The complexity of counting locally maximal satisfying assignments of Boolean CSPs*

Leslie Ann Goldberg[†]   Mark Jerrum[‡]

April 7, 2016

Abstract

We investigate the computational complexity of the problem of counting the locally maximal satisfying assignments of a Constraint Satisfaction Problem (CSP) over the Boolean domain $\{0, 1\}$. A satisfying assignment is *locally maximal* if any new assignment which is obtained from it by changing a 0 to a 1 is unsatisfying. For each constraint language Γ, #LocalMaxCSP(Γ) denotes the problem of counting the locally maximal satisfying assignments, given an input CSP with constraints in Γ. We give a complexity dichotomy for the problem of *exactly* counting the locally maximal satisfying assignments and a complexity trichotomy for the problem of *approximately* counting them. Relative to the problem #CSP(Γ), which is the problem of counting *all* satisfying assignments, the locally maximal version can sometimes be easier but never harder. This finding contrasts with the recent discovery that approximately counting locally maximal independent sets in a bipartite graph is harder (under the usual complexity-theoretic assumptions) than counting all independent sets.

Keywords. Constraint satisfaction problem; computational complexity of counting problems; approximate computation.

1 Introduction

A Boolean Constraint Satisfaction Problem (CSP) is a generalised satisfiability problem. An instance of a Boolean CSP is a set of variables together with a collection of constraints that enforce certain relationships between the variables. These constraints are chosen from an agreed finite set ("language") Γ of relations of various arities on the Boolean domain $\{0, 1\}$. The study of the computational complexity of Boolean CSPs has a long history, starting with Schaefer, who described the complexity of the basic decision problem: is a given Boolean CSP instance satisfiable? The computational complexity of the satisfiability problem depends, of course, on the constraint language Γ, becoming potentially harder as Γ becomes larger and more expressive. Schaefer showed [15] that, depending on Γ, the satisfiability problem is

1

either polynomial-time solvable or NP-complete, and he provided a precise characterisation of this dichotomy.

The problem #CSP(Γ) is the problem of determining the number of satisfying assignments of a CSP instance with constraint language Γ. A dichotomy for this counting problem was given by Creignou and Hermann [1].

Theorem 1. *(Creignou, Hermann [1]) Let Γ be a constraint language with domain $\{0, 1\}$. The problem #CSP(Γ) is in FP if every relation in Γ is affine. Otherwise, #CSP(Γ) is #P-complete.*

A relation is *affine* if it is expressible as the set of solutions to a system of linear equations over the two-element field $\mathbb{F}_2$. The constraint language Γ is said to be affine if and only if every constraint in Γ is affine. Thus, Theorem 1 shows (assuming FP $\neq$ #P) that #CSP(Γ) is tractable if and only if the set of satisfying assignments can be expressed as the set of solutions to a system of linear equations.

Since most constraint languages Γ lead to intractable counting problems, it is natural to consider the complexity of *approximately* counting the satisfying assignments of a CSP. Dyer, Goldberg, Greenhill and Jerrum [7] used approximation-preserving reductions (AP-reductions) between counting problems to explore the complexity of approximately computing solutions. They identified three equivalence classes of interreducible counting problems within #P: (i) problems that have a polynomial-time approximation algorithm or "FPRAS", (ii) problems that are equivalent to #BIS under AP-reductions, and (iii) problems that are equivalent to #SAT under AP-reductions. Here, #BIS is the problem of counting independent sets in a bipartite graph and #SAT is the problem of counting the satisfying assignments of a Boolean formula in CNF. Dyer, Goldberg and Jerrum [8] show that all Boolean counting CSPs can be classified using these classes.

Theorem 2. *([8, Theorem 3]) Let Γ be a constraint language with domain $\{0, 1\}$. If every relation in Γ is affine then #CSP(Γ) is in FP. Otherwise if every relation in Γ is in IM_2 then #CSP(Γ) $\equiv_{\mathsf{AP}}$ #BIS. Otherwise #CSP(Γ) $\equiv_{\mathsf{AP}}$ #SAT.*

In the statement of Theorem 2, $\equiv_{\mathsf{AP}}$ is the equivalence relation "interreducible via approximation-preserving reductions". In order to define IM_2, we must first define the binary implication relation Implies $= \{(0, 0), (0, 1), (1, 1)\}$. Then IM_2 is the set of relations that can be expressed using conjunctions of these implications, together with unary constraints. The precise definition of IM_2 is given in Section 2, along with precise definitions of the other concepts that appear in this introduction.

There are many other questions that one can ask about Boolean CSPs aside from deciding satisfiability and counting the satisfying assignments. Here, we study the complexity of counting and approximately counting the number of locally maximal satisfying assignments of a CSP instance. A satisfying assignment is *locally maximal* if any new assignment which is obtained from it by changing a single 0 to a 1 is unsatisfying. So local maximality is with respect to the set of 1's in the satisfying assignment. Also, it is with respect to local changes — changing a single 0 to a 1. Other notions of maximality are discussed in Section 4.

Goldberg, Gysel and Lapinskas [9] show (assuming that #BIS is not equivalent to #SAT under AP reductions) that counting locally maximal structures can be harder than counting all structures. In particular, Theorem 1 of [9] shows that counting the locally maximal independent sets in a bipartite graph is equivalent to #SAT under AP-reductions. Obviously, counting *all* independent sets in a bipartite graph is exactly the problem #BIS, which

is presumed to be easier. Thus, Goldberg, Gysel and Lapinskas have found an example of a (restricted) Boolean counting CSP (namely, #BIS) where approximately counting locally maximal satisfying assignments is apparently harder than approximately counting all satisfying assignments. However, #BIS isn't exactly a Boolean counting CSP — rather, it is a Boolean counting CSP with a restriction (bipartiteness) on the problem instance. This prompted us to investigate the complexity of approximating the number of satisfying assignments of unrestricted Boolean CSPs. In particular, we study #LocalMaxCSP(Γ), the problem of counting locally maximal satisfying assignments of an instance of a Boolean CSP with constraint language Γ.

Given the phenomenon displayed by #BIS, one might expect to find constraint languages Γ that exhibit a jump upwards in computational complexity when passing from #CSP(Γ) to #LocalMaxCSP(Γ), but we determine that this does not in fact occur. The reverse may occur: counting locally maximal vertex covers in a graph is trivial (there is just one), but counting all vertex covers is equivalent under AP-reducibility to #SAT. It turns out that this trivial phenomenon, which occurs when the property in question is monotone increasing, is essentially the only difference between #CSP(Γ) and #LocalMaxCSP(Γ).

Our first result (Theorem 3) presents a dichotomy for the complexity of exactly solving #LocalMaxCSP(Γ) for all Boolean constraint languages Γ. In most cases, #LocalMaxCSP(Γ) is equivalent in complexity to #CSP(Γ). However, if Γ is *essentially monotone* (the proper generalisation of the vertex cover property) then #LocalMaxCSP(Γ) is in FP. Our second result (Theorem 4) presents a trichotomy for the complexity of *approximately* solving #LocalMaxCSP(Γ). Once again, in most cases, #LocalMaxCSP(Γ) is equivalent with respect to AP-reductions to #CSP(Γ). The only exceptional case is the one that we have already seen — if Γ is *essentially monotone* then #LocalMaxCSP(Γ) is in FP.

The result leaves us with a paradox. There is a very direct reduction from #BIS to #CSP({Implies}) that is "parsimonious", i.e., preserves the number of solutions. So #BIS is AP-reducible to #CSP({Implies}) (which can also be seen from Theorem 2). How can it be, then, that the complexity of approximately counting locally maximal independent sets in bipartite graphs jumps upwards from the complexity of approximately counting all independent sets, whereas Theorem 4 tells us that #LocalMaxCSP({Implies}) remains AP-equivalent to #CSP({Implies})?

