof. By Proposition 16, we can assume that $\mathbf{T}$ is symmetric. If $\mathbf{T} = \mathbf{NAE}$ then $\mathrm{PCSP}(\textbf{t-in-k}, \mathbf{T})$ is tractable by Proposition 8. Otherwise, we show that $\mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$ does not contain any of the tractable polymorphism families identified in the symmetric Boolean PCSP dichotomy (Theorem 5), and therefore $\mathrm{PCSP}(\textbf{t-in-k}, \mathbf{T})$ is NP-hard.

The families we need to rule out are constants, OR, AND, XOR, AT, and THR_q for $q \in \mathbb{Q}$, as well as their negations. We deal first with the non-negated families. Since $\mathrm{CSP}(\mathbf{T})$ is NP-hard, by Proposition 25, we have $0^k \notin \mathbf{T}$ and $1^k \notin \mathbf{T}$. Hence, $\mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$ does not contain constants.

Let C_k^t be the $k \times k$ matrix containing the k cyclic shifts of the column 1^t0^{k-t}. Then C_k^t prevents the polymorphism families OR, AND, XOR (if k is odd), and THR_q for all $q \neq \frac{t}{k}$. The case $q = \frac{t}{k}$ is ruled out by [20, Fact B.3], and AT is ruled out by [10, Claim 4.6]. Since $\mathrm{CSP}(\mathbf{T})$ is NP-hard, by Proposition 25, it remains to show that for even k, $\mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$ excludes XOR when t is even, and likewise when t is odd and $\mathbf{T}$ is missing a tuple of odd weight.

Let k and t be even. Applying XOR_k to the matrix C_k^t returns the tuple 0^k, so applying XOR_{k-1} to the first $k-1$ columns of C_k^t returns the last column $1^{t-1}0^{k-t}1$. We can "fill in" the 0's in the output by swapping $0/1$ pairs of values in the input matrix. In particular, in the columns $k-1, k-3, \ldots, t+1$, we swap the entries in the pairs of rows $(k-1, k-2), (k-3, k-4), \ldots, (t+1, t)$, respectively. The resulting $k \times (k-1)$ matrix M then satisfies $\mathrm{XOR}_{k-1}(M) = 1^k$ and the arity $k-1$ is odd as required. An example with swapped values in bold is illustrated in Figure (5a).

$$\mathrm{XOR}_7 \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & \mathbf{1} & 0 & 0 \\ 0 & 1 & 1 & 1 & \mathbf{0} & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & \mathbf{1} \\ 0 & 0 & 0 & 1 & 1 & 1 & \mathbf{0} \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{matrix}$$

(a) $t = 4$ and $k = 8$.

$$\mathrm{XOR}_5 \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{matrix}$$

(b) $t = 3$, $k = 6$, and $d = 5$.

Figure 5: XOR.

Now let k be even, t be odd, and suppose that $\mathbf{T}$ does not contain the tuple $\mathbf{x} = 1^d0^{k-d}$ of odd weight d. By Observation 20, we can assume without loss of generality that $t < d$. Then

XOR_d applied to the matrix C_d^t padded with $k-d$ rows of 0's returns $\mathbf{x}$. An illustration is given in Figure (5b). Therefore $\mathrm{XOR} \not\subseteq \mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$.

Negations: Let F be a family of functions. We reduce the task of showing $\overline{F} \not\subseteq \mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$ to the already completed task of showing $F \not\subseteq \mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$. Let $\mathbf{x} \in \{0,1\}^k \setminus \mathbf{T}$, let $f \in F$ be a function of arity m, and let M be a $k \times m$ matrix of inputs to f whose columns are **t-in-k** tuples. We established $F \not\subseteq \mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$ by finding f and M with $f(M) = \mathbf{x}$, and in the remaining cases we must find $\overline{f} \in \overline{F}$ and M such that $\overline{f}(M) = \mathbf{x}$. But since $\overline{f}(M) = \mathbf{x} \Leftrightarrow f(M) = \overline{\mathbf{x}}$, it suffices to find $f \in F$ such that $f(M) = \overline{\mathbf{x}}$, where $\overline{\mathbf{x}} = (1-x_1, \ldots, 1-x_k)$ if $\mathbf{x} = (x_1, \ldots, x_k)$.

