The Approximability of Constraint Satisfaction Problems[*]

Sanjeev Khanna[†] Madhu Sudan[‡] Luca Trevisan[§] David P. Williamson[¶]

October 21, 2000

Abstract

We study optimization problems that may be expressed as "Boolean constraint satisfaction problems". An instance of a Boolean constraint satisfaction problem is given by m constraints applied to n Boolean variables. Different computational problems arise from constraint satisfaction problems depending on the nature of the "underlying" constraints as well as on the goal of the optimization task. Here we consider four possible goals: MAX CSP (MIN CSP) is the class of problems where the goal is to find an assignment maximizing the number of satisfied constraints (minimizing the number of unsatisfied constraints). MAX ONES (MIN ONES) is the class of optimization problems where the goal is to find an assignment satisfying all constraints with maximum (minimum) number of variables set to 1. Each class consists of infinitely many problems and a problem within a class is specified by a finite collection of finite Boolean functions that describe the possible constraints that may be used.

Tight bounds on the approximability of every problem in MAX CSP were obtained by Creignou [11]. In this work we determine tight bounds on the "approximability" (i.e., the ratio to within which each problem may be approximated in polynomial time) of every problem in MAX ONES, MIN CSP and MIN ONES. Combined with the result of Creignou, this completely classifies all optimization problems derived from Boolean constraint satisfaction. Our results capture a diverse collection of optimization problems such as MAX 3-SAT, MAX CUT, MAX CLIQUE, MIN CUT, NEAREST CODEWORD etc. Our results unify recent results on the (in)approximability of these optimization problems and yield a compact presentation of most known results. Moreover, these results provide a formal basis to many statements on the behavior of natural optimization problems, that have so far only been observed empirically.

[*] Preliminary versions of parts of this paper appeared in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 11–20, El Paso, Texas, 4-6 May 1997; and *Proceedings, Twelfth Annual IEEE Conference on Computational Complexity*, pages 282–296, Ulm, Germany, 24-27 June 1997. IEEE Computer Society Press.

[†] sanjeev@cis.upenn.edu. Dept. of CIS, University of Pennsylvania, Philadelphia, PA 19104. Part of this work was done when this author was a graduate student at Stanford University, and another part was done when this author was at Bell Laboratories.

[‡] madhu@lcs.mit.edu. Laboratory for Computer Science, MIT, 545 Technology Square, Cambridge, MA 02139. Part of this work was done when this author was at the IBM Thomas J. Watson Research Center.

[§] luca@eecs.berkeley.edu. Computer Science Division, University of California at Berkeley, Berkeley, CA 94720. Work done at the University of Rome "La Sapienza".

[¶] dpw@almaden.ibm.com. IBM Almaden Research Center, K53/B1, 650 Harry Road, San Jose, CA 95120.

1 Introduction

The approximability of an optimization problem is the best possible performance ratio that is achieved by a polynomial-time approximation algorithm for the problem. The approximability is studied as a function of the input size and is always a function bounded from below by 1. Research in the nineties has led to dramatic progress in our understanding of the approximability of many central optimization problems. The results cover a large number of optimization problems, deriving tight bounds on the approximability of some, while deriving "asymptotically" tight bounds on many more. [1]

In this paper we study optimization problems derived from "Boolean constraint satisfaction problems" and present a complete classification of these problems based on their approximability. Our work is motivated by an attempt to unify this recent progress on the (in)approximability of combinatorial optimization problems. In the case of positive results, i.e., bounding the approximability from above, a few paradigms have been used repeatedly and these serve to unify the results nicely. In contrast, there is a lack of similar unification among negative or inapproximability results. Inapproximability results are established by approximation preserving reductions from hard problems, and such reductions tend to exploit every feature of the problem whose hardness is being shown, rather than isolating the "minimal" features that would suffice to obtain the hardness result. As a result inapproximability results are typically isolated, and are not immediately suited for unification.

The need for a unified study is however quite essential at this stage. The progress in the understanding of optimization problems has shown large amounts of diversity in their approximability. Despite this diversity, natural optimization problems do seem to exhibit some noticeable trends in their behavior. However in the absence of a terse description of known results it is hard to extract the trends; leave alone, trying to provide them with a formal basis. Some such trends are described below:

- There exist optimization problems that are solvable exactly, that admit polynomial time approximation schemes or PTAS (i.e., for every constant $\alpha > 1$, there exists a polynomial time α-approximation algorithm), that admit constant factor approximation algorithms, logarithmic factor approximation algorithms and polynomial factor approximation algorithms. But this list appears to be nearly exhaustive, raising the question: "Are there "natural" optimization problems with intermediate approximability?" [2]

- A number of minimization problems have an approximability of logarithmic factors. However so far no natural maximization problem has been shown to have a similar approximability, raising the question: "Are there any "natural" maximization problems which are approximable to within polylogarithmic factors, but no better?"

- Papadimitriou and Yannakakis [39] define a class of optimization problems called MAX SNP. This class has played a central role in many of the recent inapproximability results, and yet

[1] We say that the approximability of an optimization is known asymptotically, if we can determine a function $f : \mathcal{Z} \to \mathcal{Z}$ and constants c_1, c_2 such that the approximability is between $1 + f(n)$ and $1 + c_1 f(n^{c_2})$. This choice is based on the common choice of an approximation preserving reduction. See Definition 2.7.

[2] There are problems such as the *minimum feedback arc set* for which the best known approximation factor is $O(\log n \log \log n)$ [16] and the *asymmetric p-center problem* where the best known approximation factor is $O(\log^* n)$ [38]. However, no matching inapproximability results are known for such problems.

> even now the class does not appear to be fully understood. The class contains a number of NP-hard problems, and for all such known problems it turns out to be the case that the approximability is bounded away from 1! This raises the natural question: "Are there any NP-hard problems in MAX SNP that admit polynomial time approximation schemes?"

In order to study such questions, or even to place them under a formal setting, one needs to first specify the optimization problems in some uniform framework. Furthermore, one has to be careful to ensure that the task of determining whether the optimization problem studied is easy or hard (to, say, compute exactly) is decidable. Unfortunately, barriers such as Rice's theorem (which says this question may not in general be decidable) or Ladner's theorem (which says problems may not be just easy or hard [35]) force us to severely restrict the class of problems which can be studied in such a manner.

Schaefer [42] isolates one class of decision problems which can actually be classified completely. He obtains this classification by restricting his attention to "Boolean constraint satisfaction problems". A problem in this class is specified by a finite set $\mathcal{F}$ of Boolean functions on finitely many variables, referred to as the constraints. (These functions are specified by, say, a truth table.) A function $f : \{0,1\}^k \to \{0,1\}$, when applied to k variables $x_1, \ldots, x_k$ represents the constraint $f(x_1, \ldots, x_k) = 1$. An instance of a constraint satisfaction problem specified by $\mathcal{F}$ consists of m "constraint applications" on n Boolean variables where each constraint application is the application of one of the constraints from $\mathcal{F}$ to some ordered subset of the n variables. The language $\textsc{Sat}(\mathcal{F})$ consists of all instances which have an assignment satisfying all m constraints. Schaefer describes six classes of function families, such that if $\mathcal{F}$ is a subset of one of these classes, then the decision problem is in P, else he shows that the decision problem is NP-hard.

Our Setup: In this paper we consider four different optimization versions of Boolean constraint satisfaction problems. In each case the problem is specified by a family $\mathcal{F}$, and the instance by m constraints from $\mathcal{F}$ applied to n Boolean variables. The goals for the four versions vary as follows: In the problem MAX CSP($\mathcal{F}$) the goal is to find an assignment that maximizes the number of satisfied constraints. Analogously in the problem MIN CSP($\mathcal{F}$) the goal is to find an assignment that minimizes the number of unsatisfied constraints. Notice that while the problems are equivalent w.r.t. exact computation, their approximability may be (and often is) very different. In the problem MAX ONES($\mathcal{F}$) (MIN ONES($\mathcal{F}$)) the goal is to find an assignment satisfying all constraints, while maximizing (minimizing) the number of variables set to 1. We also consider the weighted version of all the above problems. In the case of WEIGHTED MAX CSP($\mathcal{F}$) (WEIGHTED MIN CSP($\mathcal{F}$)) the instance includes a non-negative weight for every constraint and the goal is to maximize (minimize) the sum of the weights of the satisfied (unsatisfied) constraints. In the case of WEIGHTED MAX ONES($\mathcal{F}$) (WEIGHTED MIN ONES($\mathcal{F}$)) the instance includes a non-negative weight for every variable and the goal is to find an assignment satisfying all constraint maximizing (minimizing) the weight of the variables set to 1. The collection of problems $\{\text{MAXCSP}(\mathcal{F}) \mid \mathcal{F} \text{ finite}\}$ yields the class MAX CSP, and similarly we get the classes (WEIGHTED) MIN CSP, MAX ONES, MIN ONES.

Together these classes capture a host of interesting optimization problems. MAX CSP is a subset of MAX SNP and forms a combinatorial core of the problems in MAX SNP. It also includes a number of well-studied MAX SNP-complete problems, including MAX 3-SAT, MAX 2-SAT, and MAX CUT. MAX ONES shows more varied behavior among maximization problems and includes MAX CLIQUE and a problem equivalent to MAX CUT. MIN CSP and MIN ONES are closely related to each other capturing very similar problems. The list of problems expressible as one of these includes: The s-t Min Cut problem, Vertex Cover, Hitting Set with bounded size sets, Integer programs with two variables per inequality [25], Min UnCut [20], Min 2CNF Deletion [33], and Nearest Codeword [2]. The ability to study all these different problems in a uniform framework and extract the features that make the problems easier/harder than the others shows the advantage of studying optimization problems under the constraint satisfaction framework.

We provide a complete characterization of the asymptotic approximability of every optimization problem in the classes mentioned above. For the class Max CSP such a classification was obtained by Creignou [11] who shows that every problem in the class is either solvable to optimality in polynomial time, or has a constant approximability bounded away from 1. For the remaining classes we provide complete characterizations. The detailed statement of our results, comprising of 22 cases, appear in Theorems 2.11-2.14. (This includes a technical strengthening of the results of Creignou [11].) In short the results show that every Max Ones problem is either solvable optimally in P, or has constant factor approximability, or polynomial approximability or it is hard to find feasible solutions. For the minimization problems, the results show that the approximability of every problem lies in one of at most 7 levels. However it does not pin down the approximability of every problem — but rather highlights a number of open problems in the area of minimization that deserve further attention. In particular, it exposes a class of problems for which Min UnCut is complete, a class for which Min 2CNF Deletion is complete and a class for which Nearest Codeword is complete. The approximability of these problems is not yet resolved.

Our results do indeed validate some of the observations about trends exhibited by optimization problems. We find that when restricted to constraint satisfaction problems; the following can be formally established. The approximability of optimization problems does come from a small number of levels; maximization problems do not have a log-approximable representative while minimization problems may have such representatives (e.g. Min UnCut). NP-hard Max CSP problems are also MAX SNP-hard. We also find that weights do not play any significant role in the approximability of combinatorial optimization problems, a thesis in the work of Crescenzi et al. [15][3].

Finally, we conclude with some thoughts on directions for further work. We stress that while constraint satisfaction problems provide a good collection of core problems to work with, they are by no means an exhaustive or even near-exhaustive collection of optimization problems. Our framework lacks such phenomena as polynomial time approximation schemes (PTAS); it does not capture several important optimization problems such as TSP and numerous scheduling, sequencing and graph partitioning problems. One possible reason for the non-existence of PTAS is that in our problems the input instances have no restrictions in the manner in which constraints may be imposed on the input variables. Significant insight may be gleaned by restricting the problem instances. A widely prescribed condition is that the incidence graph on the variables and the constraints should form a planar graph. This restriction has been studied by Khanna and Motwani [28] and they show that it leads to polynomial time approximation schemes for a general class of constraint satisfaction problems. Another input restriction of interest could be that variables are allowed to participate only in a bounded number of constraints. We are unaware of any work on this front. An important extension of our work would be to consider constraint families which contain constraints of unbounded arity (such as those included in the class Min $F^+\Pi_1$ studied by Kolaitis and Thakur [34]). Such an extension would allow us to capture problems such as Set Cover. Other directions include working with larger domain sizes (rather than Boolean domains for the variables), and working

[3] Our definition of an unweighted problem is more loose than that of Crescenzi et al. In their definition they disallow instances with repeated constraints, while we do allow them. We believe that it may be possible to remove this discrepancy from our work by a careful analysis of all proofs. We do not carry out this exercise here.

over spaces where the solution space is the set of all permutations of $[n]$ rather than $\{0,1\}^n$.

Related Work: The works of Schaefer [42] and Creignou [11] have already been mentioned above. We reproduce some of the results of Creignou in Theorem 2.11, with some technical strengthenings. This strengthening is described in Section 2.5. Another point of difference with the result of Creignou is that our techniques allow us to directly work with the approximability of optimization problems, while in her case the results formally establish NP-hardness and the hardness of approximation can in turn be derived from them. A description of these techniques appear in Section 2.6. Among other works focusing on classes showing dichotomy is that of Feder and Vardi [17] who consider the "largest" possible class of natural problems in NP that may exhibit a dichotomy. They motivate constraint satisfaction problems over larger domains and highlight a number of central open questions that lie on the path to the resolution of the complexity of deciding them. Creignou and Hermann [12] show a dichotomy result analogous to Schaefer's for counting versions of constraint satisfaction problems. In the area of approximability, the works of Lund and Yannakakis [37] and Zuckerman [45] provide two instances where large classes of problems are shown to be hard to approximate simultaneously — to the best of our knowledge these are the only cases where the results provide hardness for many problems simultaneously. Finally we mention a few results that are directly related to the optimization problems considered here. Trevisan et al. [43] provide an algorithm for finding optimal implementations (or "gadgets" in their terminology) reducing between MAX CSP problems. Karloff and Zwick [27] describe generic methods for finding "semidefinite relaxations" of MAX CSP problems - and use these to provide approximation algorithms for these problems. These results further highlight the appeal of the "constraint satisfaction" framework for studying optimization problems.

2 Definitions and Results

2.1 Constraints, Constraint Applications and Constraint Families

We start by formally defining constraints and constraint satisfaction problems. Schaefer's work [42] proposes the study of such problems as a generalization of 3-satisfiability (3-SAT). We will use the same example to illustrate the definitions below.

A constraint is a function $f : \{0,1\}^k \rightarrow \{0,1\}$. A constraint f is satisfied by an input $s \in \{0,1\}^k$ if $f(s) = 1$. A constraint family $\mathcal{F}$ is a finite collection of constraints $\{f_1, \ldots, f_l\}$. For example, constraints of interest for 3-SAT are described by the constraint family $\mathcal{F}_{3\text{SAT}} = \{\text{OR}_{k,j} : 1 \leq k \leq 3, 0 \leq j \leq k\}$, where $\text{OR}_{k,j} : \{0,1\}^k \rightarrow \{0,1\}$ denotes the constraint $\neg x_1 \bigvee \cdots \bigvee \neg x_j \bigvee x_{j+1} \bigvee \cdots \bigvee x_k$. A constraint application, of a constraint f to n Boolean variables, is a pair $\langle f, (i_1, \ldots, i_k)\rangle$, where the indices $i_j \in [n]$ select k of the n Boolean variables to whom the the constraint is applied. (Here and throughout the paper we use the notation $[n]$ to denote the set $\{1, \ldots, n\}$.) For example to generate the clause $(x_5 \bigvee \neg x_3 \bigvee x_2)$ we could use the constraint application $\langle \text{OR}_{3,1}, (3,5,2)\rangle$ or $\langle \text{OR}_{3,1}, (3,2,5)\rangle$. Note that the applications allow the constraint to be applied to different ordered sets of variables but *not literals*. This distinction is an important one, and is the reason that we need all the constraints $\text{OR}_{3,0}, \text{OR}_{3,1}$ etc. to describe 3-SAT. In a constraint application $\langle f, (i_1, \ldots, i_k)\rangle$, we require that $i_j \neq i_{j'}$ for $j \neq j'$, i.e., the variables are not allowed to be replicated within a constraint application. This is why we need both the functions $\text{OR}_{2,0}$ as well as $\text{OR}_{3,0}$ in 3-SAT.

Constraints and constraint families are the ingredients that specify an optimization *problem.* Thus it is necessary that their description be finite. (Notice that the description of $\mathcal{F}_{\rm 3SAT}$ is finite.) Constraint applications are used to specify *instances* of optimization problems (as well as instances of Schaefer's generalized satisfiability problems) and the fact that their description lengths grow with the instance size is crucially exploited here. (Notice that the description size of a constraint application used to describe a 3-SAT clause will be $\Omega(\log n)$.) While this distinction between constraints and constraint applications is important, we will often blur this distinction in the rest of this paper. In particular we may often let the constraint application $C = \langle f, (i_1, \ldots, i_k)\rangle$ refer just to the constraint f. In particular, we will often use the expression "$C \in \mathcal{F}$" when we mean "$f \in \mathcal{F}$, where f is the first component of C". We now describe Schaefer's class of satisfiability problems and the optimization problems considered in this paper.

Definition 2.1 (Sat($\mathcal{F}$))
Instance: *A collection of m constraint applications of the form $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^m$, on Boolean variables $x_1, x_2, ..., x_n$ where $f_j \in \mathcal{F}$ and k_j is the arity of f_j.*
Objective: *Decide if there exists a Boolean assignment to the x_i's which satisfies all the constraints.*

For example, the problem Sat($\mathcal{F}_{\rm 3SAT}$) is the classical 3-SAT problem.