The resolution of the paradox is as follows. Suppose $G = (U, V, E)$ is an instance of #BIS, i.e., a graph with bipartition $U \cup V$ and edge set $E \subseteq U \times V$. The parsimonious reduction from #BIS to #CSP({Implies}) simply interprets the vertices $U \cup V$ of instance G as Boolean variables, and each edge $(u, v) \in E$ as a constraint Implies(u, v). Then there is an obvious bijection between independent sets in G and satisfying assignments of the constructed instance of #CSP({Implies}), but it involves interpreting 0 and 1 in different ways on the opposite sides of the bipartition: on U, 1 means "in the independent set" while on V, 1 means "out of the independent set". Of course, local maximality is not preserved by this change in interpretation, as it presumes a particular ordering on 0 and 1. Local maximality is sensitive to the precise encoding of solutions, and parsimonious reductions are no longer enough to capture complexity equivalences.

Thus, we are left with the situation that approximately counting locally maximal independent sets in bipartite graphs is apparently more difficult than approximately counting all independent sets, but within the realm of Boolean constraint satisfaction, this phenomenon does not occur. Relative to #CSP(Γ), the problem #LocalMaxCSP(Γ) can sometimes be easier but never harder.

2 Notation and Preliminaries

2.1 Locally maximal Constraint Satisfaction Problems

A Boolean constraint language Γ is a set of relations on $\{0, 1\}$. Once we have fixed the constraint language Γ, an *instance I* of the CSP consists of a set V of *variables* and a set C of *constraints*. Each constraint has a *scope*, which is a tuple of variables and a relation from Γ of the same arity, which constrains the variables in the scope. An *assignment σ* is a function from V to the Boolean domain $\{0, 1\}$. The assignment σ is *satisfying* if the scope of every constraint is mapped to a tuple that is in the corresponding relation. Given an assignment σ, a variable v, and a Boolean value s, let $\sigma_{[v \to s]}$ be the assignment defined as follows: $\sigma_{[v \to s]}(v) = s$ and for all $w \in V \setminus \{v\}$, $\sigma_{[v \to s]}(w) = \sigma(w)$. (Thus, $\sigma_{[v \to s]}$ agrees with σ, except possibly at variable v, which it assigned Boolean value s.) We say that a satisfying assignment σ is *maximal* for v if either

- $\sigma(v) = 1$, or

- $\sigma_{[v \to 1]}$ is unsatisfying.

We say that the satisfying assignment σ is *locally maximal* if it is maximal for every variable $v \in V$.

For example, consider the ternary relation $R = \{(0, 0, 0), (0, 0, 1), (1, 0, 0), (0, 1, 1), (1, 1, 1)\}$ which excludes the three tuples $(0, 1, 0)$, $(1, 1, 0)$ and $(1, 0, 1)$. Let Γ be the size-one constraint language $\Gamma = \{R\}$. Let I be the instance with variable set $V = \{v_1, v_2, v_3, v_4, v_5\}$ and constraint set $C = \{R(v_1, v_2, v_3), R(v_3, v_4, v_5)\}$. Consider the following assignments.

σ	$\sigma(v_1)$	$\sigma(v_2)$	$\sigma(v_3)$	$\sigma(v_4)$	$\sigma(v_5)$
σ_1	0	0	1	1	0
σ_2	0	0	1	1	1
σ_3	1	1	1	0	0

The assignment σ_1 is not satisfying because the constraint $R(v_3, v_4, v_5)$ is not satisfied since $(1, 1, 0)$ is not in R. Assignments σ_2 and σ_3 are satisfying. Assignment σ_2 is not maximal for v_2 since $\sigma_2(v_2) = 0$ and $(\sigma_2)_{[v_2 \to 1]}$ is satisfying. However, σ_2 is maximal for every other variable v_i. Assignment σ_3 is locally maximal.

Given an instance I of a CSP with constraint language Γ, the *decision problem* $\mathsf{CSP}(\Gamma)$ is to determine whether any assignment satisfies I. The *counting problem* $\#\mathsf{CSP}(\Gamma)$ is to determine the *number* of satisfying assignments of I. Finally, the *locally maximal counting problem* $\#\mathsf{LocalMaxCSP}(\Gamma)$ is to determine the number of locally maximal satisfying assignments of I.

2.2 Boolean Relations

A Boolean relation R is said to be *0-valid* if the all-zero tuple is in R and it is said to be *1-valid* if the all-one tuple is in R. For every positive integer k and every $i \in \{1, \ldots, k\}$, let $e_{i,k}$ be the k-ary tuple with a one in position i and zeroes in the other positions. We say that a k-ary relation R is *monotone* if, for every tuple $(s_1, \ldots, s_k) \in R$ and every i, the tuple $(s_1, \ldots, s_k) \vee e_{i,k}$ is also in R, where $\vee$ is the *or* operator, applied position-wise.

The set of *zero positions* of R, written $\mathcal{Z}(R)$, is $\{i \in \{1, \ldots, k\} \mid \forall (s_1, \ldots, s_k) \in R, s_i = 0\}$. Of course, $\mathcal{Z}(R)$ may be the empty set. We use $\mathcal{N}(R)$ to denote the set containing all other

positions, so $\mathcal{N}(R) = [k] \setminus \mathcal{Z}(R)$. We use R^* to denote the relation induced on positions in $\mathcal{N}(R)$. We say that R is *essentially monotone* if R^* is monotone.

For example, the relation $R = \{(0,0,1),(0,1,1),(1,1,1)\}$ is not monotone because the tuple $(0,0,1)$ has a zero in the first position but $(0,0,1) \lor (1,0,0) = (1,0,1)$ which is not in R. The relation $R = \{(0,0,1,1),(0,1,1,1)\}$ has $\mathcal{Z}(R) = \{1\}$ because all tuples in R have a zero in the first position, and $\mathcal{N}(R) = \{2,3,4\}$. The relation R^* induced on positions in $\mathcal{N}(R)$ is $R^* = \{(0,1,1),(1,1,1)\}$. R^* is monotone, so R is essentially monotone.

The binary implication relation $\mathsf{Implies}$ is defined as $\mathsf{Implies} = \{(0,0),(0,1),(1,1)\}$. We will also consider two unary relations U_0 and U_1. U_0 is defined by $\mathsf{U}_0 = \{(0)\}$. The constraint $\mathsf{U}_0(x)$ is often called "pinning x to 0". Similarly, U_1 is defined by $\mathsf{U}_1 = \{(1)\}$ and the constraint $\mathsf{U}_1(x)$ is called "pinning x to 1".

A Boolean *co-clone* [3] is a set of Boolean relations containing the equality relation $\{(0,0),(1,1)\}$ and closed under certain operations. For completeness, the operations are finite Cartesian products, projections, and identifications of variables, but it will not be necessary to define these here. We will not require any information about co-clones, apart from the fact that the set of all affine relations is a co-clone, and so is a certain set IM_2 which we we have already discussed, and will define below.

Suppose that Γ is a co-clone. A subset B of Γ is said to be a "plain basis" for Γ [3, Definition 1] if and only if it is the case that every constraint C over Γ is logically equivalent to a conjunction of constraints over B using the same variables as C.

Creignou et al. [3] have shown that $\{\mathsf{Implies}, \mathsf{U}_0, \mathsf{U}_1\}$ is a plain basis for the set IM_2. In fact, IM_2 can just be defined this way. A relation $R(x_1,\ldots,x_k)$ is in IM_2 if and only if it is logically equivalent to a conjunction of constraints over $\{\mathsf{Implies}, \mathsf{U}_0, \mathsf{U}_1\}$ using the variables $x_1,\ldots,x_k$. For example, consider the relation $R = \{(0,0,0,1),(0,1,1,1)\}$. The relation R is in IM_2 because the constraint $R(w,x,y,z)$ is logically equivalent to the conjunction of constraints $\mathsf{U}_0(w)$, $\mathsf{Implies}(x,y)$, $\mathsf{Implies}(y,x)$ and $\mathsf{U}_1(z)$. Note that the conjunction does not use any variables other than w, x, y and z, and this is important.