The families $\overline{\mathrm{AND}}$, $\overline{\mathrm{OR}}$, $\overline{\mathrm{XOR}}$ (except when k is even and t is odd), and $\overline{\mathrm{THR}_q}$ for all $q \neq \frac{t}{k}$ are excluded from $\mathrm{Pol}(\textbf{t-in-k}, \mathbf{T})$ in the same way as AND, OR, XOR, and THR_q with the same matrices serving as counterexamples. In detail, the matrix C_k^t, which contains k cyclic shifts of the column $1^t 0^{k-t}$, prevents the polymorphism families $\overline{\mathrm{AND}}$, $\overline{\mathrm{OR}}$, $\overline{\mathrm{XOR}}$ (if k is odd), and $\overline{\mathrm{THR}_q}$ (if $q \neq \frac{t}{k}$). The case $\overline{\mathrm{XOR}}$ with k even, t even is ruled out the same way as before, illustrated in Figure (5a).

To see that $\overline{\mathrm{AT}}$ and $\overline{\mathrm{THR}_{\frac{t}{k}}}$ (with $q = \frac{t}{k}$) are also excluded, let $\mathbf{x} \notin \mathbf{T}$ be a tuple of weight $d \neq t$. Then the tuple $\overline{\mathbf{x}}$ of weight $k-d$ can be returned by an AT function [10, Claim 4.6] and a $\mathrm{THR}_{\frac{t}{k}}$ function [20, Fact B.3]. If $k-d = t$, then the AT and $\mathrm{THR}_{\frac{t}{k}}$ functions of arity 1 output $\overline{\mathbf{x}}$ on input $\overline{\mathbf{x}}$.

Finally, when k is even, t is odd, and $\mathbf{T}$ does not contain the tuple $\mathbf{x}$ of odd weight d, the XOR argument above (illustrated in Figure 5b) applies since $\overline{\mathbf{x}}$ also has odd weight $k-d$. Again, if $k-d=t$, then the XOR function of arity 1 outputs $\overline{\mathbf{x}}$ on input $\overline{\mathbf{x}}$. ☐

Theorem (Theorem 14 restated)**.** Let $k \geq 3$ and $\emptyset \neq S \subseteq (\textbf{t-in-k})^c \cap \mathbf{NAE}$. If t is odd, k is even, and S contains tuples of only even weight, then $\mathrm{PCSP}(\textbf{t-in-k}, \mathbf{NAE} \setminus \mathbf{S})$ is tractable. Otherwise, $\mathrm{PCSP}(\textbf{t-in-k}, \mathbf{NAE} \setminus \mathbf{S})$ is NP-hard.

Proof. The tractability in the first statement of the theorem is proved in Section 3. Otherwise, t is even, or k is odd, or S contains a tuple of odd weight. Take $\mathbf{T} = \mathbf{NAE} \setminus \mathbf{S}$. Observe that case (1) of Proposition 25 does not apply as neither 0^k nor 1^k is part of the template. Moreover, case (2) of Proposition 25 does not apply either: If t is odd and k is even then S contains a tuple of odd weight and hence $\mathbf{NAE} \setminus \mathbf{S}$ cannot have all odd weight tuples. Thus, by Proposition 25, $\mathrm{CSP}(\mathbf{T})$ is NP-hard. Then, by Theorem 13, $\mathrm{PCSP}(\textbf{t-in-k}, \mathbf{T}) = \mathrm{PCSP}(\textbf{t-in-k}, \mathbf{NAE} \setminus \mathbf{S})$ is NP-hard. ☐

Acknowledgements

We would like to thank the anonymous referees of both the conference [13] and this full version of the paper. We also thank Kristina Asimi and Libor Barto for useful discussions regarding the content of this and their paper [1].

References

[1] Kristina Asimi and Libor Barto. Finitely tractable promise constraint satisfaction problems. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS'21)*, volume 202 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.11.

[2] Albert Atserias and Víctor Dalmau. Promise Constraint Satisfaction and Width. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA'22)*, pages 1129–1153, 2022. arXiv:2107.05886, doi:10.1137/1.9781611977073.48.

[3] Per Austrin, Amey Bhangale, and Aditya Potukuchi. Improved inapproximability of rainbow coloring. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1479–1495, 2020. arXiv:1810.02784, doi:10.1137/1.9781611975994.90.

[4] Per Austrin, Venkatesan Guruswami, and Johan Håstad. (2+ϵ)-Sat is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017. doi:10.1137/15M1006507.