Definition 2.2 (Max CSP($\mathcal{F}$) (Min CSP($\mathcal{F}$)))
Instance: *A collection of m constraint applications of the form $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^m$, on Boolean variables $x_1, x_2, ..., x_n$ where $f_j \in \mathcal{F}$ and k_j is the arity of f_j.*
Objective: *Find a Boolean assignment to x_i's so as to maximize (minimize) the number of satisfied (unsatisfied) constraints.*

In the weighted problem Weighted Max CSP($\mathcal{F}$) *(*Weighted Min CSP($\mathcal{F}$)*) the input instance includes m non-negative weights $w_1 \ldots, w_m$ and the objective is to find an assignment which maximizes (minimizes) the sum of the weights of the satisfied (unsatisfied) constraints.*

Definition 2.3 (Max Ones($\mathcal{F}$) (Min Ones($\mathcal{F}$)))
Instance: *A collection of m constraint applications of the form $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^m$, on Boolean variables $x_1, x_2, ..., x_n$ where $f_j \in \mathcal{F}$ and k_j is the arity of f_j.*
Objective: *Find a Boolean assignment to x_i's which satisfies all the constraints and maximizes (minimizes) the total number of variables assigned true.*

In the weighted problem Weighted Max Ones($\mathcal{F}$) *(*Weighted Min Ones($\mathcal{F}$)*) the input instance includes n non-negative weights $w_1 \ldots, w_n$ and the objective is to find an assignment which satisfies all constraints and maximizes (minimizes) the sum of the weights of variables assigned true.*

The class (Weighted) Max CSP is the set of all optimization problems (Weighted) Max CSP($\mathcal{F}$) for every constraint family $\mathcal{F}$. The classes (Weighted) Max Ones, Min CSP, Min Ones are defined similarly.

The optimization problem Max 3Sat is easily seen to be equivalent to Max CSP($\mathcal{F}_{\rm 3SAT}$). This and the other problems Max Ones($\mathcal{F}_{\rm 3SAT}$), Min CSP($\mathcal{F}_{\rm 3SAT}$) and Min Ones($\mathcal{F}_{\rm 3SAT}$) are considered in the rest of this paper. More interesting examples of Max Ones, Min CSP and Min Ones problems are described in Section 2.3. We start with some preliminaries on approximability that we need to state our results.

2.2 Approximability, Reductions and Completeness

A *combinatorial optimization* problem is defined over a set of *instances* (admissible input data); a finite set $\mathsf{sol}(x)$ of *feasible solutions* is associated to any instance. An *objective function* attributes an integer value to any solution. The *goal* of an optimization problem is, given an instance x, find a solution $y \in \mathsf{sol}(x)$ of *optimum* value. The optimum value is the largest one for *maximization* problems and the smallest one for *minimization* problems. A combinatorial optimization problem is said to be an NPO problem if instances and solutions can be recognized in polynomial time, solutions are polynomial-bounded in input size, and the objective function can be computed in polynomial time (see e.g. [10]).

Definition 2.4 (Performance Ratio) *A solution s to an instance $\mathcal{I}$ of an* NPO *problem A is r-approximate if it has a value V satisfying:*

$$\max\left\{\frac{V}{\text{OPT}(\mathcal{I})}, \frac{\text{OPT}(\mathcal{I})}{V}\right\} \leq r.$$

An approximation algorithm for an NPO *problem A has* performance ratio $\mathcal{R}(n)$ *if, given any instance $\mathcal{I}$ of A with $|\mathcal{I}| = n$, it outputs an $\mathcal{R}(n)$-approximate solution.*

We say that a NPO problem is approximable to within a factor $\mathcal{R}(n)$ if it has a polynomial-time approximation algorithm with performance ratio $\mathcal{R}(n)$.

Definition 2.5 (Approximation Classes) *An* NPO *problem A is in the class* PO *if it is solvable to optimality in polynomial time. A is in the class* APX *(resp.* log-APX/ poly-APX*) if there exists a polynomial-time algorithm for A whose performance ratio is bounded by a constant (resp. logarithmic/polynomial factor in the size of the input).*

Completeness in approximation classes can be defined using appropriate approximation preserving reducibilities. In this paper, we use two notions of reducibility: A-reducibility and AP-reducibility. We discuss the difference between the two and the need for having two different notions after the definitions.

Definition 2.6 (A-reducibility [14]) *An* NPO *problem A is said to be A-reducible to an* NPO *problem B, denoted $A \leq_{\text{A}} B$, if two polynomial time computable functions F and G and a constant α exist such that:*

(1) *For any instance $\mathcal{I}$ of A, $F(\mathcal{I})$ is an instance of B.*

(2) *For any instance $\mathcal{I}$ of A and any feasible solution $\mathcal{S}'$ for $F(\mathcal{I})$, $G(\mathcal{I}, \mathcal{S}')$ is a feasible solution for $\mathcal{I}$.*

(3) *For any instance $\mathcal{I}$ of A and any $r \geq 1$, if $\mathcal{S}'$ is a r-approximate solution for $F(\mathcal{I})$ then $G(\mathcal{I}, \mathcal{S}')$ is an (αr)-approximate solution for $\mathcal{I}$.*

Definition 2.7 (AP-reducibility [13]) *For a constant $\alpha > 0$ and two* NPO *problems A and B, we say that A is α-AP-reducible to B, denoted $A \leq_{\text{AP}} B$, if two polynomial-time computable functions F and G exist such that the following holds:*

(1) *For any instance $\mathcal{I}$ of A, $F(\mathcal{I})$ is an instance of B.*

(2) *For any instance $\mathcal{I}$ of A, and any feasible solution $\mathcal{S}'$ for $F(\mathcal{I})$, $G(\mathcal{I}, \mathcal{S}')$ is a feasible solution for $\mathcal{I}$.*

(3) *For any instance $\mathcal{I}$ of A and any $r \geq 1$, if $\mathcal{S}'$ is an r-approximate solution for $F(\mathcal{I})$, then $G(\mathcal{I}, \mathcal{S}')$ is an $(1 + (r-1)\alpha + o(1))$-approximate solution for $\mathcal{I}$, where the $o()$-notation is with respect to $|\mathcal{I}|$.*

We say that A is AP-reducible to B if a constant $\alpha > 0$ exists such that A is α-AP-reducible to B.

Remark:

1. Notice that Conditions (3) of both reductions only preserve the quality of an approximate solution in absolute terms (to within the specified limits) and not as functions of the instance size. For example, an A-reduction from Π to Π' which blows up instance size by quadratic factor and an $O(n^{1/3})$ approximation algorithm for Π' combine to give only an $O(n^{2/3})$ approximation algorithm for Π.

2. The difference between the two reductions is the level of approximability that is preserved by them. (Condition (3) in the definitions.) A-reductions preserve constant factor approximability or higher, i.e., if Π is A-reducible to Π' and Π' is approximable to within a factor of $r(n)$, then Π is approximable to within $\alpha r(n^c)$ for some constants α, c. This property suffices to preserve membership in APX (log-APX, poly-APX), i.e., if Π is in APX (log-APX, poly-APX) then Π' is also in APX (resp. log-APX, poly-APX). However it does not preserve membership in PO or PTAS, as can be observed by setting $r = 1$.

3. AP-reductions reductions are more sensitive than A-reductions. Thus if Π is AP-reducible to Π, then an r-approximate solution is mapped to an $h(r)$ approximate solution where $h(r) \rightarrow 1$ as $r \rightarrow 1$. Thus AP-reductions preserve membership in PTAS as well. However they need not preserve membership in PO (due to the $o(1)$-term in their preservation of approximability).

4. Condition (3) of the definition of AP-reductions is strictly stronger than the corresponding condition in the definition of A-reductions. Thus, every AP-reduction is also an A-reduction. Unfortunately, neither one of these reductions on their own suffice for our purposes. We need AP-reductions to show APX-hardness of problems, but we need the added flexibility of A-reductions for other hardness results.

5. The original definitions of AP-reducibility and A-reducibility of [14] and [13] were more general. Under the original definitions, the A-reducibility does not preserve membership in log-APX, and it is not clear whether every AP-reduction is also an A-reduction. The restricted versions defined here are more suitable for our purposes. In particular, it is true that the Vertex Cover problem is APX-complete under our definition of AP-reducibility.

Definition 2.8 (APX**,** log-APX**, and** poly-APX**-completeness)** *An* NPO *problem* Π *is* APX*-hard if every* APX *problem is AP-reducible to* Π*. An* NPO *problem* Π *is* log-APX*-hard (*poly-APX*-hard) if every* log-APX *(*poly-APX*) problem is A-reducible to* Π*. A problem* Π *is* APX*(*log-APX, poly-APX*)-complete if it is in* APX *(resp.* log-APX*,* poly-APX*) and it is* APX*(resp.* log-APX*,* poly-APX*)-hard.*

The class APX contains the class MAX SNP as defined by Papadimitriou and Yannakakis [39]. The containment is strict in a syntactic sense (e.g. MAX SNP does not contain any minimization problems); however, when one takes the closure of APX under AP-reductions, one obtains the class MAX SNP [29]. The notion of reductions used here is also less stringent than the notion of reduction used in [39]. Thus APX, APX-hardness, and APX-completeness are (mild) generalizations of the notions of MAX SNP, MAX SNP-hardness, and MAX SNP-completeness.

Most problems we consider are known/shown to be in PO, or else are APX-complete or poly-APX-complete. However in some cases, we will not be able to establish the exact approximability of a given problem. However, we will nevertheless be able to compile all problems into a finite number of equivalence classes, with some equivalence classes being defined as "problems equivalent to Π" for some problem Π of unknown approximability. The following definition captures this concept.

Definition 2.9 (Π-completeness) *For* NPO *problems* Π *and* Π', Π' *is said to be* Π-*complete if* $\Pi \leq_{\mathrm{A}} \Pi'$ *and* $\Pi' \leq_{\mathrm{A}} \Pi$.

2.3 Problems captured by Max CSP, Max Ones, Min CSP and Min Ones

We first specify our notation for commonly used functions.

- **0** and **1** are the functions which are always satisfied and never satisfied respectively. Together these are the trivial functions. We will assume that all our function families do not have any trivial functions.
- T and F are unary functions given by $T(x) = x$ and $F(x) = \neg x$.
- For a positive integer i and non-negative integer $j \leq i$, $\mathrm{OR}_{i,j}$ is the function on i variables given by $\mathrm{OR}_{i,j}(x_1, \ldots x_i) = \neg x_1 \bigvee \cdots \bigvee \neg x_j \bigvee x_{j+1} \bigvee \cdots \bigvee x_i$. $\mathrm{OR}_i = \mathrm{OR}_{i,0}$; $\mathrm{NAND}_i = \mathrm{OR}_{i,i}$; $\mathrm{OR} = \mathrm{OR}_2$; $\mathrm{NAND} = \mathrm{NAND}_2$.
- Similarly, $\mathrm{AND}_{i,j}$ is given by $\mathrm{AND}_{i,j}(x_1, \ldots x_i) = \neg x_1 \bigwedge \cdots \bigwedge \neg x_j \bigwedge x_{j+1} \bigwedge \cdots \bigwedge x_i$. $\mathrm{AND}_i = \mathrm{AND}_{i,0}$; $\mathrm{NOR}_i = \mathrm{AND}_{i,i}$; $\mathrm{AND} = \mathrm{AND}_2$; $\mathrm{NOR} = \mathrm{NOR}_2$.
- The function XOR_i is given by $\mathrm{XOR}(x_1, \ldots, x_i) = x_1 \oplus \cdots \oplus x_i$. $\mathrm{XOR} = \mathrm{XOR}_2$.
- The function XNOR_i is given by $\mathrm{XNOR}(x_1, \ldots, x_i) = \neg(x_1 \oplus \cdots \oplus x_i)$. $\mathrm{XNOR} = \mathrm{XNOR}_2$.

Now we enumerate some interesting maximization and minimization problems which are "captured" by (i.e., are equivalent to some problem in) Max CSP, Max Ones, Min CSP and Min Ones. The following list is interesting for several reasons. First, it highlights the importance of these classes as ones that contain interesting optimization problems, and shows the diversity of the problems captured by these classes. Furthermore, each of these problems turn out to be "complete" problems for the partitions they belong to. Some are even necessary for a full statement of our results. Last, for several of the minimization problems listed below, their approximability is not yet fully resolved. We feel that these problems are somehow representative of the lack of our understanding of the approximability of minimization problems. We start with the maximization problems.

- For any positive integer k, Max kSat = Max CSP($\{\mathrm{OR}_{i,j} | i \in [k], 0 \leq j \leq i\}$). Max kSat is a well-studied problem and known to be MAX SNP-complete [39], for $k \geq 2$. Every MAX SNP-complete problem is in APX (i.e., approximable to within a constant factor in polynomial time) [39]. Also for MAX SNP-complete problem there exists a constant α greater than 1, such that the problem is problem is not α-approximable unless NP = P [3].
- For any positive integer k, Max EkSat = Max CSP($\{\mathrm{OR}_{k,j} | 0 \leq j \leq k\}$). The problem Max E$k$Sat is a variant of Max kSat restricted to have clauses of length exactly k.
- Max Cut = Max CSP($\{\mathrm{XOR}\}$). Max Cut is also MAX SNP-complete [39] and the best known approximation algorithm for this problem, due to [22], achieves a performance ratio of $1.14 \approx 1/.878$

- Max Clique = Max Ones(NAND). Max Clique is known to be approximable to within a factor of $O(n/\log^2 n)$ in an n-vertex graph [9] and is known to be hard to approximate to within a factor of $\Omega(n^{1-\epsilon})$ for any $\epsilon > 0$ unless NP = RP [18, 23].

We now go on to the minimization problems.

- The well known minimum s-t cut problem in directed graphs is equivalent to Weighted Min CSP($\mathcal{F}$) for $\mathcal{F} = \{\mathrm{OR}_{2,1}, T, F\}$. This is shown in Section 5.1. This problem is well-known to be solvable exactly in polynomial time.

- The Hitting Set problem restricted to instances in which all sets are of size at most B, can be captured as Min Ones($\mathcal{F}$) for $\mathcal{F} = \{\mathrm{OR}_k | k \leq B\}$. Also, of interest to our paper is a slight generalization of this problem which we call the Implicative Hitting Set-B Problem (Min IHS-B) which is Min CSP($\{\mathrm{OR}_k : k \leq B\} \cup \{\mathrm{OR}_{2,1}, F\}$). The Min Ones version of this problem will be of interest to us as well. The Hitting Set-B problem is well-known to be approximable to within a factor of B. We show that Min IHS-B is approximable to within a factor of $B + 1$.

- Min UnCut = Min CSP({XOR}). This problem has been studied previously by Klein et al. [32] and Garg et al. [20]. The problem is known to be MAX SNP-hard and hence not approximable to within some constant factor greater than 1. On the other hand, the problem is known to be approximable to within a factor of $O(\log n)$ [20].

- Min 2CNF Deletion = Min CSP({OR, NAND}). This problem has been studied by Klein et al. [33]. They show that the problem is MAX SNP-hard and that it is approximable to within a factor of $O(\log n \log \log n)$.

- Nearest Codeword = Min CSP($\{\mathrm{XOR}_3, \mathrm{XNOR}_3\}$). This is a classical problem for which hardness of approximation results have been shown by Arora et al. [2]. The Min Ones version of this problem is essentially identical to this problem. For both problems, the hardness result of Arora et al. [2] shows that approximating this problem to within a factor of $\Omega(2^{\log^{1-\epsilon} n})$ is hard for every $\epsilon > 0$, unless NP $\subseteq$ QP. No non-trivial approximation guarantees are known for this problem (the trivial bound being a factor of m, which is easily achieved since deciding if all equations are satisfiable amounts to solving a linear system).

- Lastly we also mention one more problem which is required to present our main theorem. Min Horn Deletion = Min CSP($\{\mathrm{OR}_{3,1}, T, F\}$). The currently known bounds on the approximability of this problem are similar to those of the Nearest Codeword, i.e., it is in poly-APX and hard to approximate to within a factor of $2^{\Omega(\log^{1-\epsilon} n)}$ (see Lemma 7.21).

2.4 Properties of function families

We start with the six properties defined by Schaefer:

- A constraint f is *0-valid* (resp. *1-valid*) if $f(0, \ldots, 0) = 1$ (resp. $f(1, \ldots, 1) = 1$).

- A constraint is *weakly positive* (resp. *weakly negative*) if it can be expressed as a CNF-formula having at most one negated variable (resp. at most one unnegated variable[4]) in each clause.

[4] Such clauses are usually called Horn clauses.

- A constraint is *affine* if it can be expressed as a conjunction of linear equalities over $\mathcal{Z}_2$.

- A constraint is *2cnf* if it is expressible as a 2CNF-formula.

The above definitions extend to constraint families naturally. For instance, a constraint family $\mathcal{F}$ is *0-valid* if *every* constraint $f \in \mathcal{F}$ is 0-valid. The above definitions are central to Schaefer's main theorem, restated below.

Theorem 2.10 (Schaefer's Theorem [42]) *For any constraint family $\mathcal{F}$,* SAT($\mathcal{F}$) *is in* P *if $\mathcal{F}$ is* 0*-valid or* 1*-valid or weakly positive or weakly negative or affine or 2cnf; else deciding* SAT($\mathcal{F}$) *is* NP*-hard.*

We use the shorthand "$\mathcal{F}$ is (not) decidable" to say that deciding membership in SAT($\mathcal{F}$) is solvable in P (is NP-hard). Abusing our vocabulary slightly, we say MAX ONES($\mathcal{F}$) (or MIN ONES($\mathcal{F}$)) is not decidable, to indicate that determining if a given instance of this problem has a feasible solution is NP-hard.

We need to define some additional properties to describe the approximabilities of the optimization problems we consider:

- f if *2-monotone* if $f(x_1, \ldots, x_k)$ is expressible as $(x_{i_1} \bigwedge \cdots \bigwedge x_{i_p}) \bigvee (\neg x_{j_1} \bigwedge \cdots \bigwedge \neg x_{j_q})$, for some $p, q \geq 0$, $(p, q) \neq (0, 0)$ (i.e., f is expressible as a DNF-formula with at most two terms - one containing only positive literals and the other containing only negative literals).

- A constraint is *width-2 affine* if it is expressible as a conjunction of linear equations over $\mathcal{Z}_2$ such that each equation has at most 2 variables.

- A constraint is *strongly 0-valid* if it is satisfied by all assignments with at most one 1. (Note that a strongly 0-valid constraint is also 0-valid.)

- A constraint f is *IHS-B+* (for *Implicative Hitting Set-Bounded+*) if it is expressible as a CNF formula where the clauses are of one of the following types: $x_1 \bigvee \cdots \bigvee x_k$ for some positive integer $k \leq B$, or $\neg x_1 \bigvee x_2$, or $\neg x_1$. IHS-$B-$ constraints and constraint families are defined analogously (with every literal being replaced by its complement). A family is a IHS-B family if the family is a IHS-$B+$ family or a IHS-$B-$ family.