Let $\mathbb{L}$ denote the set of relations corresponding to linear equations over the two-element field $\mathbb{F}_2$. For example, the equality relation $\{(0,0),(1,1)\}$ is in $\mathbb{L}$ because it corresponds to the equation $x_1 \oplus x_2 = 0$. Also, the relation $\{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}$ is in $\mathbb{L}$ because it corresponds to the equation $x_1 \oplus x_2 \oplus x_3 = 1$. Let $\mathbb{L}_k$ denote the set containing all relations in $\mathbb{L}$ of arity at most k. We have already said that a relation is *affine* if it is expressible as the set of solutions to a system of linear equations over $\mathbb{F}_2$ and that a constraint language Γ is affine iff every constraint in Γ is affine. Creignou et al. [3] note that $\mathbb{L}$ is a plain basis for the set of affine relations. Thus, if Γ is affine, and every constraint in Γ has arity at most k, then every constraint C over Γ is logically equivalent to a conjunction of constraints over $\mathbb{L}_k$ using the same variables as C.

2.3 The complexity of approximate counting

We now recall the necessary complexity-theoretic background from [7]. A *randomised approximation scheme* is an algorithm for approximately computing the value of a function $f : \Sigma^* \to \mathbb{N}$. The approximation scheme has a parameter $\varepsilon > 0$ which specifies the error tolerance. A *randomised approximation scheme* for f is a randomised algorithm that takes as input an instance $x \in \Sigma^*$ (e.g., an encoding of a CSP instance) and an error tolerance $\varepsilon > 0$, and outputs an integer z (a random variable on the "coin tosses" made by the algorithm) such

that, for every instance x,

$$\Pr\left[e^{-\epsilon}f(x) \le z \le e^{\epsilon}f(x)\right] \ge \frac{3}{4}. \tag{1}$$

The randomised approximation scheme is said to be a *fully polynomial randomised approximation scheme*, or *FPRAS*, if it runs in time bounded by a polynomial in $|x|$ and ϵ^{-1}. (See Mitzenmacher and Upfal [13, Definition 10.2].)

Suppose that f and g are functions from Σ^* to $\mathbb{N}$. An "approximation-preserving reduction" (AP-reduction) from f to g is a randomised algorithm $\mathcal{A}$ for computing f using an oracle for g.[1] The algorithm $\mathcal{A}$ takes as input a pair $(x,\varepsilon) \in \Sigma^* \times (0,1)$, and satisfies the following three conditions: (i) every oracle call made by $\mathcal{A}$ is of the form (w,δ), where $w \in \Sigma^*$ is an instance of g, and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \le \text{poly}(|x|, \varepsilon^{-1})$; (ii) the algorithm $\mathcal{A}$ meets the specification for being a randomised approximation scheme for f (as described above) whenever the oracle meets the specification for being a randomised approximation scheme for g; and (iii) the run-time of $\mathcal{A}$ is polynomial in $|x|$ and ε^{-1}. The key property of this notion of reducibility is that the class of functions computable by an FPRAS is closed under AP-reducibility.

If there is an AP-reduction from f to g then we say f is *AP-reducible to g* and write $f \le_{\mathsf{AP}} g$. If $f \le_{\mathsf{AP}} g$ and $g \le_{\mathsf{AP}} f$ then we say that f and g are *AP-interreducible* and write $f \equiv_{\mathsf{AP}} g$. A class of counting problems that are all AP-interreducible has the property that either all problems in the class have an FPRAS or none do. A word of warning about terminology: the notation $\le_{\mathsf{AP}}$ has also been used (see e.g. [4]) to denote a different type of approximation-preserving reduction which applies to optimisation problems. We will not study optimisation problems in this paper, so hopefully this will not cause confusion.

The class of problems AP-interreducible with #BIS, the problem of counting independent sets in a bipartite graph, has received particular attention. It is generally believed that problems in this class do not have an FPRAS.

3 Our main results

Our main results give a dichotomy for exactly solving #LocalMaxCSP(Γ) and a trichotomy for its approximation.

Theorem 3. *Let Γ be a constraint language with domain $\{0,1\}$.*

- *If every relation in Γ is essentially monotone then #LocalMaxCSP(Γ) is in* FP.

- *If every relation in Γ is affine then #LocalMaxCSP(Γ) is in* FP.

- *Otherwise, #LocalMaxCSP(Γ) is #P-complete.*

Theorem 4. *Let Γ be a constraint language with domain $\{0,1\}$.*

- *If every relation in Γ is essentially monotone then #LocalMaxCSP(Γ) is in* FP.

- *If every relation in Γ is affine then #LocalMaxCSP(Γ) is in* FP.

---

[1]The reader who is not familiar with oracle Turing machines can just think of this as an imaginary (unwritten) subroutine for computing g.

- *If Γ has a relation that is not essentially monotone and Γ has a relation that is not affine, but every relation in Γ is in IM_2 then $\#\mathsf{LocalMaxCSP}(\Gamma) \equiv_{\mathsf{AP}} \#\mathsf{BIS}$.*

- *If Γ has a relation that is not essentially monotone and a relation that is not affine and a relation that is not in IM_2 then $\#\mathsf{LocalMaxCSP}(\Gamma) \equiv_{\mathsf{AP}} \#\mathsf{SAT}$.*

Theorem 3 shows that if Γ contains a relation R that is not essentially monotone and a relation R' that is not affine then $\#\mathsf{LocalMaxCSP}(\Gamma)$ is $\#\mathsf{P}$-complete. It is not necessary for R and R' to be distinct. Similarly, the relations in Γ witnessing "not essentially monotone", non-affineness and non-containment in IM_2 will not in general be distinct.

4   Other notions of maximality

In Section 2.1 we defined *local maximality*. A satisfying assignment σ is locally maximal if, for all v with $\sigma(v) = 0$, the configuration $\sigma_{[v \to 1]}$ which is obtained from v by locally flipping the value of $\sigma(v)$ from 0 to 1 is unsatisfying.

The study of approximately counting locally-optimal structures is motivated by the following often-arising situation which is associated, for example, with Johnson, Papadimitriou and Yannakakis's complexity class PLS (polynomial-time local search) [12]. Often, it is easy to construct an arbitrary structure, difficult to construct a globally-optimal structure, and of intermediate complexity to construct a locally-optimal structure. A similar phenomenon arises in the study of *listing* combinatorial structures (see [9] for details). Notably, [9] found that this situation is not replicated in the context of approximately counting independent sets in bipartite graphs — a context in which approximately counting locally-optimal structures is (subject to complexity-theoretic assumptions) *more difficult* than counting globally-optimal structures. This was the motivation for the present paper, which studies the problem of approximately counting locally-optimal (locally maximal) structures in the context of CSPs.

In this section, we note that there are also other definitions of "maximal" that are not related to local optimality. One such example arises in the work of Durand, Hermann and Kolaitis [6]. Following them, we describe their work in terms of *minimality* rather than *maximality*, but this is not an essential difference. The essential difference is that their notion of minimality is based on subset inclusion rather than on local changes. They say that a satisfying assignment σ of a Boolean formula is *minimal* if there is no satisfying assignment σ' derived from σ by flipping any non-empty *set* of values in the assignment from 1 to 0. The analogous version of our definition would only allow one value to flip. They studied the problem $\#\mathsf{Circumscription}$, in which the input is a Boolean formula, and the output is the number of minimal satisfying assignments. Assuming that the counting hierarchy does not collapse, this problem is not even in $\#\mathsf{P}$. Indeed, they show [6, Theorem 5.1] that it is complete in Hemaspaandra and Vollmer's complexity class $\# \cdot \mathsf{coNP}$ [10], which is equivalent to Valiant's class $\#\mathsf{NP}$ [17, 18].[2] In [5], Durant and Hermann investigate the complexity of $\#\mathsf{Circumscription}$ when the input Boolean formula has a prescribed form. Their result contrasts sharply with Theorem 3. For example, they show [5, Theorem 4] that $\#\mathsf{Circumscription}$ is $\#\mathsf{P}$-complete when the formula is restricted to be affine, even though affine is one of the

---

[2]A theorem of Toda and Watanabe [16] tells us that $\#\mathsf{NP}$ is the same as $\#\mathsf{P}$ once we close with respect to polynomial-time Turing reductions, but Durand et al. study higher counting classes via subtractive reductions under which the higher counting classes are closed.