[5] Libor Barto, Diego Battistelli, and Kevin M. Berg. Symmetric Promise Constraint Satisfaction Problems: Beyond the Boolean Case. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS'21)*, volume 187 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. arXiv:2010.04623, doi:10.4230/LIPIcs.STACS.2021.10.

[6] Libor Barto, Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *Journal of the ACM*, 68(4):28:1–28:66, 2021. arXiv:1811.00970, doi:10.1145/3457606.

[7] Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei Krokhin and Stanislav Živný, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.1.

[8] Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223(1):363–398, Feb 2018. arXiv:1510.04521, doi:10.1007/s11856-017-1621-9.

[9] Joshua Brakensiek and Venkatesan Guruswami. An Algorithmic Blend of LPs and Ring Equations for Promise CSPs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 436–455. SIAM, 2019. arXiv:1807.05194, doi:10.1137/1.9781611975482.28.

[10] Joshua Brakensiek and Venkatesan Guruswami. Promise Constraint Satisfaction: Algebraic Structure and a Symmetric Boolean Dichotomy. *SIAM J. Comput.*, 50(6):1663–1700, 2021. doi:10.1137/19M128212X.

[11] Joshua Brakensiek, Venkatesan Guruswami, Marcin Wrochna, and Stanislav Živný. The power of the combined basic LP and affine relaxation for promise CSPs. *SIAM J. Comput.*, 49:1232–1248, 2020. arXiv:1907.04383, doi:10.1137/20M1312745.

[12] Alex Brandts, Marcin Wrochna, and Stanislav Živný. The complexity of promise SAT on non-Boolean domains. *ACM Trans. Comput. Theory*, 13(4):26:1–26:20, 2021. arXiv:1911.09065, doi:10.1145/3470867.

[13] Alex Brandts and Stanislav Živný. Beyond PCSP(1-in-3,NAE). In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP'21)*, volume 198 of *LIPIcs*, pages 121:1–121:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. arXiv:2104.12800, doi:10.4230/LIPIcs.ICALP.2021.121.

[14] Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005. doi:10.1137/S0097539700376676.

[15] Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, 2017. arXiv:1703.03021, doi:10.1109/FOCS.2017.37.

[16] Lorenzo Ciardo and Stanislav Živný. CLAP: A New Algorithm for Promise CSPs. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA'22)*, pages 1057–1068, 2022. arXiv:2107.05018, doi:10.1137/1.9781611977073.46.

[17] Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3):843–873, 2009. doi:10.1137/07068062X.

[18] Irit Dinur, Oded Regev, and Clifford Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, September 2005. doi:10.1007/s00493-005-0032-4.

[19] Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

[20] Miron Ficak, Marcin Kozik, Miroslav Olšák, and Szymon Stankiewicz. Dichotomy for Symmetric Boolean PCSPs. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP'19)*, volume 132 of *LIPIcs*, pages 57:1–57:12, 2019. arXiv:1904.12424, doi:10.4230/LIPIcs.ICALP.2019.57.

[21] M. R. Garey and David S. Johnson. The complexity of near-optimal graph coloring. *J. ACM*, 23(1):43–49, 1976. doi:10.1145/321921.321926.

[22] Venkatesan Guruswami and Sai Sandeep. d-To-1 Hardness of Coloring 3-Colorable Graphs with O(1) Colors. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, volume 168 of *LIPIcs*, pages 62:1–62:12, 2020. doi:10.4230/LIPIcs.ICALP.2020.62.

[23] Pavol Hell and Jaroslav Nešetřil. On the Complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

[24] Peter Jeavons, David A. Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997. doi:10.1145/263867.263489.

[25] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, 1972. URL: http://www.cs.berkeley.edu/%7Eluca/cs172/karp.pdf.

[26] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

[27] Andrei Krokhin and Jakub Opršal. The complexity of 3-colouring H-colourable graphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS'19)*, pages 1227–1239, 2019. arXiv:1904.03214, doi:10.1109/FOCS.2019.00076.

[28] Thomas Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth Annual ACM Symposium on the Theory of Computing (STOC '78)*, pages 216–226, 1978. doi:10.1145/800133.804350.

[29] Marcin Wrochna and Stanislav Živný. Improved hardness for H-colourings of G-colourable graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1426–1435, 2020. arXiv:1907.00872, doi:10.1137/1.9781611975994.86.

[30] Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, August 2020. arXiv:1704.01914, doi:10.1145/3402029.