We use the following shorthand for the above families: (1) $\mathcal{F}_0$ is the family of 0-valid constraints; (2) $\mathcal{F}_1$ is the family of 1-valid constraints; (3) $\mathcal{F}_{\mathrm{S0}}$ is the family of strongly 0-valid constraints; (4) $\mathcal{F}_{\mathrm{2M}}$ is the family of 2-monotone constraints; (5) $\mathcal{F}_{\mathrm{IHS}}$ is the family of IHS-B constraints; (6) $\mathcal{F}_{\mathrm{2A}}$ is the family of width-2 affine constraints; (7) $\mathcal{F}_{\mathrm{2CNF}}$ is the family of 2CNF constraints; (8) $\mathcal{F}_{\mathrm{A}}$ is the family of affine constraints; (9) $\mathcal{F}_{\mathrm{WP}}$ is the family of weakly positive constraints; (10) $\mathcal{F}_{\mathrm{WN}}$ is the family of weakly negative constraints.

2.5 Main Results

We now present the main results of this paper. A pictorial representation is available in Appendices B.1, B.2, B.3 and B.4. All theorems are stated assuming that $\mathcal{F}$ has no trivial constraints, i.e., constraints that are always satisfied or never satisfied. The first theorem is a minor strengthening of Creignou's theorem [11] so as to cover problems such as MAX EkSAT. The remaining theorems cover MAX ONES, MIN CSP and MIN ONES respectively.

Theorem 2.11 (MAX CSP **classification)** *For any constraint set $\mathcal{F}$, the problem (*WEIGHTED*)* MAX CSP($\mathcal{F}$) *is always either in* PO *or is* APX*-complete. Furthermore, it is in* PO *if and only if $\mathcal{F}$ is 0-valid or 1-valid or 2-monotone.*

Theorem 2.12 (MAX ONES **classification)** *For any constraint set $\mathcal{F}$, the problem (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is either in* PO *or is* APX*-complete or* poly-APX*-complete or decidable but not approximable to within any factor or not decidable. Furthermore,*

(1) *If $\mathcal{F}$ is 1-valid or weakly positive or affine with width* 2*, then (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is in* PO*.*

(2) *Else if $\mathcal{F}$ is affine then (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is* APX*-complete.*

(3) *Else if $\mathcal{F}$ is strongly* 0*-valid or weakly negative or 2CNF then (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is* poly-APX*-complete.*

(4) *Else if $\mathcal{F}$ is 0-valid then* SAT($\mathcal{F}$) *is in* P *but finding a solution of positive value is* NP*-hard.*

(5) *Else finding a feasible solution to (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is* NP*-hard.*

Theorem 2.13 (MIN CSP **classification)** *For any constraint set $\mathcal{F}$, the problem (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is in* PO *or is* APX*-complete or* MIN UNCUT*-complete or* MIN 2CNF DELETION*-complete or* NEAREST CODEWORD*-complete or* MIN HORN DELETION*-complete or or even deciding if the optimum is zero is* NP*-hard. Furthermore,*

(1) *If $\mathcal{F}$ is 0-valid or 1-valid or 2-monotone, then (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is in* PO*.*

(2) *Else if $\mathcal{F}$ is IHS-B then (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is* APX*-complete.*

(3) *Else if $\mathcal{F}$ is width-2 affine then (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is* MIN UNCUT*-complete.*

(4) *Else if $\mathcal{F}$ is 2CNF then (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is* MIN 2CNF DELETION*-complete.*

(5) *Else if $\mathcal{F}$ is affine then (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is* NEAREST CODEWORD*-complete.*

(6) *Else if $\mathcal{F}$ is weakly positive or weakly negative then (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is* MIN HORN DELETION*-complete.*

(7) *Else deciding if the optimum value of an instance of (*WEIGHTED*)* MIN CSP($\mathcal{F}$) *is zero is* NP*-complete.*

Theorem 2.14 (MIN ONES **classification)** *For any constraint set $\mathcal{F}$, the problem (*WEIGHTED*)* MIN ONES($\mathcal{F}$) *is either in* PO *or* APX*-complete or* NEAREST CODEWORD*-complete or* MIN HORN DELETION*-complete or* poly-APX*-complete or inapproximable to within any factor or not decidable. Furthermore,*

(1) *If $\mathcal{F}$ is 0-valid or weakly negative or width-2 affine, then (*WEIGHTED*)* MIN ONES($\mathcal{F}$) *is in* PO*.*

(2) *Else if $\mathcal{F}$ is 2CNF or IHS-B then (*WEIGHTED*)* MIN ONES($\mathcal{F}$) *is* APX*-complete.*

(3) *Else if $\mathcal{F}$ is affine then* MIN ONES($\mathcal{F}$) *is* NEAREST CODEWORD*-complete.*

(4) *Else if $\mathcal{F}$ is weakly positive then (*WEIGHTED*)* MIN ONES($\mathcal{F}$) *is* MIN HORN DELETION*-complete.*

(5) *Else if $\mathcal{F}$ is* 1*-valid then* Min Ones($\mathcal{F}$) *is* poly-APX*-complete and* Weighted Min Ones($\mathcal{F}$) *is decidable but hard to approximate to within any factor.*

(6) *Else finding any feasible solution to (*Weighted*)* Min Ones($\mathcal{F}$) *is* NP*-hard.*

2.6 Techniques

Two simple ideas play an important role in this paper. First is the notion of an *implementation* which shows how to use the constraints of a family $\mathcal{F}$ to enforce constraints of a different family $\mathcal{F}'$, thereby laying the groundwork of a reduction among problems. The notion of an implementation is inspired by the notion of gadgets formalized by Bellare et al. [8] who in our language define implementations for specific pairs of function families $(\mathcal{F}, \mathcal{F}')$. In this work we unify their definition, so as to make it work for arbitrary pairs of function families. This definition of implementation also finds applications in the work of Trevisan et al. [43] who, in our language, show uniform methods for searching for efficient implementations for pairs of function families $(\mathcal{F}, \mathcal{F}')$.

A second simple idea we exploit here is that of working with weighted versions of optimization problems. Even though our primary concerns were only the approximability of the unweighted versions of problems, many of our results use as intermediate steps the weighted versions of these problems. The weights allow us to manipulate problems more locally. However, simple and well-known ideas eventually allow us to get rid of the weights and thereby yielding hardness of the unweighted problem as well. As a side-effect we also show that the unweighted and weighted problems are equally hard to approximate in all the relevant optimization problems. This extends to minimization problems the results of Crescenzi et al. [15].

The definitions of implementations and weighted problems follows in Section 3. Section 4 shows some technical results showing how we exploit the fact that we have functions which don't exhibit some property. The results of this section play a crucial role in all the hardness results. This sets us up for the proofs of our main theorems. In Section 5 we show the containment results and hardness results for Max CSP. Similarly Sections 6, 7, and 8 deal with the classes Max Ones, Min CSP, and Min Ones, respectively.

3 Implementations

We now describe the main technique used in this paper to obtain hardness of approximation results. Suppose we want to show that for some constraint set $\mathcal{F}$, the problem Max CSP($\mathcal{F}$) is APX-hard. We will start with a problem that is known to be APX-hard, such as Max Cut, which turns out to be Max CSP({XOR}). We will then wish to reduce this problem to Max CSP($\mathcal{F}$). The main technique we use to do this is to "implement" the constraint XOR using constraints from the constraint set $\mathcal{F}$. We show how to formalize this notion next and then show how this translates to approximation preserving reductions.

Definition 3.1 (Implementation) *A collection of constraint applications $C_1, \ldots, C_m$ over a set of variables $\mathbf{x} = \{x_1, \ldots, x_p\}$ called* primary variables *and $\mathbf{y} = \{y_1, \ldots, y_q\}$ called* auxiliary variables*, is an α-*implementation *of a constraint $f(\mathbf{x})$ for a positive integer α if the following conditions are satisfied:*

(1) *For any assignment to $\mathbf{x}$ and $\mathbf{y}$ at most α constraints from $C_1, \ldots, C_m$ are satisfied.*

(2) *For any* $\mathbf{x}$ *such that* $f(\mathbf{x}) = 1$*, there exists* $\mathbf{y}$ *such that exactly* α *constraints are satisfied.*

(3) *For any* $\mathbf{x}, \mathbf{y}$ *such that* $f(\mathbf{x}) = 0$*, at most* $(\alpha - 1)$ *constraints are satisfied.*

Definition 3.2 (Strict/Perfect Implementations) *An* α*-implementation is a* strict α*-implementation if for every* $\mathbf{x}$ *such that* $f(\mathbf{x}) = 0$*, there exists* $\mathbf{y}$ *such that exactly* $(\alpha - 1)$ *constraints are satisfied. An* α*-implementation (not necessarily strict) is a* perfect *implementation, if* $\alpha = m$*.*

We say that a constraint set $\mathcal{F}$ (strictly / perfectly) implements a constraint f if there exists a (strict / perfect) α-implementation of f using constraints of $\mathcal{F}$ for some $\alpha < \infty$. We use the notation $\mathcal{F} \Longrightarrow_\alpha f$ to denote that $\mathcal{F}$ α-implements f, and $\mathcal{F} \Longrightarrow f$ to denote that $\mathcal{F}$ implements f. Similarly we use the notation $\mathcal{F} \overset{s/p}{\Longrightarrow} f$ to denote that $\mathcal{F}$ implements f strictly/perfectly. The above notation is also extended to allow the target to be a family of functions. For instance, $\mathcal{F} \Longrightarrow \mathcal{F}'$ denotes that $\mathcal{F}$ implements every function in $\mathcal{F}'$.

Remark: The definition of [8] defined (non-strict and non-perfect) implementations for specific choices of f and $\mathcal{F}$. For each choice they provided a separate definition. We unify their definitions into a single one. Furthermore as we will show later, the use of strictness and/or perfectness greatly enhance the power of implementations. These aspects are formalized for the first time here.

A constraint f 1-implements itself strictly and perfectly ($\{f\} \overset{s/p}{\Longrightarrow}_1 f$). Some more examples of strict and/or perfect implementations are given below.

Proposition 3.3 $\{\text{XOR}\} \overset{s/p}{\Longrightarrow}_2 \text{XNOR}$.

Proof: The constraints $\text{XOR}(x, z_{\text{AUX}})$ and $\text{XOR}(y, z_{\text{AUX}})$ perfectly and strictly implement the constraint $\text{XNOR}(x, y)$. □

Proposition 3.4 *If* $f(\mathbf{x}) = f_1(\mathbf{x}) \bigwedge \cdots \bigwedge f_k(\mathbf{x})$*, then* $\{f_1, \ldots, f_k\} \overset{p}{\Longrightarrow}_k f$*.*

Proof: The collection $\{f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})\}$ is a perfect (but not necessarily strict) k-implementation of $f(\mathbf{x})$. □

The following lemma shows that the implementations of constraints compose together, if they are strict or perfect.

Lemma 3.5 *If* $\mathcal{F}_a \overset{s}{\Longrightarrow} \mathcal{F}_b$ *and* $\mathcal{F}_b \overset{s}{\Longrightarrow} \mathcal{F}_c$*, then* $\mathcal{F}_a \overset{s}{\Longrightarrow} \mathcal{F}_c$*. An analogous result holds for perfect implementations also.*

Proof: It suffices to consider the case when $\mathcal{F}_c$ consists of a single function f. Furthermore, we observe that it suffices to prove the following simpler assertion (to prove the lemma): If $\mathcal{F} \overset{s}{\Longrightarrow} g$ and $\mathcal{F} \cup \{g\} \overset{s}{\Longrightarrow} f$ then $\mathcal{F} \overset{s}{\Longrightarrow} f$. To see that this suffices, let $\mathcal{F}_b = \{g_1, \ldots, g_l\}$. Define $\mathcal{F}_0 = \mathcal{F}_a$, $\mathcal{F}_i = \mathcal{F}_a \cup \{g_1, \ldots, g_i\}$. Note that by hypothesis we have $\mathcal{F}_l \overset{s}{\Longrightarrow} f$ and $\mathcal{F}_i \overset{s}{\Longrightarrow} g_{i+1}$ and $\mathcal{F}_l \overset{s}{\Longrightarrow} f$. The assertion above says that if $\mathcal{F}_{i+1} \overset{s}{\Longrightarrow} f$, then $\mathcal{F}_i \overset{s}{\Longrightarrow} f$. Thus by induction $\mathcal{F}_0 \overset{s}{\Longrightarrow} f$.

We now prove the assertion: If $\mathcal{F} \overset{s}{\Longrightarrow} g$ and $\mathcal{F} \cup \{g\} \overset{s}{\Longrightarrow} f$ then $\mathcal{F} \overset{s}{\Longrightarrow} f$. Let $C_1, \ldots, C_{m_1}$ be constraint applications from $\mathcal{F} \cup \{g\}$ on variables $\mathbf{x}, \mathbf{y}$ giving an α_1-implementation of $f(\mathbf{x})$ with $\mathbf{x}$ being the primary variables. Let $C'_1, \ldots, C'_{m_2}$ be constraint applications from $\mathcal{F}$ on variable set $\mathbf{x}', \mathbf{z}'$ yielding an α_2-implementation of $g(\mathbf{x}')$. Further let the first β constraints of $C_1, \ldots, C_{m_1}$ be applications of the constraints g.

We create a collection of $m_1 + \beta(m_2 - 1)$ constraints from $\mathcal{F}$ on a set of variables $\mathbf{x}, \mathbf{y}, \mathbf{z}'_1, \ldots, \mathbf{z}'_\beta$, where $\mathbf{x}$ and $\mathbf{y}$ are the original variables, and $\mathbf{z}'_{\mathbf{1}}, \ldots, \mathbf{z}'_\beta$ are new sets of disjoint auxiliary variables. (I.e., the vectors $\mathbf{z}'_i$ and $\mathbf{z}'_j$ do not share any variables, if $i \neq j$.)

The $m_1 + \beta(m_2 - 1)$ constraints introduced are as follows. We include the constraint applications $C_{\beta+1}, \ldots, C_{m_1}$ on variables $\mathbf{x}, \mathbf{y}$ and for every constraint application C_j, for $j \in \{1, \ldots, \beta\}$, on variables $\mathbf{v}_j$ (which is a subset of variables from $\mathbf{x}, \mathbf{y}$) we place the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$ on variable set $\mathbf{v}_j, \mathbf{z'}_j$ with $\mathbf{z'}_j$ being the auxiliary variables.

We now show that this collection of constraints satisfies properties (1)-(3) from Definition 3.1 with $\alpha = \alpha_1 + \beta(\alpha_2 - 1)$. Additionally we show that perfectness and/or strictness is preserved. We start with properties (1) and (3).

Consider any assignment to $\mathbf{x}$ satisfying f. Then any assignment to $\mathbf{y}$ satisfies at most α_1 constraints from the set $C_1, \ldots, C_{m_1}$. Let γ of these be from the set $C_1, \ldots, C_\beta$. Now for every $j \in \{1, \ldots, \beta\}$ any assignment to $\mathbf{z'}_j$ satisfies at most α_2 of the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$. Furthermore if the constraint C_j was not satisfied by the assignment to $\mathbf{x}, \mathbf{y}$, then at most $\alpha_2 - 1$ constraints are satisfied. Thus the total number of constraints satisfied by any assignment is at most $\gamma\alpha_2 + (\beta - \gamma)(\alpha_2 - 1) + (\alpha_1 - \gamma) = \alpha_1 + \beta(\alpha_2 - 1)$. This yields property (1). Property (3) is achieved similarly.

We now show that if the α_1- and α_2-implementations are perfect we get property (2) with perfectness. In this case for any assignment to $\mathbf{x}$ satisfying f, there exists an assignment to $\mathbf{y}$ satisfying $C_1, \ldots, C_{m_1}$. Furthermore for every $j \in \{1, \ldots, \beta\}$, there exists an assignments to $\mathbf{z'}_j$ satisfying all the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$. Thus there exists an assignment to $\mathbf{x}, \mathbf{y}, \mathbf{z'}_1, \ldots, \mathbf{z'}_\beta$ satisfying all $m_1 + \beta(m_2 - 1)$ constraints. This yields property (2) with perfectness.

Finally we consider the case when the α_1- and α_2-implementations are strict (but not necessarily perfect) and show that in this case also the collection of constraints above satisfies Property (2) with strictness. Given an assignment to $\mathbf{x}$ satisfying f there exists an assignment to $\mathbf{y}$ satisfying α_1 constraints from $C_1, \ldots, C_{m_1}$. Say this assignment satisfied γ clauses from the set $C_1, \ldots, C_\beta$ and $\alpha_1 - \gamma$ constraints from the set $C_{\beta+1}, \ldots, C_{m_1}$. Then for every $j \in \{1, \ldots, \beta\}$ such that the clauses C_j is satisfied by this assignment to $\mathbf{x}, \mathbf{y}$, there exists an assignment to $\mathbf{z'}_j$ satisfying α_2 clauses from the set $C'_{1,j}, \ldots, C'_{m_2,j}$. Furthermore, for the remaining values of $j \in \{1, \ldots, \beta\}$ there exists an assignment to the variables $\mathbf{z'}_j$ satisfying $\alpha_2 - 1$ of the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$ (here we are using the strictness of the α_2 implementations). This setting to $\mathbf{y}, \mathbf{z'}_1, \ldots, \mathbf{z'}_\beta$ satisfies $\gamma\alpha_2 + (\beta - \gamma)(\alpha_2 - 1) + \alpha_1 - \gamma = \alpha_1 + \beta(\alpha_2 - 1)$ of the m constraints. This yields Property (2). A similar analysis can be used to show the strictness. □

Next we show a simple monotonicity property of implementations.