7

easy cases in Theorem 3. This difference illustrates how different subset-inclusion maximality and local maximality really are.

Counting globally optimal (*maximum*) structures is different from counting either type of *maximal* structures and there is also some interesting work about the former [14, 11].

5 Proofs

We now give the proofs of our main theorems, Theorem 3 and Theorem 4. We start with Theorem 3. Given a constraint language Γ, it is easy to see that #LocalMaxCSP(Γ) $\in$ #P — this is witnessed by the brute-force algorithm which checks every assignment to the CSP instance and checks whether it is a locally maximal satisfying assignment. (Note that this check can be done in polynomial time.) Theorem 3 follows from this fact and from Lemmas 5, 6 and 7, which we will prove in the remainder of the paper.

Lemma 5. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is essentially monotone then #LocalMaxCSP(Γ) is in* FP.

Lemma 6. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is affine then #LocalMaxCSP(Γ) is in* FP.

Lemma 7. *Let Γ be a constraint language with domain $\{0,1\}$. If Γ has a relation that is not affine and a relation that is not essentially monotone then #LocalMaxCSP(Γ) is #P-hard.*

All problems in #P are AP-reducible to #SAT [7] so #LocalMaxCSP(Γ) $\leq_{\text{AP}}$ #SAT. Theorem 4 follows from this fact, from Lemmas 5 and 6 and from the following additional lemmas, which we will also prove in the remainder of the paper.

Lemma 8. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is in IM_2 then #LocalMaxCSP(Γ) $\leq_{\text{AP}}$ #BIS.*

Lemma 9. *Let Γ be a constraint language with domain $\{0,1\}$. If Γ has a relation that is not affine and a relation that is not essentially monotone then #BIS $\leq_{\text{AP}}$ #LocalMaxCSP(Γ).*

Lemma 10. *Let Γ be a constraint language with domain $\{0,1\}$. If Γ has a relation that is not affine and a relation that is not essentially monotone and a relation that is not in IM_2 then #SAT $\leq_{\text{AP}}$ #LocalMaxCSP(Γ).*

5.1 Proofs of the easiness lemmas.

We start by proving Lemmas 5, 6 and 8.

Lemma 5. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is essentially monotone then #LocalMaxCSP(Γ) is in* FP.

Proof. Consider a CSP instance I with a set V of variables and a set C of constraints. Let U be the set of variables that are "pinned" to 0 by zero positions of R. Specifically,

$$U = \big\{v \in V \mid \text{ there is a constraint } R(v_1, \dots, v_k) \text{ in } C \text{ such that } v_i = v \text{ and } i \in \mathcal{Z}(R) \big\}.$$

Let $W = V \setminus U$. We claim that every locally maximal satisfying assignment of I maps the variables in U to the Boolean value 0 and the variables in W to the Boolean value 1.

8

Thus, there is at most one locally maximal satisfying assignment, and it is easy to check in polynomial time whether or not this satisfying assignment exists. Let us see why the claim is true. It is clear from the definition of $\mathcal{Z}(R)$ that every satisfying assignment maps all variables in U to 0. But all of the induced constraints on variables in W are monotone so if the instance I has a satisfying assignment then the assignment that maps all variables in W to 1 is the only locally maximal satisfying assignment. $\square$

Lemma 6. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is affine then #LocalMaxCSP(Γ) is in FP.*

Proof. Let Γ be an affine constraint language with domain $\{0,1\}$. Let k be the maximum arity of any constraint in Γ. We know from Section 2.2 that every constraint C over Γ is logically equivalent to a conjunction of constraints over $\mathbb{L}_k$ using the same variables as C. We can thus transform an instance I of #LocalMaxCSP(Γ) on variable set V into an equivalent instance J of #LocalMaxCSP($\mathbb{L}_k$) on the same set of variables. Satisfying assignments of I correspond to satisfying assignments of J and locally maximal satisfying assignments of I correspond to locally maximal satisfying assignments of J.

We will make one further transformation. Let $W \subseteq V$ be the set of variables that are constrained in the instance J (i.e., that occur in some constraint in J). Let $U = V \setminus W$ be the set of unconstrained variables. Let J' be the instance on variable set W obtained from J by removing the variables in U.

Now, in any locally maximal satisfying assignment of J, the variables U, being unconstrained, must take the value 1. Thus, there is a one-to-one correspondence between locally maximal satisfying assignments of J and locally maximal satisfying assignments of J'.

Finally, observe that every satisfying assignment of J' is locally maximal: flipping any variable v from 0 to 1 will violate all the constraints that involve v. So the instance J' has the same number of satisfying assignments as locally maximal satisfying assignments. Thus, we can count the locally maximal satisfying assignments of I by counting the locally maximal satisfying assignments of J which is the same as counting the locally maximal satisfying assignments of J' which is the same as counting all of the satisfying assignments of J'. This final step can be done by Gaussian elimination. $\square$

Lemma 8 follows directly from Lemma 13 and 14, which we prove next.

Lemma 13. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is in IM_2 then #LocalMaxCSP(Γ) $\leq_{\mathsf{AP}}$ #LocalMaxCSP($\{\mathsf{Implies}\}$).*

Proof. As we have noted in Section 2.2, the set IM_2 has the plain basis $B = \{\mathsf{Implies}, \mathsf{U}_0, \mathsf{U}_1\}$. Thus, we can thus transform an instance of #LocalMaxCSP(Γ) into an equivalent instance of #LocalMaxCSP(B) on the same set of variables. So to finish, we just need to give an AP-reduction from #LocalMaxCSP(B) to #LocalMaxCSP($\{\mathsf{Implies}\}$).

Let I be an instance of #LocalMaxCSP(B) on variable set V. It is convenient to model the structure of I as a directed graph $G(I)$ with vertex set V: There is a directed edge from u to v in $G(I)$ whenever there is a constraint $\mathsf{Implies}(u,v)$ in I. Define subsets $V_1(I), V_0(I)$ and $U(I)$ of V as follows. Note that these sets can be computed from I in polynomial time.

- $v \in V_1(I)$ if, for some $u \in V$, there is a constraint $\mathsf{U}_1(u)$ in I and there is a directed path from u to v in $G(I)$. Note that we include the empty directed path (of length 0) so for every constraint $\mathsf{U}_1(u)$ in I, the vertex u is in $V_1(I)$.

- $v \in V_0(I)$ if, for some $w \in V$, there is a constraint $\mathsf{U}_0(w)$ in I and there is a directed path from v to w in $G(I)$. Once again, we include length-0 paths.

- $U(I) = V \setminus (V_0(I) \cup V_1(I))$.

The instance I is satisfiable if and only if $V_0(I) \cap V_1(I)$ is non-empty. (To see this, note that if $V_0(I) \cap V_1(I)$ is non-empty, there is a variable u that is forced to take value 0 and value 1 in any satisfying assignment, which is impossible. On the other hand, if the intersection is empty, then the assignment that maps $V_0(I)$ to value 0 and the remaining vertices to value 1 is satisfying.)

If I is unsatisfiable, then our AP-reduction just returns the number of satisfying assignments of I (which is zero) without using the oracle for #LocalMaxCSP({Implies}).

Suppose instead that I is satisfiable. Let I' be the instance of #LocalMaxCSP({Implies}) on variable set U that is induced from I. That is, for any vertices u_1 and u_2 in U, Implies(u_1, u_2) is a constraint in I' if and only if it is a constraint in I. Then the locally maximal satisfying assignments of I are in one-to-one correspondence with the locally maximal satisfying assignments of I'. □

Lemma 14. #LocalMaxCSP({Implies}) $\leq_{\mathsf{AP}}$ #BIS.

Proof. Let I be an instance of #LocalMaxCSP({Implies}) on variable set V. As in the proof of Lemma 13, It is convenient to model the structure of I as a directed graph $G(I)$ with vertex set V: There is a directed edge from u to v in $G(I)$ whenever there is a constraint Implies(u, v) in I. In the following, we refer to the strongly-connected components of $G(I)$ as "components".

First, suppose that $|V| > 1$ and that $G(I)$ has a singleton component $\{v\}$. We will show below how to construct (in polynomial-time) an instance I' of #LocalMaxCSP({Implies}) on variable set $V - \{v\}$ such that the number of locally maximal satisfying assignments of I is equal to the number of locally maximal satisfying assignments of I'.

Before giving the details of the construction, we show how to use it to obtain the desired AP-reduction. Given I, we repeat the construction as many times as necessary to obtain an instance I^* of #LocalMaxCSP({Implies}) such that I^* has the same number of locally maximal satisfying assignments as I and either (1) I^* has only one variable, or (2) $G(I^*)$ has no singleton components. In Case (1), the number of locally maximal satisfying assignments of I^* (and hence of I) is one. So consider Case (2). Now note that every satisfying assignment of I^* is locally maximal since flipping the value of a single variable without flipping the rest of the variables in its component does not preserve satisfiability. Thus, the number of locally maximal satisfying assignments of I is equal to the number of satisfying assignments of I^*. To finish, we use an oracle for #BIS to approximately count the satisfying assignments of I^*. This is possible since #CSP({Implies}) is AP-reducible to #BIS by Theorem 2 (which is from [8]).

To finish the proof, we give the construction. So suppose that $|V| > 1$ and that $G(I)$ has a singleton component $\{v\}$. Let P be the (potentially empty) set of in-neighbours of vertex v in $G(I)$ and let S be the (potentially empty) set of out-neighbours of v. Construct I' from I by deleting variable v and all constraints involving v and adding all constraints Implies(u, w) for $u \in P$ and $w \in S$. To finish, we will give a bijection between the locally maximal satisfying assignments of I and I'.

We start by partitioning the locally maximal satisfying assignments of I and I' into three sets.

- Let $\Sigma_{1,*}$ be the set of locally maximal satisfying assignments σ of I for which there exists $u \in P$ with $\sigma(u) = 1$. Let $\Sigma'_{1,*}$ be the set of locally maximal satisfying assignments σ' of I' for which there exists $u \in P$ with $\sigma'(u) = 1$. We make the following deductions about every $\sigma \in \Sigma_{1,*}$ and $\sigma' \in \Sigma'_{1,*}$.

  (A1) $\sigma(v) = 1$. (This follows since σ is satisfying.)

  (A2) $\forall w \in S$, $\sigma(w) = \sigma'(w) = 1$. (This follows since σ and σ' are satisfying.)

  (A3) For every $u \in P$ with $\sigma'(u) = 0$, there is an out-neighbour z' of u in $G(I')$ which has $\sigma'(z') = 0$. For every $u \in P$ with $\sigma(u) = 0$, there is an out-neighbour z of u in $G(I)$ which is not equal to v and has $\sigma(z) = 0$.
  These follow since σ' and σ are maximal for u and $\sigma(v) = 1$.)

- Let $\Sigma_{*,0}$ be the set of locally maximal satisfying assignments σ of I for which there exists $w \in S$ with $\sigma(w) = 0$. Let $\Sigma'_{*,0}$ be the set of locally maximal satisfying assignments σ' of I' for which there exists $w \in S$ with $\sigma'(w) = 0$. We make the following deductions about every $\sigma \in \Sigma_{*,0}$ and $\sigma' \in \Sigma'_{*,0}$.

  (B1) $\sigma(v) = 0$. (This follows since σ is satisfying.)

  (B2) $\forall u \in P$, $\sigma(u) = \sigma'(u) = 0$. (This follows since σ and σ' are satisfying.)

- Let Σ_* be the set of all other locally maximal satisfying assignments of I. Let Σ'_* be the set of all other locally maximal satisfying assignments of I. We make the following deductions about every $\sigma \in \Sigma_*$ and $\sigma' \in \Sigma'_*$.

  (C1) For all $u \in P$, $\sigma(u) = \sigma'(u) = 0$. (This follows from the definitions of $\Sigma_{1,*}$ and $\Sigma'_{1,*}$.)

  (C2) For all $w \in S$, $\sigma(w) = \sigma'(w) = 1$. (This follows from the definitions of $\Sigma_{*,0}$ and $\Sigma'_{*,0}$.)

  (C3) $\sigma(v) = 1$. (This follows because σ is maximal for v.)

  (C4) For every $u \in P$, there is an out-neighbour z' of u in $G(I')$ which has $\sigma'(z') = 0$. For every $u \in P$, there is an out-neighbour $z \neq v$ of u in $G(I)$ which has $\sigma(z) = 0$. (This follows because σ' and σ are maximal for u and $\sigma(v) = 1$.)

Given $\sigma \in \Sigma_{1,*}$, let σ'' be the induced assignment of I'. Since by (A2) $\sigma(w) = 1$ for every $w \in S$, all of the new constraints (u, w) in I' are satisfied, so σ'' is satisfying. By construction, σ'' is maximal for the vertices outside of P (since vertices outside of P have the same out-neighbours in $G(I)$ and $G(I')$). We now check that it is maximal for vertices $u \in P$. This follows from (A3). Thus, we have given an injection from $\Sigma_{1,*}$ into $\Sigma'_{1,*}$. We now show that the reverse direction is an injection from $\Sigma'_{1,*}$ to $\Sigma_{1,*}$. Consider $\sigma' \in \Sigma'_{1,*}$ and let σ'' be the assignment of I formed from σ' by taking $\sigma''(v) = 1$ (satisfying (A1) for $\sigma = \sigma''$). Since by (A2) $\sigma'(w) = 1$ for all $w \in S$, we conclude that σ'' satisfies all of the constraints involving v in I so σ'' is satisfying. Once again, we must check that σ'' is maximal for vertices $u \in P$, and this follows from (A3).

In a similar way, we will establish a bijection from $\Sigma_{*,0}$ to $\Sigma'_{*,0}$. Given $\sigma \in \Sigma_{*,0}$, let σ'' be the induced assignment of I'. (B2) allows us to conclude that σ'' is satisfying. The edges in $G(I')$ from P to w allow us to conclude that σ'' is maximal for all $u \in P$ so it is locally maximal. Going the other direction, consider $\sigma' \in \Sigma_{*,0}$ and let σ'' be the assignment of I

formed from σ' by taking $\sigma''(v) = 0$ (satisfying (B1) for $\sigma = \sigma''$). Since by (B2) $\sigma'(u) = 0$ for all $u \in P$, σ'' is satisfying. Since $\sigma''(u) = 0$, σ'' is maximal for every vertex in P. Since $\sigma''(w) = 0$, σ'' is maximal for v. Thus, it is locally maximal.

In exactly the same way, we establish a bijection from Σ_* to Σ'_*. We conclude that I' has the same number of locally maximal satisfying assignments as I, so we have completed the proof. □

5.2 Maximality Gadgets

We will now examine gadgets that are useful for proving Lemmas 7, 9 and 10. We start with some useful definitions.