Lemma 3.6 *For integers α, α' with $\alpha \leq \alpha'$, if $\mathcal{F} \Longrightarrow_\alpha f$ then $\mathcal{F} \Longrightarrow_{\alpha'} f$. Furthermore, strictness and perfectness are preserved under this transformation.*

Proof: Let constraint applications $C_1, \ldots, C_m$ from $\mathcal{F}$ on $\mathbf{x}, \mathbf{y}$ form an α-implementation of $f(\mathbf{x})$. Let g be any constraint from $\mathcal{F}$ and let k be the arity of g. Let $C_{m+1}, \ldots, C_{m+\alpha'-\alpha}$ be $\alpha' - \alpha$ applications of the constraint g on new variables $\mathbf{z} = \{z_1, \ldots, z_k\}$. Then the collection of constraints $C_1, \ldots, C_{m+\alpha'-\alpha}$ on variable set $\mathbf{x}, \mathbf{y}, \mathbf{z}$ form an α'-implementation of f. Furthermore the transformation preserves strictness and perfectness. □

3.1 Reduction from strict implementations

Here we show how strict implementations are useful in establishing AP-reducibility among Max CSP problems. But first we need a simple statement about the approximability of Max CSP problems.

Proposition 3.7 ([39]) *For every constraint family $\mathcal{F}$ there exists a constant k such that given any instance $\mathcal{I}$ of* Weighted Max CSP($\mathcal{F}$) *with constraints of total weight W a solution satisfying constraints of weight W/k can be found in polynomial time.*

Proof: The proposition follows from the proof of Theorem 1 in [39] which shows the above for every MAX SNP problem. (Note, in particular, that a random assignment satisfies a constant fraction of Weighted Max CSP($\mathcal{F}$) instance; and such an assignment can be found deterministically by using the method of conditional probabilities.) □

Lemma 3.8 *If $\mathcal{F}' \overset{s}{\Longrightarrow} \mathcal{F}$ then* Max CSP($\mathcal{F}$) $\leq_{\mathrm{AP}}$ Max CSP($\mathcal{F}'$).

Proof: The reduction uses Proposition 3.7 above. Let β a constant such that given an instance $\mathcal{I}$ of Max CSP($\mathcal{F}$) with m constraints an assignment satisfying $\frac{m}{\beta}$ constraints can be found in polynomial time.

Recall that we need to show polynomial time computable functions F and G such that F maps an instance $\mathcal{I}$ of Max CSP($\mathcal{F}$) to an instance of Max CSP($\mathcal{F}'$), and G maps a solution to $F(\mathcal{I})$ back to a solution of $\mathcal{I}$.

Given an instance $\mathcal{I}$ on n variables and m constraints, the mapping F simply replaces every constraint in $\mathcal{I}$ (which belongs to $\mathcal{F}$) with a strict α-implementation using constraints of $\mathcal{F}'$, for some constant α. (Notice that by Lemma 3.6 some such α does exist.) The mapping retains the original n variables of $\mathcal{I}$ as primary variables and uses m independent copies of the auxiliary variables; one independent copy for every constraint in $\mathcal{I}$.

Let $\langle \mathbf{x}, \mathbf{y} \rangle$ be a r-approximate solution to the instance $F(\mathcal{I})$, where $\mathbf{x}$ denotes the original variables of $\mathcal{I}$ and $\mathbf{y}$ denote the auxiliary variables introduced by F. The mapping G uses two possible solutions and takes the better of the two: The first solution is x; and the second solution x' is the solution which satisfies at least m/β of the constraints in $\mathcal{I}$. G outputs the solution which satisfies more constraints.

We now show that a r-approximate solution leads to an r'-approximate solution where $r' \leq 1+\gamma(r-1)$ for some constant γ. Let $\textsc{opt}$ denote the value of the optimum to $\mathcal{I}$. Then the optimum of $F(\mathcal{I})$ is exactly $\textsc{opt} + m(\alpha-1)$. This computation uses the fact that for every satisfied constraint in the optimal assignment to $\mathcal{I}$, we can satisfy α constraints of its implementation by choosing the auxiliary variables appropriately (from Properties (1) and (2) of Definition 3.1); and for every unsatisfied constraint exactly $\alpha - 1$ constraints of its implementation can be satisfied (from Property (3) and strictness of the implementation). Thus the solution $\langle \mathbf{x}, \mathbf{y} \rangle$ satisfies at least $\frac{1}{r}(\textsc{opt} + m(\alpha-1))$ constraints of $F(\mathcal{I})$. Thus $\mathbf{x}$ satisfies at least $\frac{1}{r}(\textsc{opt}+m(\alpha-1)) - m(\alpha-1)$ constraints in $\mathcal{I}$. (Here we use Properties (1) and (3) of Definition 3.1 to see that there must be at least $\frac{1}{r}(\textsc{opt} + m(\alpha-1)) - m(\alpha-1)$ constraints of $\mathcal{I}$ in whose implementations exactly α constraints must be satisfied.) Thus the solution output by G satisfies at least

$$\max\{\frac{1}{r}(\textsc{opt} + m(\alpha-1)) - m(\alpha-1), \frac{m}{\beta}\}$$

constraints. Using the fact that $\max\{a,b\} \geq \lambda a + (1-\lambda)b$ for any $\lambda \in [0,1]$ and using $\lambda = \frac{r}{r+\beta(\alpha-1)(r-1)}$, we lower bound the above expression by

$$\frac{\textsc{opt}}{r + \beta(\alpha-1)(r-1)}.$$

Thus

$$r' \le \frac{\text{OPT}}{\text{OPT}/(r+\beta(\alpha-1)(r-1))} = r+\beta(\alpha-1)(r-1) = 1+(\beta(\alpha-1)+1)(r-1).$$

Thus we find that G maps r-approximate solutions of $F(\mathcal{I})$ to $(1+\gamma(r-1))$-approximate solutions to $\mathcal{I}$ for $\gamma = \beta(\alpha-1)+1 < \infty$ as required. □

3.2 Reductions from perfect implementations

We now show how to use perfect implementations to get reductions. Specifically we obtain reductions among Weighted Max Ones, Weighted Min Ones and Min CSP problems.

Lemma 3.9 *If* $\mathcal{F}' \overset{p}{\Longrightarrow} \mathcal{F}$ *then* Weighted Max Ones($\mathcal{F}$) *(*Weighted Min Ones($\mathcal{F}$)*) is AP-reducible to* Weighted Max Ones($\mathcal{F}'$) *(resp.* Weighted Min Ones($\mathcal{F}'$)*).*

Proof: Again we need to show polynomial time computable functions F and G such that F maps an instance $\mathcal{I}$ of Weighted Max Ones($\mathcal{F}$) (Weighted Min Ones($\mathcal{F}$)) to an instance of Weighted Max Ones($\mathcal{F}'$) (Weighted Min Ones($\mathcal{F}$)), and G maps a solution to $F(\mathcal{I})$ back to a solution of $\mathcal{I}$.

Given an instance $\mathcal{I}$ on n variables and m constraints, the mapping F simply replaces every constraint in $\mathcal{I}$ (which belongs to $\mathcal{F}$) with a perfect α-implementation using constraints of $\mathcal{F}'$, for some constant α. (Notice that by Lemma 3.6 some such α does exist.) The mapping retains the original n variables of $\mathcal{I}$ as primary variables and uses m independent copies of the auxiliary variables; one independent copy for every constraint in $\mathcal{I}$. Further, $F(\mathcal{I})$ retains the weight of the primary variables from $\mathcal{I}$ and associates a weight of zero to all the newly created auxiliary variables. Given a solution to $F(\mathcal{I})$, the mapping G is simply the projection of the solution back to the primary variables. It is clear that every feasible solution to $\mathcal{I}$ can be extended into a feasible solution to $F(\mathcal{I})$ such that $\text{OPT}(\mathcal{I}) = \text{OPT}(F(\mathcal{I}))$. Furthermore, the mapping G maps feasible solutions to $F(\mathcal{I})$ into feasible solutions to $\mathcal{I}$ with the same obective. (This is where the perfectness of the implementations is being used.) Thus the optimum of $F(\mathcal{I})$ equals the value of the optimum of $\mathcal{I}$ and given an r-approximate solution to $F(\mathcal{I})$, the mapping G yields an r-approximate solution to $\mathcal{I}$. □

Lemma 3.10 *If* $\mathcal{F}' \overset{p}{\Longrightarrow} \mathcal{F}$ *then* Min CSP($\mathcal{F}$) $\le_{\text{A}}$ Min CSP($\mathcal{F}'$).

Proof: Let α be large enough so that any constraint from $\mathcal{F}$ has a perfect α-implementation using constraints from $\mathcal{F}'$. Let $\mathcal{I}$ be an instance of Min CSP($\mathcal{F}$) and let $\mathcal{I}'$ be the instance of Min CSP($\mathcal{F}'$) obtained by replacing each constraint of $\mathcal{I}$ with the respective α-implementation. Once again each implementation uses the original set of variables for its primary variables and uses its own independent copy of the auxiliary variables. Note that the optimum of $\mathcal{I}'$ may be as high as αo if o is the optimum of $\mathcal{I}$ (since the implementations are not strict). It is easy to check that any assigment for $\mathcal{I}'$ of cost V yields an assigment for $\mathcal{I}$ whose cost is between V/α and V. In particular, if the solution is an r-approximate solution to $\mathcal{I}'$ then, $V \ge \frac{o}{\alpha r}$ and thus it induces a solution that is at least an (αr)-approximate solution to $\mathcal{I}$. (Note that if the implementations were strict, we would have obtained an AP-reduction by the above.) □

3.3 Weighted vs. unweighted problems

Lemma 3.9 crucially depends on its ability to work with weighted problems to obtain reductions. The following lemma shows that in most cases showing hardness for weighted problems is sufficient. Specifically it shows that as long as a problem is weakly approximable, its weighted and unweighted versions are equivalent. The result uses a similar result from Crescenzi et al. [15] who prove that for a certain class of problems in poly-APX that they term "nice", weighted problems AP-reduce to problems with polynomially-bounded integral weights. (We include a sketch of their proof, specialized to our case for the sake of completeness.) Using this result we scale all weights down to small integers and then simulate the small integral weights by replication of clauses and/or variables. (We note that the little-oh slackness in the definition of AP-reduction is exploited in this step.)

Lemma 3.11 *For every family $\mathcal{F}$, if* WEIGHTED MAX ONES($\mathcal{F}$) *is in* poly-APX*, then* WEIGHTED MAX ONES($\mathcal{F}$) *AP-reduces to* MAX ONES($\mathcal{F}$)*. Analogous results hold for* MIN CSP($\mathcal{F}$)*,* MAX CSP($\mathcal{F}$) *and* MIN ONES($\mathcal{F}$)*.*

Proof: Fix a family $\mathcal{F}$. We first reduce WEIGHTED MAX ONES($\mathcal{F}$) to WEIGHTED MAX ONES($\mathcal{F}$) restricted to instances with polynomially bounded positive integer weights, provided WEIGHTED MAX ONES($\mathcal{F}$) is in poly-APX. This step uses a scaling idea as in [15, Theorem 4]. Essentially the same proof also works for the cases of WEIGHTED MAX CSP($\mathcal{F}$), WEIGHTED MIN CSP($\mathcal{F}$) or WEIGHTED MIN ONES($\mathcal{F}$). Given an instance $\mathcal{I} = (\mathbf{x}, \mathbf{C}, \mathbf{w})$ of WEIGHTED MAX ONES($\mathcal{F}$), we will define a new vector of weights $\mathbf{w}'$ and use this to define a new instance $\mathcal{I}' = (\mathbf{x}, \mathbf{C}, \mathbf{w}')$ of WEIGHTED MAX ONES($\mathcal{F}$) with polynomially bounded weights. Let A be a $p(n)$-approximation algorithm for WEIGHTED MAX ONES($\mathcal{F}$); and let t be the value of the solution returned by A on $\mathcal{I}$. We let $N = n^2(p(n))^2 + np(n)$, and let $w_i'' = \left\lfloor \frac{w_i \cdot N}{t} \right\rfloor + 1$, and finally let $w_i' = \min\{w_i'', N \cdot p(n) + 1\}$. It is clear that the weights w_i' are polynomially bounded. Further note that if $w_i' < w_i''$ then no feasible solution to $\mathcal{I}$ (or $\mathcal{I}'$) can have x_i set to 1, since any such solution would have value at least $w_i > t \cdot p(n)$, contradicting the assumption that A is a $p(n)$-approximation algorithm. Thus, in particular, we have $\text{OPT}(\mathcal{I}') \geq (N/t) \cdot \text{OPT}(\mathcal{I})$. Given an r-approximate solution s' to $\mathcal{I}'$ we return the better of the solutions s' and the solution returned A as the solution to $\mathcal{I}$. It is clear that if $r \geq p(n)$, then the returned solution is still an r-approximate solution. Below we see that an r-approximate solution to $\mathcal{I}'$, with $r \leq p(n)$, is also a $(r + 1/n)$-approximate solution to $\mathcal{I}$ of value at least

$$\begin{aligned}
(t/N) \cdot (\text{OPT}(\mathcal{I}')/r) - n) &\geq \text{OPT}(\mathcal{I})/r - (nt/N) \\
&\geq \text{OPT}(\mathcal{I})/r - (n \cdot \text{OPT}(\mathcal{I})/N) \\
&\geq \text{OPT}(\mathcal{I})(\frac{1}{r} - \frac{1}{nr^2 + r}) \\
&= \text{OPT}(\mathcal{I})/(r + 1/n).
\end{aligned}$$

This concludes the first step of the reduction. In the next step we give an AP-reduction from the class of problems with polynomially bounded weights to the unweighted case.

We start with the case of WEIGHTED MAX CSP($\mathcal{F}$) first. Given an instance of WEIGHTED MAX CSP($\mathcal{F}$) on variables $x_1, \ldots, x_n$, constraints $C_1, \ldots, C_m$ and polynomially bounded integer weights $w_1, \ldots, w_m$; we reduce it to the unweighted case by replication of constraints. Thus the reduced instance has variables $x_1, \ldots, x_n$ and constraint $\{\{C_i^j\}_{j=1}^{w_i}\}_{i=1}^m$ where constraint $C_i^j = C_i$. It is clear that the reduced instance is essentially the same as the instance we started with. Similarly we reduce Weighted Min CSP($\mathcal{F}$) to Min CSP($\mathcal{F}$).

Given an instance $\mathcal{I}$ of Weighted Max Ones($\mathcal{F}$) on variables $x_1, \ldots, x_n$, constraints $C_1, \ldots, C_m$ and weights $w_1, \ldots, w_n$; we create an instance $\mathcal{I}'$ of

Max Ones($\mathcal{F}$) on variables $\{\{y_i^j\}_{j=1}^{w_i}\}_{i=1}^{n}$. For every constraint C_j of $\mathcal{I}$ of the form $f(x_{i_1}, \ldots, x_{i_k})$, and for every $j \in \{1, \ldots, k\}$ and $n_j \in \{1, \ldots, w_{i_j}\}$ we impose the constraints $f(y_{i_1}^{n_1}, \ldots, y_{i_k}^{n_k})$. We now claim that the reduced instance is essentially equivalent to the instance we started with. To see this, notice that given any feasible solution $\mathbf{y}$ to the instance $\mathcal{I}'$, we may convert it to another feasible solution $\mathbf{y}'$ in which, for every i, all the variables $\{(\mathbf{y}')_i^j | j = 1, \ldots, w_i\}$ have the same assignment, by setting $(\mathbf{y}')_i^j$ to 1 if any of the variables $y_i^{j'}$, $j' = 1, \ldots, w_i$ is set to 1. Notice that this preserves feasibility; and only increases the contribution to the objective function. The assignment $\mathbf{y}'$ now induces an assignment to $\mathbf{x}$ with the same value of the objective function. Thus the reduced instance is essentially equivalent to the original one. This concludes the reduction from Weighted Max Ones($\mathcal{F}$) to Max Ones($\mathcal{F}$). The reduction from Weighted Min Ones($\mathcal{F}$) to Min Ones($\mathcal{F}$) is similar. □

4 Characterizations: New and Old

In this section we characterize some of the properties of functions that we study. Most of the properties are defined so as to describe how a function behaves if it exhibits the property. For the hardness results however we need to see how to exploit the fact that a function does not satisfy some given property. For this we would like to see some simple witness to the fact that the function does not have a given property. As an example consider the affineness property. If a function is affine, it is easy to see how to use this property. What will be important to us is whether there exists a simple witness to the fact that a function f is not affine. Schaefer [42] provides such a characterization: If a function is not affine, then there exist assignments s_1, s_2 and s_3 that satisfy f such that $s_1 \oplus s_2 \oplus s_3$ does not satisfy f. This is exploited by Schaefer in his classification theorem (and by us, in our classifications). In this section, we describe other such characterizations and the implementations that are obtained from them. First we introduce some more definitions and notations that we will be used in the rest of the paper.

4.1 Definitions and Notations

For $s \in \{0,1\}^k$, we let $\bar{s} \in \{0,1\}^k$ denote the bit-wise complement of s. For a constraint f of arity k, let f^- be the constraint $f^-(s) = f(\bar{s})$. For a constraint family $\mathcal{F}$, let $\mathcal{F}^- = \{f^- : f \in \mathcal{F}\}$. For $s_1, s_2 \in \{0,1\}^k$, $s_1 \oplus s_2$ denotes the bitwise exclusive-or of the assignments s_1 and s_2. For $s \in \{0,1\}^k$, $Z(s)$ denotes the subset of indices $i \in [k]$ where s is zero and $O(s)$ denotes the subset of indices where s in one.

For a constraint f of arity k, $S \subseteq [k]$ and $b \in \{0,1\}$, $f|_{(S,b)}$ is the constraint of arity $k' = k - |S|$ defined as follows: For variables $x_{i_1}, \ldots, x_{i_{k'}}$, where $\{i_1, \ldots, i_{k'}\} = [k] - S$, we define $f|_{(S,b)}(x_{i_1}, \ldots, x_{i_{k'}}) = f(x_1, \ldots, x_k)$ where $x_i = b$ for $i \in S$. We will sometimes use the notation $f|_{(i,b)}$ to denote the function $f|_{(\{i\},b)}$. For a constraint family $\mathcal{F}$, the family $\mathcal{F}|_0$ is the family $\{f|_{S,0} | f \in \mathcal{F}, S \subseteq [\text{arity}(f)]\}$. The family $\mathcal{F}|_1$ is defined analogously. The family $\mathcal{F}|_{0,1}$ is the family $(\mathcal{F}|_0)|_1$ (or equivalently the family $(\mathcal{F}|_1)|_0$).