Definition 15. *Let I be a CSP instance with a set V of variables and a distinguished variable $r \in V$. We use $\mathcal{M}_0(I, r)$ to denote the number of locally maximal satisfying assignments σ of I with $\sigma(r) = 0$. We use $\mathcal{M}_1(I, r)$ to denote the number of locally maximal satisfying assignments σ of I with $\sigma(r) = 1$. Finally, we use $\mathcal{B}(I, r)$ to denote the number of satisfying assignments σ of I such that σ is maximal for every variable in $V \setminus \{r\}$ but σ is not maximal for r. A maximality gadget for a relation R is a CSP instance I with constraint language $\{R\}$ and distinguished variable r such that $\mathcal{M}_0(I, r) = \mathcal{M}_1(I, r) = 1$ and $\mathcal{B}(I, r) = 0$.*

Definition 15 could be weakened since we do not really need $\mathcal{M}_0(I, r)$ and $\mathcal{M}_1(I, r)$ to be 1, we only need them to be equal and non-zero. However, since our constructions satisfy the stronger definition, we simplify the remainder of the paper by using the stronger definition that we have stated.

Recall that R^* is the relation induced from R on the (non-zero) positions in $\mathcal{N}(R)$. We first relate maximality gadgets for R and R^*.

Lemma 16. *Let R be a Boolean relation. If there is a maximality gadget for R^* then there is a maximality gadget for R.*

Proof. Suppose R has arity h and R^* has arity $k \leq h$. Without loss of generality, suppose that that $\mathcal{N}(R) = \{1, \ldots, k\}$ and $\mathcal{Z}(R) = \{k + 1, \ldots, h\}$; thus R^* is the restriction of R to the first k places. Suppose we have a maximality gadget for R^*. That is, we have a CSP instance I^*, with R^*-constraints, on variables V, with a distinguished variable $r \in V$, satisfying Definition 15. We show how to construct a maximality gadget for R.

If $k = h$ then $R^* = R$ and there is nothing to show, so suppose $k < h$. Construct a new CSP instance I with R-constraints as follows. The variable set of I is $V \cup \{w\}$, where w is a new variable that is not in V. Replace each constraint $R^*(v_1, \ldots, v_k)$ in I^* by a constraint $R(v_1, \ldots, v_k, w, \ldots, w)$ in I. (Note that there are $h - k$ occurrences of the variable w.) Since $k < h$, the variable w is forced to 0 in any satisfying assignment of I. Thus, there is a bijection between satisfying assignments of I^* and satisfying assignments of I, obtained by setting w to 0. The bijection preserves locally maximality. From this bijection it is clear that $\mathcal{M}_0(I, r) = \mathcal{M}_0(I^*, r) = 1$, $\mathcal{M}_1(I, r) = \mathcal{M}_1(I^*, r) = 1$ and $\mathcal{B}(I, r) = \mathcal{B}(I^*, r) = 0$. Thus, I is a maximality gadget for R. □

Lemma 17 below shows how maximality gadgets can be constructed. Some of the constructions are a little bit reminiscent of the "strict, perfect, faithful implementations" of Creignou, Khanna, and Sudan [2] (see Lemmas 5.24 and 5.25, Claim 5.31 and Lemma 5.30).

In Lemmas 13–15 of [8], we use these implementations to realise one of the Boolean functions Implies, NAND $= \{(0,0),(0,1),(1,0)\}$ or OR $= \{(0,1),(1,0),(1,1)\}$ using a relation R. However, there are two important differences. (1) The implementations of [2] and [8] allow the use of auxiliary variables, and the use of these must be carefully controlled in order to satisfy the maximality constraints in Definition 15. (2) While implementations of Implies and NAND are useful for maximality gadgets, implementations of OR do not seem to be useful. Thus, the implementations from earlier papers do not suffice for building maximality gadgets.

Lemma 17. *Let R be a Boolean relation that is not essentially monotone. Then there is a maximality gadget for R.*

Proof. Since R is not essentially monotone, we know that R^* is not monotone. Suppose R^* has arity k; clearly $k \geq 1$ (in fact it is not difficult to see that $k \geq 2$). A simple but useful observation about R^* is the following.

$$\text{For all } i, \ 1 \leq i \leq k, \text{ there exists a tuple } (v_1, \ldots, v_k) \in R^* \text{ with } v_i = 1. \tag{2}$$

This follows from the fact that $i \notin \mathcal{Z}(R)$.

We consider several cases, depending on R. In each case, we construct a maximality gadget for R^*, which by Lemma 16 gives a maximality gadget for R.

Case 1 R^* **is 0-valid and 1-valid.** Since R^* is not monotone, it cannot be the complete relation. Let $s = (s_1, \ldots, s_k)$ be a tuple not in R^*. Our maximality gadget I has two variables r and x, and two constraints $R^*(a_1, \ldots, a_k)$ and $R^*(b_1, \ldots, b_k)$. Each a_j or b_j stands for an occurrence of r or x; specifically,

- If $s_j = 0$ then $a_j = r$ and $b_j = x$.
- If $s_j = 1$ then $a_j = x$ and $b_j = r$.

The following table lists the four possible assignments of (r,x) and analyses whether they are satisfying.

σ	$\sigma(r)$	$\sigma(x)$	$I(r,x)$ satisfied?
σ_1	0	0	yes, since R^* is 0-valid
σ_2	0	1	no, since $\sigma(a_1, \ldots, a_k) = s \notin R^*$
σ_3	1	0	no, since $\sigma(b_1, \ldots, b_k) = s \notin R^*$
σ_4	1	1	yes, since R^* is 1-valid

Since σ_1 and σ_4 are the unique satisfying assignments σ with $\sigma_1(r) = 0$ and $\sigma_4(r) = 1$, respectively, and both are locally maximal, we have $\mathcal{M}_0(I,r) = \mathcal{M}_1(I,r) = 1$. As there are no other satisfying assignments, $\mathcal{B}(I,r) = 0$. So the conditions for I to be a maximality gadget are satisfied.

Case 2 R^* **is 0-valid but not 1-valid.** Let m be the maximum number of ones in any tuple in R^*. By observation (2), $m \geq 1$, and since R^* is not 1-valid, $m < k$. Let $s = (s_1, \ldots, s_k)$ be a tuple in R^* with m ones. Again by observation (2), there is a tuple $s' = (s'_1, \ldots, s'_k) \in R^*$ such that, for some i, $s_i = 0$ and $s'_i = 1$. Note that the tuple $s'' = s \vee s'$ is not in R^*, since it has more than m ones. We split the analysis into two sub-cases.

13

(a) **The tuple $t = s \wedge \neg s'$ is in R^*.** (The tuple t has $t_j = 1$ precisely when $s_j = 1$ and $s'_j = 0$.) The gadget I will have variables r, x and w and constraints $R^*(w, \ldots, w)$ and $R^*(a_1, \ldots, a_k)$ where a_j is defined as follows.

- If $s'_j = 1$ then $a_j = x$.
- If $s'_j = 0$ and $s_j = 0$ then $a_j = w$.
- If $s'_j = 0$ and $s_j = 1$ then $a_j = r$.

Since R^* is 0-valid but not 1-valid, the constraint $R^*(w, \ldots, w)$ ensures that every satisfying assignment σ of I has $\sigma(w) = 0$. So we consider the four potential satisfying assignments.

σ	$\sigma(r)$	$\sigma(x)$	$\sigma(w)$	$I(r, x, w)$ satisfied?
σ_1	0	0	0	yes, since R^* is 0-valid
σ_2	0	1	0	yes, since $\sigma(a_1, \ldots, a_k) = s' \in R^*$
σ_3	1	0	0	yes, since $\sigma(a_1, \ldots, a_k) = t \in R^*$
σ_4	1	1	0	no, since $\sigma(a_1, \ldots, a_k) = s'' \notin R^*$

The assignment σ_1 is not maximal for x but σ_2 and σ_3, are locally maximal, so $\mathcal{M}_0(I, r) = \mathcal{M}_1(I, r) = 1$ and $\mathcal{B}(I, r) = 0$.

(b) **The tuple $t = s \wedge \neg s'$ is not in R^*.** The gadget I will have variables r, x and w and the constraints $R^*(w, \ldots, w)$, $R^*(a_1, \ldots, a_k)$ and $R^*(b_1, \ldots, b_k)$, where a_j and b_j are defined as follows.

- If $s_j = 0$ then $a_j = b_j = w$.
- If $s_j = 1$ and $s'_j = 0$ then $a_j = x$ and $b_j = r$.
- If $s_j = 1$ and $s'_j = 1$ then $a_j = r$ and $b_j = x$.

Since R^* is 0-valid but not 1-valid, every satisfying assignment σ has $\sigma(w) = 0$. So we consider the four possible satisfying assignments.

σ	$\sigma(r)$	$\sigma(x)$	$\sigma(w)$	$I(r, x, w)$ satisfied?
σ_1	0	0	0	yes, since R^* is 0-valid
σ_2	0	1	0	no, since $\sigma(a_1, \ldots, a_k) = t \notin R^*$
σ_3	1	0	0	no, since $\sigma(b_1, \ldots, b_k) = t \notin R^*$
σ_4	1	1	0	yes, since $\sigma(a_1, \ldots, a_k) = \sigma(b_1, \ldots, b_k) = s \in R^*$

Both satisfying assignments are locally maximal so $\mathcal{M}_0(I, r) = \mathcal{M}_1(I, r) = 1$ and $\mathcal{B}(I, r) = 0$.

Case 3 R^* **is not 0-valid but is 1-valid.** As R^* is not monotone, we may choose a tuple $s = (s_1, \ldots, s_k)$ in R^*, and an index $i \in [k]$ such that $s_i = 0$ and $s' = s \vee e_{i,k}$ is not in R^*. The gadget I will have variables r, x and y and the constraints $R^*(y, \ldots, y)$, $R^*(a_1, \ldots, a_k)$ and $R^*(b_1, \ldots, b_k)$ where $a_i = x$ and $b_i = r$ and for $j \neq i$, a_j and b_j are defined as follows.

- If $s'_j = 0$ then $a_j = r$ and $b_j = x$.
- If $s'_j = 1$ then $a_j = b_j = y$.

Since R^* is 1-valid but not 0-valid, the constraint $R^*(y, \ldots, y)$ ensures that every satisfying assignment σ of I has $\sigma(y) = 1$. We consider the potential satisfying assignments with $\sigma(y) = 1$.

σ	$\sigma(r)$	$\sigma(x)$	$\sigma(y)$	$I(r, x, y)$ satisfied?
σ_1	0	0	1	yes, since $\sigma(a_1, \ldots, a_k) = \sigma(b_1, \ldots, b_k) = s \in R^*$
σ_2	0	1	1	no, since $\sigma(a_1, \ldots, a_k) = s' \notin R^*$
σ_3	1	0	1	no, since $\sigma(b_1, \ldots, b_k) = s' \notin R^*$
σ_4	1	1	1	yes, since R^* is 1-valid

Note that σ_1 and σ_4 are locally maximal, so $\mathcal{M}_0(I, r) = \mathcal{M}_1(I, r) = 1$ and $\mathcal{B}(I, r) = 0$.

Case 4 R^* **is not** 0-**valid and not** 1-**valid.** We split the analysis into two sub-cases.

(a) **There is a tuple** $s \in R^*$ **such that** $\neg s \in R^*$. The gadget I will have variables r and x and the single constraint $R^*(a_1, \ldots, a_k)$ where

- If $s_j = 0$ then $a_j = r$.
- If $s_j = 1$ then $a_j = x$.

The potential satisfying assignments are

σ	$\sigma(r)$	$\sigma(x)$	$I(r, x)$ satisfied?
σ_1	0	0	no , since R^* is not 0-valid
σ_2	0	1	yes, since $\sigma(a_1, \ldots, a_k) = s \in R^*$
σ_3	1	0	yes since $\sigma(a_1, \ldots, a_k) = \neg s \in R^*$
σ_4	1	1	no, since R^* is not 1-valid

Note that σ_2 and σ_3 are locally maximal, so $\mathcal{M}_0(I, r) = \mathcal{M}_1(I, r) = 1$ and $\mathcal{B}(I, r) = 0$.

(b) **There is no tuple** s **with** $s \in R^*$ **and** $\neg s \in R^*$. Let m be the maximum number of ones in any tuple in R^*. By observation (2) $m \geq 1$, and since R^* is not 1-valid, $m < k$. Let $s = (s_1, \ldots, s_k)$ be a tuple in R^* with m ones. Again by observation (2) there is a tuple $s' = (s'_1, \ldots, s'_k) \in R^*$ such that, for some i, $s'_i = 1$ and $s_i = 0$. Note that the tuple $s \vee s'$ is not in R^*, since it has more than m ones. The gadget I will have four variables r, x, y and w and the constraints $R^*(a_1, \ldots, a_k)$ and $R^*(b_1, \ldots, b_k)$, where a_j and b_j are defined as follows.

- If $s_j = s'_j = 0$ then $a_j = b_j = w$.
- If $s_j = 0$ and $s'_j = 1$ then $a_j = w$ and $b_j = x$.
- If $s_j = 1$ and $s'_j = 0$ then $a_j = y$ and $b_j = r$.
- If $s_j = s'_j = 1$ then $a_j = b_j = y$.

Since R^* is not 0-valid or 1-valid, and $s \in R^*$ but $\neg s \notin R^*$, the constraint $R^*(a_1, \ldots, a_k)$ ensures that every satisfying assignment σ has $\sigma(w) = 0$ and $\sigma(y) = 1$. So we consider the four possible satisfying assignments.

σ	$\sigma(r)$	$\sigma(x)$	$\sigma(y)$	$\sigma(w)$	$I(r, x, y, w)$ satisfied?
σ_1	0	0	1	0	maybe
σ_2	0	1	1	0	yes, since $\sigma(b_1, \ldots, b_k) = s' \in R^*$
σ_3	1	0	1	0	yes, since $\sigma(b_1, \ldots, b_k) = s \in R^*$
σ_4	1	1	1	0	no, since $\sigma(b_1, \ldots, b_k) = s \vee s' \notin R^*$

15

There is no need to determine whether σ_1 is satisfying. If it is, then it is not maximal for x. The satisfying assignments σ_2 and σ_3 are locally maximal. So $\mathcal{M}_0(I,r) = \mathcal{M}_1(I,r) = 1$ and $\mathcal{B}(I,r) = 0$.

$\square$

5.3 Proofs of the hardness lemmas

We now prove Lemmas 7, 9 and 10.

Lemma 7. *Let Γ be a constraint language with domain $\{0,1\}$. If Γ has a relation that is not affine and a relation that is not essentially monotone then #LocalMaxCSP(Γ) is #P-hard.*

Proof. Suppose that R_1 is an arity k_1 relation in Γ that is not affine and R_2 is an arity-k_2 relation in Γ that is not essentially monotone. Let $k = k_1 + k_2$ and let R be the Cartesian product of R_1 and R_2. Specifically,

$$R = \{(x_1,\ldots,x_k) \mid (x_1,\ldots,x_{k_1}) \in R_1, (x_{k_1+1},\ldots,x_k) \in R_2\}.$$

Since R_1 is not affine, neither is R. Thus, Theorem 1 (due to Creignou and Hermann) shows that #CSP($\{R\}$) is #P-complete.

Since R_2 is not essentially monotone, R is not essentially monotone. Thus, we can use Lemma 17 to obtain a maximality gadget I for R with variable set V and some distinguished variable r. We will next use the maximality gadget to give a polynomial-time Turing reduction from #CSP($\{R\}$) to #LocalMaxCSP($\{R\}$).