Definition 4.1 (C-closed) *A constraint f is C-closed (complementation-closed) if for every assignment s, $f(s) = f(\bar{s})$.*

Definition 4.2 (Existential zero/existential one) *A constraint f is an existential zero constraint if $f(\mathbf{0}) = 1$ and $f(\mathbf{1}) = 0$. A constraint f is an existential one constraint if $f(\mathbf{0}) = 0$ and $f(\mathbf{1}) = 1$.*

The terminology above is motivated by the fact that an existential zero constraint application $f(x_1, \ldots, x_k)$ forces at least one of the variables to be zero (while an all zero assignment definitely satisfies the application).

Every constraint f can be expressed as the conjunction of disjuncts. This representation of a function is referred to as the conjunctive normal form (CNF) representation of f. Alternately, a function can also be represented as a disjunction of conjuncts and this representation is called the disjunctive normal form (DNF) representation.

A partial setting to the variables of f that fixes the value of f to 1 is called a *term* of f. A partial setting that fixes f to 0 is called a *clause* of f. We refer to the terms and clauses in a functional form: I.e., we say $\mathrm{OR}_{3,1}(x_1, x_2, x_3) = x_1 \bigvee x_2 \bigvee \neg x_3$ is a clause of $f(x_1, \ldots, x_p)$ if setting $x_1 = x_2 = 0$ and $x_3 = 1$ fixes f to being 0. Similarly we use the $\mathrm{AND}_{i,j}$ to denote the terms. Notice that a DNF (CNF) representation of f can be obtained by expressing as the conjunction (disjunction) of its terms (clauses).

Definition 4.3 (Minterm/Maxterm) *A partial setting to a subset of the variables of f is a minterm if it is a term of f and no restriction of the setting to any strict subset of the variables fixes the value of f. Analogously a clause of f is a maxterm if it is a minimal setting to the variables of f so as to fix its value to* 0*.*

As in the case of terms and clauses, we represent minterms and maxterms functionally, i.e., using $\mathrm{OR}_{i,j}$ and $\mathrm{AND}_{i,j}$.

Definition 4.4 (Basis) *A constraint family $\mathcal{F}'$ is a basis for a constraint family $\mathcal{F}$ if any constraint of $\mathcal{F}$ can be expressed as a conjunction of constraints drawn from $\mathcal{F}'$.*

Thus, for example, the basis for affine constraints is the set $\{\mathrm{XOR}_p | p \geq 1\} \cup \{\mathrm{XNOR}_p | p \geq 1\}$. The basis for width-2 affine constraints is the set $\mathcal{F} = \{\mathrm{XOR}, \mathrm{XNOR}, T, F\}$, and the basis for 2CNF constraints is the set $\mathcal{F} = \{\mathrm{OR}_{2,0}, \mathrm{OR}_{2,1}, \mathrm{OR}_{2,2}, T, F\}$. The definition of a basis is motivated by the fact that if $\mathcal{F}'$ is a basis for $\mathcal{F}$, then $\mathcal{F}'$ can perfectly implement every function in $\mathcal{F}$ (see Proposition 3.4).

4.2 0-validity and 1-validity

The characterization of 0-valid and 1-valid functions is obvious. We now show what can be implemented with functions that are not 0-valid and not 1-valid.

Lemma 4.5 *Let f be a non-trivial constraint which is C-closed and is not 0-valid (or equivalently not 1-valid)*[5]*. Then $\{f\} \overset{s/p}{\Longrightarrow} \mathrm{XOR}$.*

[5] Notice that C-closedness implies that f is 0-valid if and only if it is 1-valid.

Proof: Let k denote the arity of f and let k_0 and k_1 respectively denote the maximum number of 0's and 1's in any satisfying assignment for f; clearly $k_0 = k_1$. Now let $S_x = \{x_1, \ldots, x_{3k}\}$ and $S_y = \{y_1, \ldots, y_{3k}\}$ be two disjoint sets of $3k$ variables each. In the first phase of the proof, we place a large number of constraints on the variables of S_x and S_y that ends up implementing, perfectly but not necessarily strictly, the constraints $\mathrm{XOR}(x_i, y_j)$, for every i and j. In the second phase we will introduce two new variables x and y and augment the constraints so as to implement the constraint $\mathrm{XOR}(x, y)$ perfectly and strictly.

We start by placing the constraint f on a large collection of inputs as follows: For every satisfying assignment s, we place $\binom{3k}{i}\binom{3k}{k-i}$ constraints on the variable set $S_x \cup S_y$ such that every i-variable subset of S_x appears in place of 0's in s and every $(k - i)$ variable subset of S_y appears in place of 1's in the assignment s, where i denotes the number of 0's in s. Let this collection of constraints be denoted by $\mathcal{I}$. We will first show that $\mathcal{I}$ gives a perfect (but possibly non-strict) implementation of the constraint $\mathrm{XOR}(x_i, y_j)$.

Clearly, any solution which assigns identical values to all variables in S_x and the complementary value to all variables in S_y, satisfies all the constraints in $\mathcal{I}$. We will show the converse, i.e., every assignment satisfying all the above constraints assigns identical values to all variables in S_x and the complementary value to every variable in S_y.

Fix any assignment satisfying all the constraints and let Z and O respectively denote the set of variables set to zero and one respectively. We claim that any solution which satisfies all the constraints must satisfy either $Z = S_x$ and $O = S_y$ or $Z = S_y$ and $O = S_x$.

Note first that at least one of the conditions $|S_x \cap Z| \geq k$ or $|S_x \cap O| \geq k$ must hold. Consider the case where $|S_x \cap Z| \geq k$. In this case, we will show that $S_x = Z$ and $S_y = O$. (A similar argument for the other case will show $S_x = O$ and $S_y = Z$.)

- First we claim that $|S_y \cap Z| < k$ and thus $|S_y \cap O| > 2k$. Assume for contradiction that $|S_y \cap Z| \geq k$. Then there exists a constraint application in $\mathcal{I}$ with all its input variables coming from the sets $S_x \cap Z$ and $S_y \cap Z$. By definition of Z all these variables are set to zero and hence this constraint application is unsatisfied (by the 0-validity of f).
- Next we claim that every variable of S_x is set to 0: Assume otherwise and, w.l.o.g., let x_1 be set to 1. Let s be an assignment with minimal number of 0's. Assume w.l.o.g. that $s = 0^{k_0}1^{k-k_0}$. W.l.o.g., let $y_1, \ldots, y_{2k}$ be set to one. (We know $2k$ such variables exist since $|S_y \cap O| > 2k$.) By our choice of constraint applications, $f(x_1, \ldots, x_{k_0}, y_1, \ldots, y_{k-k_0})$ is one of the constraint applications. But at most $k_0 - 1$ variables of this constraint are set to 0 and thus this application can not be satisfied.
- Finally, similar to the above step, we can show that every variable in S_y is set to 1.

Thus we have shown that if $|S_x \cap Z| \geq k$, then $S_x = Z$ and $S_y = O$. The other case is similar, and this concludes the first phase.

We next augment the collection of constraints above as follows. Consider a least Hamming weight satisfying assignment s for f. Without loss of generality, we assume that $s = 10^{k-k_1-1}1^{k_1}$. We add the constraints $f(x, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$ and $f(y, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$. We now argue that the resulting collection of constraints yields a perfect and strict implementation of the constraint $\mathrm{XOR}(x, y)$.

Clearly $s' = 0^{k-k_1}1^{k_1}$ is not a satisfying assignment (since it has smaller Hamming weight than s). Since f is C-closed, we have the following situation :

$$\begin{array}{ccccc} & & & & f() \\ s' & 0 & \overbrace{00...0}^{k-k_1-1} & \overbrace{11...1}^{k_1} & 0 \\ s & 1 & 00...0 & 11...1 & 1 \\ \bar{s} & 0 & 11...1 & 00...0 & 1 \\ \bar{s'} & 1 & 11...1 & 00...0 & 0 \end{array}$$

If $x = 1$, then to satisfy the first of the two constraints (in addition to all the earlier constraints) above, we must have $Z = S_x$, $O = S_y$ and thus must have $y = 0$. Similarly if $x = 0$ then we must have $O = S_x$, $Z = S_y$ and $y = 1$. Thus the given constraints do form a perfect implementation of XOR(x, y). Finally if $x = y$, then the setting $O = S_x$ and $Z = S_y$ satisfies all constraints except one (which is one of the last two additional constraints). Thus the implementation satisfies the strictness property as well. □

Lemma 4.6 *Let f_0, f_1 and g be non-trivial constraints, possibly identical, which are not 0-valid, not 1-valid, and not C-closed, respectively. Then $\{f_0, f_1, g\} \overset{s/p}{\Longrightarrow} \{T, F\}$.*

Proof: We will only describe the implementation of constraint $T(\cdot)$; the implementation for the constraint $F(\cdot)$ is identical.

Assume, for simplicity, that all the three functions f_0, f_1 and g are of arity k. We use an implementation similar to the one used in the proof of Lemma 4.5. To implement $T(x)$, we use a set of $6k$ auxiliary variables $S_x = \{x_1, \ldots, x_{3k}\}$ and $S_y = \{y_1, \ldots, y_{3k}\}$. For each $h \in \{f_0, f_1, g\}$, for each satisfying assignment s of h, if j is the number of 0's in s we place the $\binom{3k}{j}\binom{3k}{k-j}$ constraints h with all possible subsets of S_x appearing in the indices in $Z(s)$ and all possible subsets of S_y appearing in $O(s)$. Finally we introduce one constraint involving the primary variable x. Let s be the satisfying assignment of minimum Hamming weight which satisfies f_0. Notice that s must include at least one 1. Assume, without loss of generality that $s = 10^{k-k_1-1}1^{k_1}$. Then we introduce the constraint application $f_0(x, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$.

It is clear that by setting all variables in S_x to 0 and all variables in S_y to 1 we get an assignment that satisfies all constraints except possibly the last constraint (which involves x). Furthermore the last constraint is satisfied if and only if $x = 1$. Thus, to prove the lemma, it suffices to show that any solution which satisfies all the constraints above must set x to 1, all variables in S_x to 0 and all variables in S_y to 1.

Fix an assignment satisfying all the constraints. Let O be the set of variables in $S_x \cup S_y$ set to one and Z be the set of variables set to zero. We need to show that $S_x \cap O = \emptyset$ and we do so in stages.

- First, we consider the possibility $|S_x \cap O| \geq k$. We consider two cases.
  - Case: $|S_y \cap Z| \geq k$: Consider a satisfying assignment s such that $g(\bar{s}) = 0$. Such an assignment must exist since g is not C-closed. Note that the constraint applications include at least one where g is applied to variables where the positions corresponding to $O(s)$ come from $S_y \cap Z$ and positions corresponding to $Z(s)$ come from $S_x \cap O$. But this constraint is not satisfied by the assignment (since $g(\bar{s}) = 0$).

- Case: $|S_y \cap O| > 2k$: Let s_1 be a satisfying assignment for f_1. Note that the application of the constraint f_1 with the positions corresponding to $O(s)$ coming from $S_y \cap O$ and the positions corresponding to $Z(s)$ coming from $S_x \cap O$ is one of the constraints imposed above, and is not satisfied (since f_1 is not 1-valid).

Thus in either case, we find a constraint that is not satisfied and thus this possibility ($|S_x \cap O| \geq k$) can not occur. Thus we conclude $|S_x \cap O| < k$.

- From the above, we have $|S_x \cap Z| > 2k$. If $|S_y \cap Z| \geq k$, then we can find an application of the constraint f_0 to the variables in the set Z, that will not be satisfied. Thus we have $|S_y \cap Z| < k$ and thus $|S_y \cap O| > 2k$. This can now be used to conclude that $S_y \cap Z = \phi$ as follows. Consider a satisfying assignment with smallest number of ones. The number of ones in such an assignment is positive since f_0 is not 0-valid. If we consider all the constraints corresponding to this assignment with inputs from S_y and $S_x \cap Z$ only, it is easy to see that there will be at least one unsatisfied constraint if $S_y \cap Z \neq \phi$. Hence each variable in S_y is set to one in this case. Finally, using the constraints on the constraint f_1 which is not 1-valid, it is easy to conclude that in fact $Z = S_x$.

Having concluded that $S_x = Z$ and $S_y = O$, it is easy to see that the constraint $f_0(x, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$ is satisfied only if $x = 1$. Thus the set of constraints imposed above yields a strict and perfect implementation of $T(\cdot)$. The constraint $F(\cdot)$ can be implemented in an analogous manner. □

For the CSP classes, it suffices to consider the case when $\mathcal{F}$ is neither 0-valid nor 1-valid. For the Max Ones and Min Ones classes we also need to consider the case when $\mathcal{F}$ only fails to have one of these two properties. So keeping these classes in mind we prove the following lemma, which shows how to obtain a weak version of T and F in these cases.

Lemma 4.7 *If $\mathcal{F}$ is not C-closed and not 1-valid, then $\mathcal{F} \overset{s/p}{\Longrightarrow} f$ for some existential zero constraint f_0. Analogously, if $\mathcal{F}$ is not C-closed and not 0-valid, then $\mathcal{F} \overset{s/p}{\Longrightarrow} f_1$ for some existential one constraint f_1.*

Proof: We only prove the first part of the lemma. The second part is similar.

The proof reduces to two simple sub-cases. Let $f \in \mathcal{F}$ be a constraint that is not 1-valid. If f is 0-valid, then we are done since f is an existential zero constraint. If f is not 0-valid, then $\mathcal{F}$ has a non-C-closed function, a non 0-valid function and a non-1-valid function, and hence by Lemma 4.6, $\mathcal{F}$ perfectly and strictly implements F which is an existential zero function. □

4.3 2-monotone functions

Definition 4.8 (0/1-term) *A set $V \subseteq \{1, \ldots, k\}$ is a* 0-term (1-term) *for a k-ary constraint f if every assignment s with $Z(s) \supseteq V$ (resp. $O(s) \supseteq V$) is a satisfying assignment for f.*

The choice of the name reflects the fact that a 0-term is a term consisting of all negated variables (or variables set to 0) and a 1-term consists of all positive variables.

Lemma 4.9 *A constraint f is a 2-monotone constraint if and only if all the following conditions are satisfied:*

(a) for every satisfying assignment s of f either $Z(s)$ is a 0-term or $O(s)$ is a 1-term.

(b) if V_1 and V_2 are 1-terms for f, then $V_1 \cap V_2$ is a 1-term, and

(c) if V_1 and V_2 are 0-terms for f, then $V_1 \cap V_2$ is also a 0-term.

Proof: Recall that a 2-monotone constraint is one that can be expressed as a disjunction of two terms. Every satisfying assignment must satisfy one of the two terms and this gives Property (a). Properties (b) and (c) are obtained from the fact that the constraint has at most one term with all positive literals and at most one term with all negative literals.

Conversely consider a constraint f which satisfies properties (a)-(c). Let $s_1, \ldots, s_l$ be the satisfying assignments of f such that $Z(s_i)$ is a 0-term, for $i \in \{1, \ldots, l\}$. Let $t_1 \ldots, t_k$ be the satisfying assignments of f such that $O(t_j)$ is a 1-term, for $j \in \{1, \ldots, k\}$. Then $Z = \cap_i Z(s_i)$ is a 0-term and $O = \cap_j O(t_j)$ is a 1-term for f respectively (using (b) and (c)) and together they cover all satisfying assignments of f. Thus $f(\mathbf{x}) = (\wedge_{i \in Z} \neg x_i) \vee (\wedge_{j \in O} x_j)$, which is 2-monotone. □

We now use the characterization above to prove, in Lemma 4.11, that if a function f is not 2-monotone, then the family $\{f, T, F\}$ implements the function XOR. We first prove a simple lemma which shows implementations of XOR by some specific constraint families. This will be used in Lemma 4.11.

Lemma 4.10 *1.* $\{\mathrm{AND}_{2,1}\} \overset{s}{\Longrightarrow} \mathrm{XOR}$.

2. For every $p \geq 2$, *we have* $\{f_p, T, F\} \overset{s/p}{\Longrightarrow} \mathrm{XOR}$, *where* $f_p(x_1, \ldots, x_p) = \mathrm{OR}_p(x_1, \ldots, x_p) \bigwedge \mathrm{NAND}_p(x_1, \ldots, x_p)$.

3. For every $p \geq 2$, *we have* $\{\mathrm{NAND}_p, T, F\} \overset{s}{\Longrightarrow} \mathrm{XOR}$.

Proof: For Part (1) we observe that the constraints $\{\mathrm{AND}_{2,1}(x_1, x_2), \mathrm{AND}_{2,1}(x_2, x_1)\}$ provide a strict (but not perfect) 1-implementation of $\mathrm{XOR}(x_1, x_2)$.

For Part (2) notice that the claim is trivial if $p = 2$, since the function $f_2 = \mathrm{XOR}$. For $p \geq 3$, the constraints $\{f_p(x_1, \ldots, x_p), T(x_3), \ldots, T(x_p)\}$ perfectly and strictly implement $\mathrm{NAND}(x_1, x_2)$. Similarly the constraints $\{f_p(x_1, \ldots, x_p), F(x_3), \ldots, F(x_p)\}$ perfectly and strictly implement the constraint $\mathrm{OR}(x_1, x_2)$. Finally the constraints $\mathrm{OR}(x_1, x_2)$ and $\mathrm{NAND}(x_1, x_2)$ perfectly and strictly implement the constraint $\mathrm{XOR}(x_1, x_2)$. Part (2) follows from the fact that perfect and strict implementations compose (Lemma 3.5).

Finally for Part (3), we first use the constraints $\{\mathrm{NAND}_p(x_1, \ldots, x_p), F(x_3), \ldots, F(x_p)\}$ to implement, strictly and perfectly, the constraint $\mathrm{NAND}(x_1, x_2)$. Now we may use $\{\mathrm{NAND}(x_1, x_2)$, $\mathrm{NAND}(x_1 . x_2), T(x_1), T(x_2)\}$ to obtain a 3-implementation of the constraint $\mathrm{XOR}(x_1, x_2)$. (Note that in the case the implementation is not perfect.) □

Lemma 4.11 *Let f be a constraint which is not 2-monotone. Then* $\{f, T, F\} \overset{s}{\Longrightarrow} \mathrm{XOR}$.

Proof: The proof is divided into three cases, which depend on which of the 3 conditions defining 2-monotonicity is violated by f. We first state and prove the claims.

Claim 4.12 *If f is a function violating property (a) of Lemma 4.9, then* $\{f, T, F\} \overset{s}{\Longrightarrow} \mathrm{XOR}$.

Proof: There exists some assignment s satisfying f, and two assignments s_0 and s_1 such that $Z(s) \subseteq Z(s_0)$ and $O(s) \subseteq O(s_1)$, such that $f(s_0) = f(s_1) = 0$. Rephrasing slightly, we know that there exists a triple (s_0, s, s_1) with the following properties:

$$f(s_0) = f(s_1) = 0; f(s) = 1;\ Z(s_0) \supseteq Z(s) \supseteq Z(s_1) \qquad (1)$$

Note that the condition $Z(s_0) \supseteq Z(s) \supseteq Z(s_1)$ implies that $O(s_0) \subseteq O(s) \subseteq O(s_1)$. We call property (1) the "sandwich property". Of all triples satisfying the sandwich property, pick one that minimizes $|Z(s_0) \cap O(s_1)|$.

Without loss of generality, assume that $Z(s_0) \cap O(s_1) = \{1, \ldots, p\}$, $Z(s_0) \cap Z(s_1) = \{p+1, \ldots, q\}$, and $O(s_0) \cap O(s_1) = \{q+1, \ldots, k\}$. (Notice that the sandwich property implies that $O(s_0) \cap Z(s_1) = \emptyset$.) Let f_1 be the constraint given by $f_1(x_1, \ldots, x_p) = f(x_1, \ldots, x_p, 0, \ldots, 0, 1, \ldots, 1)$. Notice that the constraint applications $f(x_1 \ldots x_k)$ and $T(x_i)$ for every $i \in O(s_0) \cap O(s_1)$ and $F(x_i)$ for every $i \in Z(s_0) \cap Z(s_1)$ implement the function f_1. Thus it suffices to show that $\{f_1, T, F\}$ implements XOR.

Below we examine some properties of the constraint f_1. We will use the characters t, t', t_i, t'_i to denote assignments to f_1, while we use the characters s, s', s_i, s'_i to denote assignments to f. Note that:

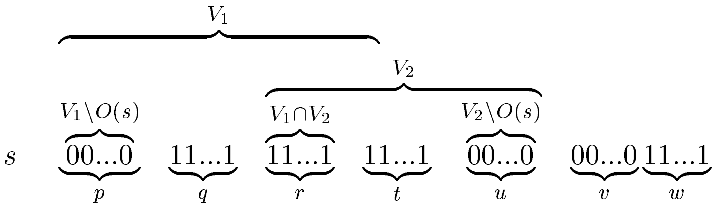
1. $f_1(\mathbf{0}) = f_1(\mathbf{1}) = 0$.
2. f_1 has a satisfying assignment. Thus p (the arity of f_1) is at least 2.
3. If $f_1(t_1) = 0$ for some $t \neq \mathbf{1}$, then for every assignment t such that $Z(t) \supseteq Z(t_1)$, it is the case that $f_1(t_1) = 0$: This follows from the minimality of $|Z(s_0) \cap O(s_1)|$ above. If not then consider the assignments $s'_0 = s_0$, $s' = t0^{q-p}1^{k-q}$, and $s'_1 = t_1 0^{q-p} 1^{k-q}$. The triple (s'_0, s', s'_1) also satisfies the sandwich property and has a smaller value of $|Z(s'_0) \cap O(s'_1)|$.)
4. If $f_1(t_0) = 0$ for some $t_0 \neq \mathbf{0}$, then for every assignment t such that $O(t) \supseteq O(t_0)$, it is the case that $f_1(t) = 0$. (Again from the minimality of $|Z(s_0) \cap O(s_1)|$.)

These properties of f_1 now allow us to identify f_1 almost completely. We show that either (a) $p = 2$ and $f_1(x_1 x_2)$ is either $\mathrm{AND}_{2,1}(x_1, x_2)$ or $\mathrm{AND}_{2,1}(x_2, x_1)$; or (b) f is satisfied by every assignment other than the all zeroes assignment and the all ones assignment. In either case $\{f_1, T, F\}$ strictly implements XOR by Lemma 4.10, Parts (1) and (2). (Note that Part (1) of Lemma 4.10 only yields a strict (but not perfect) implementation.) Thus proving that either (a) or (b) holds concludes the proof of the claim.

Suppose (b) is not the case. I.e., f_1 is left unsatisfied by some assignment t and $t \neq \mathbf{0}$ and $t \neq \mathbf{1}$. Then we will show that the only assignment that can satisfy f_1 is $\bar{t}$. But this implies that t, $\bar{t}$, $\mathbf{0}$ and $\mathbf{1}$ are the only possible assignments to f_1, implying p must be 2 thereby yielding that (a) is true. Thus it suffices to show that if $f_1(t) = 0$, and $t' \neq \bar{t}$, then $f_1(t') = 0$. Since t' is not the bitwise complement of t, there must exist some input variable which shares the same assignment in t and t'. W.l.o.g. assume this is the variable x_1. Consider the case that this variable takes on the value 0 in the assignment t. Then we claim that the assignment $f_1(01 \ldots 1) = 0$. This is true since $O(01 \ldots 1) \supseteq O(t)$. Now notice that $f(t') = 0$ since $Z(t') \supseteq Z(01 \ldots 1)$. (In case the first variable takes on the value 1 in the assignment t, is symmetric.) Thus we conclude that either (a) or (b) always holds and this concludes the proof of the claim. □

Claim 4.13 *Suppose f violates property (b) of Lemma 4.9. Then* $\{f, T, F\} \overset{s/p}{\Longrightarrow} \mathrm{XOR}$.

Proof: Let V_1 and V_2 be two 1-terms such that $V_1 \cap V_2$ is not a 1-term. I.e., There exists an assignment s s.t. $O(s) \supseteq V_1 \cap V_2$ and $f(s) = 0$. Among all such assignments let s be the one with the maximum number of 1's. The situation looks as shown below:



In other words $s = 0^p 1^{q+r+t} 0^{u+v} 1^w$ and $f(s) = 0$. Furthermore, every assignment of the form $1^{p+q+r} *^{t+u+v+w}$ satisfies f and every assignment of the form $*^{p+q} 1^{r+t+u} *^{v+w}$ satisfies f (where the $*$s above can be replaced by any of 0/1 independently). In particular this implies that $p, u \geq 1$. Consider the function f_1 on $p+u \geq 2$ variables obtained from f by restricting the variables in $O(s)$ to 1 and restricting the variables in $Z(s) - (V_1 \cup V_2)$ to 0. Notice that the constraint applications $f(x_1 \ldots x_k)$, $T(x_i)$ for $i \in O(s)$ and $F(x_i)$ for $i \in Z(s) - (V_1 \cup V_2)$ strictly implement f_1. Thus it suffices to show that $\{f_1, T, F\}$ implements XOR. We do so by observing that $f_1(x_1 \ldots x_{p+u})$ is the function NAND_{p+u}. Notice that $f_1(\mathbf{0}) = 0$. Furthermore if $f_1(t) = 0$ for any other assignment t then it contradicts the maximality of the number of 1's in s. The claim now follows from Lemma 4.10, Part (3), which shows that the family $\{\mathrm{NAND}_{p+u}, T, F\}$ implements XOR, provided $p + u \geq 2$. □

Claim 4.14 *Suppose f violates property (c) of Lemma 4.9. Then $\{f, T, F\} \overset{s/p}{\Longrightarrow}$ XOR.*

Proof: Similar to proof of the claim above. □

The lemma now follows from the fact any constraint f_2 that is not 2-monotone must violate one of the properties (a), (b) or (c) from Lemma 4.9. □

4.4 Affine functions

Lemma 4.15 ([42]) *f is an affine function if and only if for every three satisfying assignments s_1, s_2 and s_3 to f, $s_1 \oplus s_2 \oplus s_3$ is also a satisfying assignment.*

We first prove a simple consequence of the above which gives a slightly simpler sufficient condition for a function to be affine.

Corollary 4.16 *If f is not affine, then there exist two satisfying assignments s_1 and s_2 for f such that $s_1 \oplus s_2$ does not satisfy f.*

Proof: Assume otherwise. Then for any three satisfying assignments s_1, s_2 and s_3, we have that $f(s_1 \oplus s_2) = 1$ and hence $f((s_1 \oplus s_2) \oplus s_3) = 1$, thus yielding that f is affine. □

Lemma 4.17 *If f is an affine constraint then any function obtained by restricting some of the variables of f to constants and existentially quantifying over some other set of variables is also affine.*

Proof: We use Lemma 4.15 above. Let f_1 be a function derived from f as above. Consider any three assignments s_1', s_2' and s_3' which satisfy f_1. Let s_1 s_2 and s_3 be the respective extensions which satisfy f. Then the assignment $s_1 \oplus s_2 \oplus s_3$ extends $s_1' \oplus s_2' \oplus s_3'$ and satisfies f. Thus $s_1' \oplus s_2' \oplus s_3'$ satisfies f_1. Thus (using Lemma 4.15) again, we find that f_1 is affine. □

Lemma 4.18 *If f is an affine function which is not of width-2 then $\{f\} \overset{s/p}{\Longrightarrow} \mathrm{XOR}_p$ or $\{f\} \overset{s/p}{\Longrightarrow}$ XNOR_p, for some $p \geq 3$.*

Proof: Let k be the arity of f. Define a dependent set of variables to be a set of variables $S \subseteq \{1, \ldots, k\}$ such that not every assignment to the variables in S extends to a satisfying assignment of f. A dependent set S is minimally dependent set if no strict subset $S' \subset S$ is a dependent set. Notice that f can be expressed as the conjunction of constraints on its minimally dependent sets. Thus if f is not of width-2 then it must have a minimally dependent set S of cardinality at least 3. Assume $S = \{1, \ldots, p\}$, where $p \geq 3$. Consider the function

$$f_1(x_1 \ldots x_p) = \exists x_{p+1}, \ldots, x_k \text{ s.t. } f(x_1, \ldots x_k).$$

f_1 is affine (by Lemma 4.17), is not satisfied by every assignment and has at least 2^{p-1} satisfying assignments. Thus f_1 has exactly 2^{p-1} assignments (since the number of satisfying assignments must be a power of 2). Thus f_1 is described by exactly one linear constraint and by the minimality of S this must be the constraint $\mathrm{XOR}(x_1 \ldots x_p)$ or the constraint $\mathrm{XNOR}(x_1 \ldots x_p)$. □

4.5 Horn Clauses, 2CNF and IHS

Lemma 4.19 *If f is a weakly positive (weakly negative / IHS-B+/ IHS-B-/ 2CNF) constraint then any function obtained by restricting some of the variables of f to constants and existentially quantifying over some other set of variables is also weakly positive (resp. weakly negative / IHS-B+/ IHS-B-/ 2CNF).*

Proof: It is easy to see that f remains weakly positive (weakly negative / IHS-B+/ IHS-B-/ 2CNF) when some variable is restricted to a constant. Hence it suffices to consider the case where some variable y is quantified existentially. (Combinations of the possibilities can then be handled by a simple induction.) Thus consider the function $f_1(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \exists y \text{ s.t. } f(x_1, \ldots, x_k, y)$. Let

$$f(x_1, \ldots, x_k, y) = \left(\bigwedge_{j=1}^{m} C_j(\bar{x}) \right) \bigwedge \left(\bigwedge_{j_0=1}^{m_0} (C^0_{j_0}(\bar{x}) \bigvee y) \right) \bigwedge \left(\bigwedge_{j_1=1}^{m_1} (C^1_{j_1}(\bar{x}) \bigvee \neg y) \right)$$

be a conjunctive normal form expression for f which shows it is weakly positive (weakly negative / IHS-B+/ IHS-B-/ 2CNF), where the clauses C_j, $C^0_{j_0}$ and $C^1_{j_1}$ involve literals on the variables $x_1, \ldots, x_k$.

We first show a simple transformation which creates a conjunctive normal form expression for f_1. Later we show that f_1 inherits the appropriate properties of f.

Define $m_0 \times m_1$ clauses $C^{01}_{j_0 j_1}(\bar{x}) \stackrel{\text{def}}{=} C^0_{j_0}(\bar{x}) \bigvee C^1_{j_1}(\bar{x})$. Next, we note that $f_1(\bar{x})$ can be expressed as follows:

$$\begin{aligned}
f_1(\bar{x}) &= f_1(\bar{x}, 0) \bigvee f_1(\bar{x}, 1) \\
&= \left((\bigwedge_j C_j(\bar{x})) \bigwedge (\bigwedge_{j_0} C^0_{j_0}(\bar{x})) \right) \bigvee \left((\bigwedge_j C_j(\bar{x})) \bigwedge (\bigwedge_{j_1} C^1_{j_1}(\bar{x})) \right) \\
&= (\bigwedge_j C_j(\bar{x})) \bigwedge \left((\bigwedge_{j_0} C^0_{j_0}(\bar{x})) \bigvee (\bigwedge_{j_1} C^1_{j_1}(\bar{x})) \right) \\
&= (\bigwedge_j C_j(\bar{x})) \bigwedge \left(\bigwedge_{j_0} \bigwedge_{j_1} C^{01}_{j_0 j_1}(\bar{x}) \right) \qquad (2)
\end{aligned}$$

To conclude we need to verify that the right hand side of (2) satisfies the same properties as f. Furthermore we only have to consider clauses of the form $C^{01}_{j_0j_1}(\bar{x})$ since all other clauses are directly from the expression for f. We verify this below:

- If f is weakly positive, then the clause $C^0_{j_0}$ involves at most one negated variable, and the clause $C^1_{j_1}$ involves no negated variable (since the clause participating in f is $(C^1_{j_1}(\bar{x}) \bigvee \neg y)$ which has a negated y involved in it). Thus the clause defining $C^{01}_{j_0j_1}$ also has at most one negated variable.)
- Similarly if f is weakly negative, then the clauses $C^{01}_{j_0j_1}$ has at most one positive literal.
- If f is 2CNF, then the clauses $C^0_{j_0}$ and $C^1_{j_1}$ are of length 1 and hence the clause $C^{01}_{j_0j_1}$ is of length at most 2.
- If f is IHS-$B+$ then the clause $C^0_{j_0}$ either has only one literal which is negated or has only positive literals. Furthermore $C^1_{j_1}$ has at most one positive literal. Thus $C^{01}_{j_0j_1}$ either has only positive literals or has at most two literals one of which is negated. Hence $C^{01}_{j_0j_1}$ is also IHS-$B+$.
- Similarly if f in IHS-$B-$ then the clause $C^{01}_{j_0j_1}$ is also IHS-$B-$.

This concludes the proof of the lemma. □

Lemma 4.20 *f is a weakly positive (weakly negative) constraint if and only if all its maxterms are weakly positive (weakly negative).*

Proof: We prove the lemma for the weakly positive case. The other case is similar. For the easy direction, recall that a function can be expressed as the conjunction of all its maxterms. If all maxterms are weakly positive then this gives a weakly positive representation of f.

For the other direction, assume for contradiction that f be a weakly positive constraint that has $C = \neg x_1 \bigvee \cdots \bigvee \neg x_p \bigvee x_{p+1} \bigvee \cdots \bigvee x_q$ as a maxterm, for some $p \geq 2$. Let the arity of f be k. Consider the function

$$f_1(x_1x_2) \stackrel{\text{def}}{=} \exists x_{q+1}, \ldots, x_k \text{ s.t. } f(x_1x_2 1^{p-2} 0^{q-p} x_{q+1} \ldots x_k).$$

Since C is an admissible clause in a CNF representation of f, we have that if we set $x_1, \ldots, x_p$ to 1 and setting $x_{p+1}, \ldots, x_q$ to 0 then no assignment to $x_{q+1}, \ldots, x_k$ satisfies f. Thus we find that $f_1(11) = 0$. By the fact that clause is a maxterm we have that both the assignments $x_1 \ldots x_q = 01^{p-1}0^{q-p}$ and $x_1 \ldots x_q = 101^{p-2}0^{q-p}$ can be extended to satisfying assignments of f. Thus we find that $f_1(10) = f_1(01) = 1$. Thus f_1 is either the function NOR or XOR. It can be verified easily that neither of these is 2-monotone. (Every basic weakly positive function on 2 variables is unsatisfied on at least one of the two assignments 01 or 10.) But this is in contradiction to Lemma 4.19 that showed that every function obtained by restricting some variables of f to constants and existentially quantifying over some others should yield a weakly positive function. □

Lemma 4.21 *f is a* 2CNF *constraint if and only if all its maxterms are* 2CNF.

Proof: The "if" part is obvious. For the other direction we use Lemma 4.19. Assume for contradiction that f has a maxterm of the form $x_1 \bigvee x_2 \bigvee x_3 \bigvee \cdots \bigvee x_p \bigvee \neg x_{p+1} \bigvee \cdots \bigvee \neg x_q$. (For simplicity we assume $p \geq 3$. Other cases where one or more of the variables $x_1, \ldots, x_3$ are negated can be handled similarly.) Consider the function

$$f_1(x_1x_2x_3) \stackrel{\text{def}}{=} \exists x_{q+1}, \ldots, x_k \text{ s.t. } f(x_1, x_2, x_3, 0^{p-3}, 1^{q-p}, x_{q+1}, \ldots, x_k).$$

Then since $x_1 \bigvee x_2 \bigvee x_3 \ldots$ is a maxterm of f, we have that $f_1(000) = 0$ and $f_1(100) = f_1(010) = f_1(001) = 1$. We claim that f_1 can not be a 2CNF function. If not, then to make $f_1(000) = 0$, at least one of the clauses x_1, x_2, x_3, $x_1 \bigvee x_2$, $x_2 \bigvee x_3$ or $x_3 \bigvee x_1$, should be a clause of f_1 in any 2CNF representation. But all these clauses are left unsatisfied by at least one of the assignments 100, 010 or 001. This validates our claim that f_1 is not a 2CNF constraint. But f_1 was obtained from f by setting some variables to a constant and existentially quantifying over others and by Lemma 4.19 f_1 must also be a 2CNF function. This yields the desired contradiction. □

Lemma 4.22 *An affine function f is a width-2 affine function if and only if all its minimally dependent sets are of cardinality at most 2.*

Proof: We use the fact that $\mathcal{F}_{2\text{A}} \subseteq \mathcal{F}_{2\text{CNF}} \cap \mathcal{F}_{\text{A}}$. Suppose $f \in \mathcal{F}_{2\text{A}}$ has a minimally dependent set of size $p \geq 3$ and say the set is $x_1, \ldots, x_p$. Then by existential quantification over the variables $x_{p+1}, \ldots, x_k$ and by setting the variables $x_4, \ldots, x_p$ to 0, we obtain the function $f_1(x_1, x_2, x_3)$ which is an affine function (by Lemma 4.17) with x_1, x_2, x_3 as a minimally dependent set. Thus this function is either XOR$_3$ or XNOR$_3$. But now notice that neither of these functions is a 2CNF function. But since f is a 2CNF function Lemma 4.19 implies that f_1 must also be a 2CNF function. This yields the required contradiction. □

5 Classification of Max CSP

The main results of this section are in Sections 5.1 and 5.2. These results were originally obtained by Creignou [11]. Her focus however is on the the complexity of finding optimal solutions to the optimization problems. The proofs for hardness of approximation are left to the reader to verify. We give full proofs using the notions of implementations. Our proof is also stronger since it does not assume replication of variables as a basic primitive. This allows us to talk about problems such as Max EkSat. In Section 5.3 we extend Schaefer's results to establish the hardness of satisfiable Max CSP problems. Similar results, again with replication of variables being allowed, were first shown by Hunt et al. [26].