First, consider the assignments of I. The definition of maximality gadget ensures the following.

1. Since $\mathcal{B}(I,r) = 0$, every satisfying assignment σ of I that is maximal for every variable in $V \setminus \{r\}$ is also maximal for r.

2. Since $\mathcal{M}_0(I,r) = 1$ there is exactly one satisfying assigment σ_0 of I that is maximal for every variable in $V \setminus \{r\}$ and satisfies $\sigma_0(r) = 0$. Note that σ_0 is maximal for r.

3. Since $\mathcal{M}_0(I,r) = 1$ there is exactly one satisfying assignment σ_1 of I that is maximal for every variable in $V \setminus \{r\}$ and satisfies $\sigma_1(r) = 1$. Note that σ_1 is maximal for r.

Now consider an instance J of #CSP($\{R\}$) with vertex set U. We will construct an instance J' of #LocalMaxCSP($\{R\}$) with $|V| \times |U|$ variables. For every variable $u \in U$, let V_u be a set of $|V|$ variables consisting of variable u and $|V| - 1$ new variables. Let I_u be a copy of the maximality gadget I using the variables V_u with distinguished variable u. Finally, let J' be the instance of #LocalMaxCSP($\{R\}$) with variable set $\bigcup_{u \in U} V_u$ and with all of the constraints in each of the instances I_u and with all of the further R-constraints inherited from J (these constraints inherited from J constrain the vertices in U).

We will next show that the satisfying assignments of J are in one-to-one correspondence with locally maximal satisfying assignments of J'. By construction, any locally maximal satisfying assignment of J' induces a satisfying assignment of J (just look at the induced assignment on variables in U).

Consider any satisfying assignment σ of J. Consider any variable $u \in U$. If $\sigma(u) = 0$ then by item (2) above, there is exactly one way to extend σ to the vertices in V_u that is maximal

16

for all variables in $V_u \setminus \{u\}$. This extension is also maximal for u itself. (Thus, even if the constraints in J would allow the assignment at u to be flipped to a 1, the unique extension of the assignment to V_u does not allow this.) If $\sigma(u) = 1$ then by item (3),there is exactly one way to extend σ to the vertices in V_u that is maximal for all variables in $V_u \setminus \{u\}$. This extension is also maximal for u itself. Thus, σ can be extended in exactly one way to a locally maximal satisfying assignment of J'.

So we have shown that the satisfying assignments of J are in one-to-one correspondence with locally maximal satisfying assignments of J'. Since $\#\mathsf{CSP}(\{R\})$ is #P-hard, we have proved that $\#\mathsf{LocalMaxCSP}(\{R\})$ is #P-hard.

Finally, there is a trivial polynomial-time Turing reduction from $\#\mathsf{LocalMaxCSP}(\{R\})$ to $\#\mathsf{LocalMaxCSP}(\Gamma)$ since every instance of $\#\mathsf{LocalMaxCSP}(\{R\})$ can be written as an instance of $\#\mathsf{LocalMaxCSP}(\Gamma)$. □

Lemma 9. *Let Γ be a constraint language with domain $\{0, 1\}$. If Γ has a relation that is not affine and a relation that is not essentially monotone then $\#\mathsf{BIS} \leq_{\mathsf{AP}} \#\mathsf{LocalMaxCSP}(\Gamma)$.*

Proof. Even though we require an AP-reduction, the proof is essentially the same as the proof of Lemma 7. Suppose that R_1 is an arity k_1 relation in Γ that is not affine and R_2 is an arity-k_2 relation in Γ that is not essentially monotone. Let $k = k_1 + k_2$ and let R be the Cartesian product of R_1 and R_2 as in the proof of Lemma 7. Since R is not affine, Theorem 2 (due to Dyer, Goldberg and Jerrum), together with $\#\mathsf{BIS} \leq_{\mathsf{AP}} \#\mathsf{SAT}$, which follows from the fact that every problem in #P is AP-reducible to $\#\mathsf{SAT}$ [7, Section 3], shows $\#\mathsf{BIS} \leq_{\mathsf{AP}} \#\mathsf{CSP}(\{R\})$.

The polynomial-time Turing reduction from $\#\mathsf{CSP}(\{R\})$ to $\#\mathsf{LocalMaxCSP}(\{R\})$ given in the proof of Lemma 7 is actually an AP-reduction since the satisfying assignments of J are in one-to-one correspondence with locally maximal satisfying assignments of J'. So we have established $\#\mathsf{BIS} \leq_{\mathsf{AP}} \#\mathsf{LocalMaxCSP}(\{R\})$.

Finally, $\#\mathsf{LocalMaxCSP}(\{R\}) \leq_{\mathsf{AP}} \#\mathsf{LocalMaxCSP}(\Gamma)$ since every instance of $\#\mathsf{LocalMaxCSP}(\{R\})$ can be written as an instance of $\#\mathsf{LocalMaxCSP}(\Gamma)$. □

Lemma 10. *Let Γ be a constraint language with domain $\{0, 1\}$. If Γ has a relation that is not affine and a relation that is not essentially monotone and a relation that is not in IM_2 then $\#\mathsf{SAT} \leq_{\mathsf{AP}} \#\mathsf{LocalMaxCSP}(\Gamma)$.*

Proof. Suppose that R_1 is an arity k_1 relation in Γ that is not affine, R_2 is an arity-k_2 relation in Γ that is not essentially monotone, and R_3 is an arity k_3 relation in Γ that is not in IM_2. Let $k = k_1 + k_2 + k_3$ and let R be the Cartesian product of R_1 and R_2 and R_3. Since R_1 is not affine and R_3 is not in IM_2, R is not affine and is not in IM_2. Thus, Theorem 2 shows $\#\mathsf{SAT} \leq_{\mathsf{AP}} \#\mathsf{CSP}(\{R\})$. Since R_2 is not essentially monotone, R is not essentially monotone. Thus, following the proof of Lemmas 7 and 9 we can use Lemma 17 to obtain an AP-reduction from $\#\mathsf{CSP}(\{R\})$ to $\#\mathsf{LocalMaxCSP}(\{R\})$ and we can write every instance of $\#\mathsf{LocalMaxCSP}(\{R\})$ as an instance of $\#\mathsf{LocalMaxCSP}(\Gamma)$ to obtain an AP-reduction from $\#\mathsf{LocalMaxCSP}(\{R\})$ to $\#\mathsf{LocalMaxCSP}(\Gamma)$. □

References

[1] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996.

[2] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2001.

[3] Nadia Creignou, Phokion Kolaitis, and Bruno Zanuttini. Structure identification of Boolean relations and plain bases for co-clones. *J. Comput. System Sci.*, 74(7):1103–1115, 2008.

[4] Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Twelfth Annual IEEE Conference on Computational Complexity (Ulm, 1997)*, pages 262–273. IEEE Computer Soc., Los Alamitos, CA, 1997.

[5] Arnaud Durand and Miki Hermann. On the counting complexity of propositional circumscription. *Inf. Process. Lett.*, 106(4):164–170, 2008.

[6] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theor. Comput. Sci.*, 340(3):496–513, 2005.

[7] Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2003.

[8] Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for Boolean #CSP. *J. Comput. Syst. Sci.*, 76(3-4):267–277, 2010.

[9] Leslie Ann Goldberg, Rob Gysel, and John Lapinskas. Approximately counting locally-optimal structures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 654–665. Springer, 2015.

[10] Lane A. Hemaspaandra and Heribert Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.

[11] Miki Hermann and Reinhard Pichler. Complexity of counting the optimal solutions. *Theor. Comput. Sci.*, 410(38-40):3814–3825, 2009.

[12] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[13] Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[14] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[15] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[16] Seinosuke Toda and Osamu Watanabe. Polynomial-time 1-Turing reductions from #PH to #P. *Theoret. Comput. Sci.*, 100(1):205–221, 1992.

[17] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

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