5.1 Containment results for Max CSP

We start with the polynomial time solvable cases.

Proposition 5.1 Weighted Max CSP($\mathcal{F}$) *(*Weighted Min CSP($\mathcal{F}$)*) is in* PO *if $\mathcal{F}$ is 0-valid (1-valid).*

Proof: Set each variable to zero (resp. one); this satisfies all the constraints. □

Before proving the containment in PO of Max CSP($\mathcal{F}$) for 2-monotone function families, we show that the corresponding Weighted Min CSP($\mathcal{F}$) is in PO. The containment for Weighted Max CSP($\mathcal{F}$) will follow easily.

Lemma 5.2 Weighted Min CSP($\mathcal{F}$) *is in* PO *if* $\mathcal{F}$ *is 2-monotone.*

Proof: This problem reduces to the problem of finding s-t min-cut in directed weighted graphs. 2-monotone constraints have the following possible forms :

(a) $\text{AND}_p(x_{i_1}, \ldots, x_{i_p})$,

(b) $\text{NOR}_q(x_{j_1}, \ldots, x_{j_q})$, and

(c) $\text{AND}_p(x_{i_1}, \ldots, x_{i_p}) \bigvee \text{NOR}_q(x_{j_1}, \ldots, x_{j_q})$.

Construct a directed graph G with two special nodes F and T and a vertex v_i corresponding to each variable x_i in the input instance. Let ∞ denote an integer larger than the total weight of all constraints.

Now we proceed as follows for each of the above classes of constraints :

- For a constraint C of weight w of the form (a), create a new node e_C and add an edge from each v_{i_l}, $l \in [p]$, to e_C of capacity ∞ and an edge from e_C to T of capacity w.
- For a constraint C of weight w of the form (b), create a new node $\overline{e_C}$ and add an edge from $\overline{e_C}$ to each v_{j_l}, $l \in [q]$, of capacity ∞, and an edge from F to $\overline{e_C}$ of capacity w.
- Finally, for a constraint C of weight w of the form (c), we create two nodes e_C and $\overline{e_C}$. For every $l \in [p]$, we add an edge from v_{i_l} to e_C of capacity ∞, and for every $l \in [q]$, we add an edge from $\overline{e_C}$ to v_{j_l} of capacity ∞, and finally an edge from e_C to $\overline{e_C}$ of capacity w. (Note in this case there are no edges connecting F or T to any of the vertices.)

Notice that each vertex of type e_C or $\overline{e_C}$ can be associated with a term: e_C with a term on positive literals and $\overline{e_c}$ with a term on negated literals. We use this association to show that the value of the min F-T cut in this directed graph equals the weight of the minimum number of unsatisfied constraints in the given Weighted Min CSP($\mathcal{F}$) instance.

Given an assignment which fails to satisfy constraints of weight W, we associate a cut as follows: Vertex v_i is placed on the F side of the cut if and only if it is set to 0. A vertex e_C is placed on the T side if and only if the term associated with it is satisfied. A vertex $\overline{e_C}$ is placed on the F side if and only if the term associated with it is satisfied. It can be verified that such an assignment has no directed edges of capacity ∞ going from the F side of the cut to the T side of the cut. Furthermore for every constraint C of weight w, the associated edge of capacity w crosses the cut if and only if the constraint is not satisfied. Thus the capacity of this cut is exactly W and thus we find that the min F-T cut value is at most W.

In the other direction, we show that given a F-T cut in this graph of cut capacity $W < \infty$, there exists an assignment which fails to satisfy constraints of weight at most W. Such an assignment is simply to assign $x_i = 0$ iff v_i is on the F side of the cut. Note that for any constraint C, the associated vertices e_C and $\overline{e_c}$ (whichever exist) may be placed on the T and F sides of the cut (respectively) only if the associated term is satisfied (else there will be an edge of capacity ∞ crossing the cut). Thus, if a constraint C of capacity w is not satisfied by this assignment, then the edge of capacity w corresponding to C must cross the cut. Summing up we find that the assignment fails to satisfy constraints of total weight at most W.

Putting both directions together, we find that the min F-T cut in this graph has capacity exactly equal to the optimum of the Weighted Min CSP{XOR} instance, and thus the latter problem can be solved exactly in polynomial time. □

For the sake of completeness we also prove the converse direction to the above lemma. We show that the s-t min-cut problem can be phrased as a Min CSP($\mathcal{F}$) problem for a 2-monotone family $\mathcal{F}$.

Lemma 5.3 *The s-t min-cut problem is in* Weighted Min CSP($\{\mathrm{OR}_{2,1}, T, F\}$).

Proof: Given an instance $G = (V, E)$ of the s-t min-cut problem, we construct an instance of Weighted Min CSP($\mathcal{F}$) on variables $x_1, x_2, \ldots, x_n$ where x_i corresponds to the vertex $i \in V - \{s, t\}$:

- For each edge $e = (s, i)$ with weight w_e, we create the constraint $F(x_i)$ with weight w_e.
- For each edge $e = (i, t)$ with weight w_e, we create the constraint $T(x_i)$ with weight w_e.
- For each edge $e = (i, j)$ with weight w_e and such that $i, j \notin \{s, t\}$, we create the constraint $\mathrm{OR}_{2,1}(x_j, x_i)$ with weight w_e.

Given a solution to this instance of Weighted Min CSP($\mathcal{F}$), we construct an s-t cut by placing the vertices corresponding to the false variables on the s-side of the cut and the remaining on the t-side of the cut. It is easy to verify that an edge e contributes to the cut iff its corresponding constraint is unsatisfied. Hence the optimal Min CSP($\mathcal{F}$) solution and the optimal s-t min-cut solution coincide. □

Going back to our main objective, we obtain as a simple corollary to Lemma 5.2 the following:

Corollary 5.4 *For every $\mathcal{F} \subseteq \mathcal{F}_{2\mathrm{M}}$,* Weighted Max CSP($\mathcal{F}$)$\in$ PO.

Proof: Follows from the fact that given an instance $\mathcal{I}$ of Weighted Max CSP($\mathcal{F}$), the optimum solution to $\mathcal{I}$ viewed as an instance of Weighted Min CSP($\mathcal{F}$) is also an optimum solution to the Weighted Max CSP($\mathcal{F}$) version. □

Finally we prove a simple containment result for all of Max CSP($\mathcal{F}$) which follows as an easy consequence of Proposition 3.7.

Proposition 5.5 *For every $\mathcal{F}$,* Weighted Max CSP($\mathcal{F}$) *is in* APX.

Proof: Follows from Proposition 3.7 and the fact that the total weight of all constraints is an upper bound on the optimal solution. □

5.2 Negative results for Max CSP

In this section we prove that if $\mathcal{F} \not\subseteq \mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}$ then Max CSP($\mathcal{F}$) is APX-hard. We start with a simple observation which establishes Max CSP(XOR) as our starting point.

Lemma 5.6 Max CSP(XOR) *is* APX-*hard.*

Proof: We observe that Max CSP(XOR) captures the MAX CUT problem shown to be APX-hard by [39, 3]. Given a graph $G = (V, E)$ with n vertices and m edges, create an instance $\mathcal{I}_G$ of Max CSP({XOR}) with one variable x_u for every vertex $u \in V$ and with constraints $\mathrm{XOR}(x_u, x_v)$ corresponding to every edge $\{u, v\} \in E$. It is easily seen there is a one-to-one correspondence between (ordered) cuts in G and the assignments to the variables of $\mathcal{I}_G$ which maintains the values of the objective functions (i.e., the cut value and the number of satisfied constraints). □

We start with the following lemma which shows how to use the functions which are not 0-valid or 1-valid.

Lemma 5.7 *If* $\mathcal{F} \not\subseteq \mathcal{F}_0, \mathcal{F}_1$ *then* Max CSP($\mathcal{F} \cup \{T, F\}$) *is AP-reducible to* Max CSP($\mathcal{F}$) *and* Min CSP($\mathcal{F} \cup \{T, F\}$) *is A-reducible to* Min CSP($\mathcal{F}$)*.*

Proof: Let f_0 be the function from $\mathcal{F}$ that is not 0-valid and let f_1 be the function that is not 1-valid. If some function g in $\mathcal{F}$ is is not C-closed, then, by Lemma 4.6 $\mathcal{F}$ perfectly and strictly implements T and F. Hence, by Lemmas 3.8 and 3.10, Max CSP($\mathcal{F} \cup \{T, F\}$) is AP-reducible to Max CSP($\mathcal{F}$) and Min CSP($\mathcal{F} \cup \{T, F\}$) is A-reducible to Min CSP($\mathcal{F}$).

Otherwise, every function of $\mathcal{F}$ is C-closed and hence by Lemma 4.5, $\mathcal{F}$ perfectly and strictly implements the XOR function and hence, by Proposition 3.3, the XNOR function. Thus it suffices to show that Max CSP($\mathcal{F} \cup \{T, F\}$) is AP-reducible to Max CSP($\mathcal{F} \cup \{\mathrm{XOR}, \mathrm{XNOR}\}$) (and Min CSP($\mathcal{F} \cup \{T, F\}$) is A-reducible to Min CSP($\mathcal{F} \cup \{\mathrm{XOR}, \mathrm{XNOR}\}$)) for C-closed families $\mathcal{F}$. Here we use an idea from [8] described next.

Given an instance $\mathcal{I}$ of Max CSP($\mathcal{F} \cup \{T, F\}$) on variables $x_1, \ldots, x_n$ and constraints $C_1, \ldots, C_m$, we define an instance $\mathcal{I}'$ of Max CSP($\mathcal{F} \cup \{\mathrm{XOR}, \mathrm{XNOR}\}$) (Min CSP($\mathcal{F} \cup \{\mathrm{XOR}, \mathrm{XNOR}\}$)) whose variables are $x_1, \ldots, x_n$ and additionally one new auxiliary variable x_F. Each constraint of the form $F(x_i)$ (resp. $T(x_i)$) in $\mathcal{I}$ is replaced by a constraint $\mathrm{XNOR}(x_i, x_F)$ (resp. $\mathrm{XOR}(x_i, x_F)$). All the other constraints are not changed. Thus $\mathcal{I}'$ also has m constraints. Given a solution $a_1, \ldots, a_n, a_F$ for $\mathcal{I}'$ that satisfies m' of these constraints, notice that the assignment $\neg a_1, \ldots, \neg a_n, \neg a_F$ also satisfies the same collection of constraints (since every function in $\mathcal{F}$ is C-closed). In one of these cases the assignment to x_F is false and then we notice that a constraint of $\mathcal{I}$ is satisfied if and only if the corresponding constraint in $\mathcal{I}'$ is satisfied. Thus every solution to $\mathcal{I}'$ can be mapped to a solution to $\mathcal{I}$ with the same contribution to the objective function. □

The required lemma now follows as a simple combination of Lemmas 4.9 and 5.7.

Lemma 5.8 *If* $\mathcal{F} \not\subseteq \mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}$*, then* Max CSP($\mathcal{F}$) *is* APX*-hard.*

Proof: By Lemma 4.11 $\mathcal{F} \cup \{T, F\}$ strictly implements the XOR function. Thus Max CSP(XOR) AP-reduces to Max CSP($\mathcal{F} \cup \{T, F\}$) which in turn (by Lemma 5.7) AP-reduces to Max CSP($\mathcal{F}$). Thus Max CSP($\mathcal{F}$) is APX-hard. □

5.3 Hardness at Gap Location 1

Schaefer's dichotomy theorem can be extended to show that in the cases where Sat($\mathcal{F}$) in NP-hard to decide, it is actually hard to distinguish satisfiable instances from instances which are not satisfiable in a constant fraction of the constraints. This is termed hardness at gap location 1 by Petrank [40] who highlights the utility of such hardness results in other reductions. The essential observation needed is that perfect implementations preserve hardness gaps located at 1 and that Schaefer's proof is based on perfect implementations.

However Schaefer's proof of NP-hardness in his dichotomy theorem relies on the ability to replicate variables within a constraint application. Specifically, the following lemma can be abstracted from his paper.

Lemma 5.9 ([42]) *If $\mathcal{F}$ is not 0-valid or 1-valid or affine or bijunctive or weakly positive or weakly negative, then* $\mathcal{F} \cup \{\mathrm{XNOR}\} \overset{p}{\Longrightarrow} \mathcal{F}_{3\mathrm{SAT}}$.

In this section, we show that a family $\mathcal{F}$ that is not decidable also perfectly implements the XNOR constraint and thus the lemma above can be strengthened. We start with the following lemma that shows how to use functions that are not weakly negative.

Lemma 5.10 *If f is not weakly negative then* $\{f, T, F\} \overset{p}{\Longrightarrow}$ XOR *or* $\{f, T, F\} \overset{p}{\Longrightarrow}$ OR. *Similarly, if f is not weakly positive then* $\{f, T, F\} \overset{p}{\Longrightarrow}$ XOR *or* $\{f, T, F\} \overset{p}{\Longrightarrow}$ NAND.

Proof: We only prove the first part – the second part follows by symmetry. By Lemma 4.20 we find that f has a maxterm with at least two positive literals. W.l.o.g. the maxterm is of the form $x_1 \bigvee x_2 \bigvee \cdots x_p \bigvee \neg x_{p+1} \bigvee \cdots \bigvee \neg x_q$, with $p \geq 2$. We consider the function f' which is f existentially quantified over all variables but $x_1, \ldots, x_q$. Further we set $x_3, \ldots, x_p$ to 0 and $x_{p+1}, \ldots, x_q$ to 1. Then the assignment $x_1 = x_2 = 0$ is a non-satisfying assignment. The assignments $x_1 = 0 \neq x_2$ and $x_1 \neq 0 = x_2$ must be satisfying assignments by the definition of maxterm (and in particular by the minimality of the clause). The assignment $x_1 = x_2 = 1$ may go either way. Depending on this we get either the function XOR or OR. □

Corollary 5.11 *If f_2 is not weakly positive and f_3 is not weakly negative, then* $\{f_2, f_3, T, F\} \overset{p}{\Longrightarrow}$ XOR.

Lemma 5.12 *If $\mathcal{F}$ is not 0-valid or 1-valid or weakly positive or weakly negative, then* $\mathcal{F} \overset{s/p}{\Longrightarrow} \{\mathrm{XOR}, \mathrm{XNOR}\}$.

Proof: If $\mathcal{F}$ is C-closed then, by Lemma 4.5, we immediately get a strict and perfect implementation of XOR. If it is not C-closed then, by Lemma 4.6, we get perfect and strict implementations of the constraints T and F. Applying Corollary 5.11 now, we get a perfect and strict implementation of XOR in this case also. Finally we use Proposition 3.3 to get a perfect and strict implementation of XNOR from the constraint XOR. □

Combining Lemma 5.9 and the above, we get the following corollary:

Corollary 5.13 *If $\mathcal{F}$ is not 0-valid or 1-valid or affine or bijunctive or weakly positive or weakly negative, then* $\mathcal{F} \overset{p}{\Longrightarrow} \mathcal{F}_{3\mathrm{SAT}}$.

Thus we get the following theorem.

Theorem 5.14 *For every constraint set $\mathcal{F}$ either SAT($\mathcal{F}$) is easy to decide, or there exists $\epsilon = \epsilon_{\mathcal{F}} > 0$ such that it is NP-hard to distinguish satisfiable instances of SAT($\mathcal{F}$), from instances where $1 - \epsilon$ fraction of the constraints are not satisfiable.*

6 Classification of Max Ones

Again we will first prove the positive results and then show the negative results. But before we do either, we will show a useful reduction between unweighted and weighted Max Ones($\mathcal{F}$) problems which holds for most interesting function families $\mathcal{F}$.

6.1 Preliminaries

We begin with a slightly stronger notion of the definition of polynomial-time solvability of Sat($\mathcal{F}$) (than that of [42]). We then show that given this stronger form of polynomial time decidability the weighted and unweighted cases of Max Ones($\mathcal{F}$) are equivalent by showing that this stronger form of polynomial time decidability leads to a polynomial approximation algorithm. We conclude by showing that for the Max Ones problems which we hope to show to be APX-complete or poly-APX-complete, the strong form of decidability does hold.

Definition 6.1 *We say that a constraint family $\mathcal{F}$ is* strongly decidable *if, given m constraints from $\mathcal{F}$ on n variables $x_1, \ldots, x_n$* and an index $i \in \{1, \ldots, n\}$, *there exists a polynomial time algorithm to find an assignment to $x_1, \ldots, x_n$ satisfying all m constraints and additionally satisfying the property $x_i = 1$ if one such exists.*

Lemma 6.2 *For every strongly decidable constraint family $\mathcal{F}$,* Weighted Max Ones($\mathcal{F}$) *is in* poly-APX*.*

Proof: Consider an instance of Weighted Max Ones($\mathcal{F}$) with variables $x_1, \ldots, x_n$, constraint applications $C_1, \ldots, C_m$ and weights $w_1, \ldots, w_n$. Assume $w_1 \leq w_2 \leq \cdots \leq w_n$. Let i be the largest index such that there exists a feasible solution with $x_i = 1$. Notice that i can be determined in polynomial time due to the strong decidability of $\mathcal{F}$. We also use the strong decidability to find an assignment with $x_i = 1$. It is easily verified that this yields an n-approximate solution. (Weight of this solution is at least w_i, while weight of optimal is at most $\sum_{j=1}^{i} w_j \leq i w_i \leq n w_i$.) □

Before concluding we show that most problems of interest to us will be able to use the equivalence established above between weighted and unweighted problems.

Lemma 6.3 *If $\mathcal{F} \subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{S0}}, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then $\mathcal{F}$ is strongly decidable.*

Proof: Recall that for $i \in [k]$, $f|_{(\{i\},1)}$ is the constraint obtained from f by restricting the ith input to 1. Define $\mathcal{F}^*$ to be the constraint set:

$$\mathcal{F}^* \stackrel{\mathrm{def}}{=} \mathcal{F} \cup \{f|_{i,1} | f \in \mathcal{F}, i \in [k]\}.$$

First, observe that the problem of strong decidability of $\mathcal{F}$ reduces to the decision problem Sat($\mathcal{F}^*$). Further, observe that if $\mathcal{F} \subseteq \mathcal{F}'$ for $\mathcal{F}' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then $\mathcal{F}^* \subseteq \mathcal{F}'$ as well. Lastly, if $\mathcal{F}^* \subseteq \mathcal{F}_{\mathrm{S0}}$, then $\mathcal{F}^* \subseteq \mathcal{F}_0$. Thus in each case we end up with a problem from Sat($\mathcal{F}$) for a family $\mathcal{F}$ which is polynomial time decidable in Schaefer's dichotomy. □

Lemma 6.4 *If $\mathcal{F} \overset{p}{\Longrightarrow} f_0$ for some existential zero constraint f_0, then $\mathcal{F} \overset{p}{\Longrightarrow} \mathcal{F}|_0$. Similarly, if $\mathcal{F} \overset{p}{\Longrightarrow} f_1$ for some existential one constraint f_1, then $\mathcal{F} \overset{p}{\Longrightarrow} \mathcal{F}|_1$.*

Proof: Let $f \in \mathcal{F}$. We show how to implement the constraint $f(0, x_1, \ldots, x_{k-1})$. The proof can be extended to other constraints in $\mathcal{F}|_0$ by induction. Let f_0 be an existential zero constraint implementable by $\mathcal{F}$ and let K be the arity of f_0. Then the constraints $f(y_i, x_1, \ldots, x_{k-1})$, for $i \in [K]$, along with the constraint $f_0(y_1, \ldots, y_K)$ perfectly implement the constraint $f(0, x_1, \ldots, x_{k-1})$. (Observe that since at least one of the y_i's in the set $y_1, \ldots, y_K$ is zero, the constraint $f(0, x_1, \ldots, x_{k-1})$ is being enforced. Furthermore, we can always set all of $y_1, \ldots, y_K$ to zero, ensuring that any assignment to $x_1, \ldots, x_{k-1}$ satisfying $f(0, x_1, \ldots, x_{k-1})$ does satisfy all the constraints listed above.) □

6.2 Containment results

Lemma 6.5 *If $\mathcal{F}$ is 1-valid or weakly positive or width-2 affine, then* Weighted Max Ones($\mathcal{F}$) *is in* PO.

Proof: If $\mathcal{F}$ is 1-valid, then setting each variable to 1 satisfies all constraint applications with the maximum possible variable weight.

If $\mathcal{F}$ is weakly positive, consider the CNF formulae for the $f_i \in \mathcal{F}$ such that each clause has at most one negated variable. Clearly, clauses consisting of a single literal force the assignment of these variables. Setting these variables may create new clauses of a single literal; set these variables and continue the process until all clauses have at least two literals or until a contradiction is reached. In the latter case no feasible assignment is possible. In the former case, setting the remaining variables to one satisfies all constraints, and there exists no feasible assignment with a greater weight of ones.

In the case that $\mathcal{F}$ is affine with width 2, we reduce the problem of finding a feasible solution to that of checking whether a graph is bipartite, and then use the bipartition to find the optimal solution. Notice that each constraint corresponds to a conjunction of constraints of the form $X_i = X_j$ or $X_i \neq X_j$. Create a vertex X_j for each variable X_j and for each constraint $X_i \neq X_j$, add an edge (X_i, X_j). For each constraint $X_i = X_j$, identify the vertices X_i and X_j and associate the sum of their weights to the identified vertex; if this creates a self-loop, then clearly no feasible assignment is possible. Check whether the graph is bipartite; if not, then there is no feasible assignment. If it is bipartite, then for each connected component of the graph choose the larger weight side of the bipartition and set the corresponding variables to one. □

Lemma 6.6 *If $\mathcal{F}$ is affine then* Weighted Max Ones($\mathcal{F}$) *is in* APX.

Remark: Our proof actually shows that Max Ones($\mathcal{F}$) has a 2-approximation algorithm. Combined with the fact that the AP-reduction of Lemma 3.11 does not lose much in the approximation factor we essentially get the same factor for Weighted Max Ones($\mathcal{F}$) as well.

Proof: By Lemmas 3.11, 6.2 and 6.3 it suffices to consider the unweighted case. (Lemma 6.3 shows that $\mathcal{F}$ is strongly-decidable; Lemma 6.2 uses this to show that Weighted Max Ones($\mathcal{F}$) is in poly-APX; and Lemma 3.11 uses this to provide an AP-reduction from Weighted Max Ones($\mathcal{F}$) to Max Ones($\mathcal{F}$).)

Given an instance $\mathcal{I}$ of Max Ones($\mathcal{F}$), notice that finding a solution which satisfies all constraints is the problem of solving a linear system of equations over GF[2]. Say the linear system is given by $Ax = b$, where A is an $m \times n$ matrix, and b is a $m \times 1$ column vector, and the x is an $n \times 1$ vector. Assume w.l.o.g. that the rows of A are independent. By simple row operations and reordering of the variables, we can set up the linear system as $[I|A']x = b'$. Thus if x' represents the vector $\langle x_1, \ldots, x_m \rangle$ and x'' represents the vector $\langle x_{m+1}, \ldots, x_n \rangle$ then the set of feasible solutions to the given linear system are given by

$$\{\langle x', x'' \rangle | x'' \in \{0,1\}^{n-m}, x' = -A'x'' + b'\}.$$

Pick a random element of this set by picking x'' at random and setting x' accordingly. Notice that for any $i \in \{m+1, \ldots, n\}$ $x_i = 1$ w.p. $\frac{1}{2}$. Furthermore, for any $i \in [m]$, x_i is either forced to 0 in all feasible solutions, or x_i is forced to 1 in all feasible solutions or $x_i = 1$ w.p. 1/2. Thus, if $S \subseteq [n]$ is the set of variables which are ever set to 1 in a feasible solution, then expected number of 1's in a random solution is at least $|S|/2$. But S is an upper bound on OPT. Thus the expected value of the solution is at least OPT/2 and hence the solution obtained is 2-approximate solution. □

Proposition 6.7 *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{S0}}, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then* Weighted Max Ones($\mathcal{F}$) $\in$ poly-APX.

Proof: Follows immediately from Lemmas 6.2 and 6.3. □

Proposition 6.8 ([42]) *If $\mathcal{F} \subseteq \mathcal{F}_0$, then* Sat($\mathcal{F}$) *is in* P.

6.3 Hardness results

6.3.1 APX-hard case

We wish to show in this section that if $\mathcal{F}$ is an affine family but not width-2 affine, then Max Ones($\mathcal{F}$) is APX-hard. By Lemmas 6.2 and 3.11 it suffices to show this for Weighted Max Ones($\mathcal{F}$). The basic APX-hard problems we work with in this section are described in the following:

Lemma 6.9 Weighted Max Ones(XNOR$_3$) *and* Weighted Max Ones({XOR, XNOR$_4$}) *are* APX*-hard.*

Proof: We reduce the Max Cut problem to the Weighted Max Ones(XNOR$_3$) problem as follows. Given a graph $G = (V, E)$ we create a variable x_v for every vertex $v \in V$ and a variable y_e for every edge $e \in E$. The weight w_v associated with the vertex variable x_v is 0. The weight w_e of an edge variable y_e is 1. For every edge e between u and v we create the constraint $y_e \oplus x_u \oplus x_v = 0$. It is clear that any 0/1 assignment to the x_v's define a cut and for an edge $e = \{u, v\}$, y_e is one iff u and v are on opposite sides of the cut. Thus solutions to the Weighted Max Ones problem correspond to cuts in G with the objective function being the number of edges crossing the cut. This shows the APX-hardness of Weighted Max Ones(XNOR$_3$).

The reduction for Weighted Max Ones({XOR, XNOR$_4$}) is similar. Given a graph $G = (V, E)$, we create the variables x_v for every $v \in V$, y_e for every $e \in E$ and one global variable z (which is supposed to be zero) and $m \stackrel{\mathrm{def}}{=} |E|$ auxiliary variables y'_e for every $e \in E$.. For every edge $e = \{u, v\}$ in G we impose the constraints $y_e \oplus x_u \oplus x_v \oplus z = 0$. In addition we throw in the constraints $z \oplus y'_e = 1$ for every $i \in \{1, \ldots, m\}$. Finally we make the weight of the vertex variables and z zero and the weight of the edge variables y_e and the auxiliary variables y'_e is made 1. The optimum to this Weighted Max Ones problem is Max Cut$(G) + m$. Given an r-approximate solution for the Weighted Max Ones({XOR$_4$, XOR}) instance created above, we consider the two possible solutions (as usual): (1) The solution induced by the assignment with 0 vertices on one side and one vertices on the other & (2) A cut with m/K edges crossing the cut (notice such a cut can be found based on Prop 3.7). The better of these solutions has $\max\{(\frac{1}{r})(m + \text{Max Cut}(G)) - m, \frac{m}{K}\} \geq \frac{1}{r(K(1-1/r)+1)} \text{Max Cut}(G) \geq \frac{1}{1+K(r-1)} \text{Max Cut}(G)$ edges crossing the cut. Thus an r-approximate solution to Weighted Max Ones({XOR, XNOR$_4$}) yields a $(1 + K(r-1))$-approximate solution to Max Cut(G). Thus Max Cut(G) AP-reduces to Weighted Max Ones({XOR, XNOR$_4$}) and hence the latter is APX-hard. □

Lemma 6.10 *If $\mathcal{F}$ is affine but neither width-2 affine nor 1-valid, then $\mathcal{F} \overset{p}{\Longrightarrow}$ XNOR$_3$ or $\mathcal{F} \overset{p}{\Longrightarrow}$* {XOR, XNOR$_4$}.

Proof: Since $\mathcal{F}$ is affine but not of width-2, it can perfectly (and strictly) implement the function XOR$_p$ or XNOR$_p$ for some $p \geq 3$ (Lemma 4.18). Let $f \in \mathcal{F}$ be an affine constraint that is not 1-valid. We consider two possible cases depending on whether $\mathcal{F}$ is C-closed or not. If $g \in \mathcal{F}$ is not C-closed, then we find (by Lemma 4.7) that $\{f, g\}$ (and hence $\mathcal{F}$) perfectly implements some existential zero constraint. This case is covered in Claim 6.11 and we show that in this case $\mathcal{F}$ perfectly implements XNOR_3. In the other case, $\mathcal{F}$ is C-closed and hence (by Lemma 4.5) $\mathcal{F}$ perfectly implements the constraint XOR. This case is covered in Claim 6.12 and we show that in this case $\mathcal{F}$ perfectly implements either XNOR_3 or XNOR_4. This concludes the proof of Lemma 6.10 (modulo Claims 6.11 and 6.12). □

Claim 6.11 *If $\{f\}$ is an existential zero constraint and h is either the constraint XOR_p or XNOR_p for some $p \geq 3$, then $\{f, h\} \overset{p}{\Longrightarrow} \mathrm{XNOR}_3$.*

Proof: Since f is an existential zero constraint, the family $\{f, h\}$ can perfectly implement $\{f, h\}|_0$ (using Lemma 6.4). In particular, $\{f, h\}$ can implement the constraints $x_1 \oplus x_2 = b$ and $x_1 \oplus x_2 \oplus x_3 = b$ for some $b \in \{0, 1\}$. Notice finally that the constraints $x_1 \oplus x_2 \oplus y = b$ and $y \oplus x_3 = b$ form a perfect implementation of the constraint $x_1 \oplus x_2 \oplus x_3 = 0$. Thus $\{f, h\}$ perfectly implements the constraint XNOR_3. □

Claim 6.12 *If $f \in \{\mathrm{XOR}_p, \mathrm{XNOR}_p \mid p \geq 3\}$, then $\{f, \mathrm{XOR}\} \overset{p}{\Longrightarrow} \mathrm{XNOR}_3$ or $\{f, \mathrm{XOR}\} \overset{p}{\Longrightarrow} \mathrm{XNOR}_4$.*

Proof: Since XOR perfectly implements XNOR it suffices to prove this using the constraints $\{f, \mathrm{XOR}, \mathrm{XNOR}\}$.

W.l.o.g assume that f is the constraint XNOR, since else $\mathrm{XOR}_p(x_1, \ldots, x_{p-1}, y)$ and $\mathrm{XOR}(y, x_p)$ perfectly implement the constraint $\mathrm{XNOR}_p(x_1, \ldots, x_p)$.

Now if p is odd, then the constraints $\mathrm{XNOR}_p(x_1, \ldots, x_p)$ and $\mathrm{XNOR}(x_4, x_5)$, $\mathrm{XNOR}(x_6, x_7)$ and so on up to $\mathrm{XNOR}(x_{p-1}, x_p)$ perfectly implement the constraint $\mathrm{XNOR}_3(x_1, x_2, x_3)$.

Now if p is even, then the constraints $\mathrm{XNOR}_p(x_1, \ldots, x_p)$ and $\mathrm{XNOR}(x_5, x_6)$, $\mathrm{XNOR}(x_7, x_8)$ and so on up to $\mathrm{XNOR}(x_{p-1}, x_p)$ perfectly implement the constraint $\mathrm{XNOR}_4(x_1, x_2, x_3, x_4)$. □

Lemma 6.13 *If $\mathcal{F}$ is affine but neither width-2 affine nor 1-valid, then* MAX ONES$(\mathcal{F})$ *is* APX-*hard.*

Proof: By Lemma 6.6 we have WEIGHTED MAX ONES$(\mathcal{F})$ is in APX and thus (by Lemma 3.11) it suffices to show APX-hardness of WEIGHTED MAX ONES$(\mathcal{F})$. This now follows from Lemmas 3.9, 6.9,and 6.10. □

6.3.2 The poly-APX-hard case

This part turns out to be long and the bulk of the work will be done in Lemmas 6.16-6.21. We first describe the proof of the hardness result modulo the above lemmas. (Hopefully, the proof will also provide some motivation for the rest of the lemmas.)

Lemma 6.14 *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{WN}}\}$ but $\mathcal{F} \not\subseteq \mathcal{F}''$ for any $\mathcal{F}'' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}\}$, then* MAX ONES$(\mathcal{F})$ *is* poly-APX-*hard.*

Proof: As usual, by Lemmas 6.2 and 3.11, it suffices to show hardness of the weighted version. First we show in Lemma 6.15 that MAX ONES$(\{\mathrm{NAND}_k\})$ is poly-APX-hard for every $k \geq 2$. Thus our goal is to establish that any non 1-valid, non-affine, and non weakly positive constraint family can implement some NAND_k constraint. We do so in three phases.

The main complication here is that we don't immediately have a non 0-valid constraint to work with and thus we can't immediately reduce MAX ONES($\mathcal{F} \cup \{T, F\}$) to MAX ONES($\mathcal{F}$). So we go after something weaker and try to show that $\mathcal{F}$ can perfectly implement $\mathcal{F}|_{0,1}$. In Phase 3, (Lemmas 6.20 and 6.21) we show that this suffices. Lemma 6.20 uses the fact that $\mathcal{F}|_{0,1}$ is not weakly positive to implement either NAND$_2$ or XOR. In the former case we are done and in the latter case, Lemma 6.21 uses the fact that $\mathcal{F}|_{0,1}$ is not affine to implement NAND.

Thus our task reduces to that of showing that $\mathcal{F}$ can implement $\mathcal{F}|_{0,1}$. Part of this is easy. In Phase 1, we show that $\mathcal{F}$ implements every constraint in $\mathcal{F}|_0$. This is shown via Lemma 6.16 which shows that any family which is either 0-valid or 2CNF or weakly negative but not 1-valid or affine or weakly positive must have a non C-closed constraint. This along with the non 1-valid constraint allows it to implement every constraint in $\mathcal{F}|_0$ (by Lemmas 4.7 and 6.4). The remaining task for Phase 2 is to show that $\mathcal{F}|_0$ can implement $\mathcal{F}|_1$. If $\mathcal{F}$ also has a non 0-valid constraint then we are done since now we can implement all of $\mathcal{F}|_{0,1}$ (another application of Lemmas 4.7 and 6.4). Thus all lemmas in Phase 2, focus on $\mathcal{F}|_0$ for 0-valid constraint families $\mathcal{F}$. If $\mathcal{F}|_0$ is all 0-valid, then all we can show is that $\mathcal{F}|_0$ either implements NAND$_k$ for some k or OR$_{2,1}$ (Lemmas 6.17 and 6.18). The former is good, but the latter seems insufficient. In fact we are unable to implement $\mathcal{F}|_{0,1}$ in this case. We salvage the situation by reverting back to reductions. We AP-reduce the problem WEIGHTED MAX ONES($\mathcal{F}|_0 \cup \{\text{OR}_{2,1}\}$) to WEIGHTED MAX ONES($\mathcal{F}|_{0,1}$) (Lemma 6.19). This suffices to establish the poly-APX-hardness of WEIGHTED MAX ONES($\mathcal{F}$) since

$$\begin{aligned}\text{WEIGHTED MAX ONES}(\mathcal{F}|_{0,1}) &\leq_{\text{AP}} \text{WEIGHTED MAX ONES}(\mathcal{F}|_0 \cup \{\text{OR}_{2,1}\}) \\ &\leq_{\text{AP}} \text{WEIGHTED MAX ONES}(\mathcal{F})\end{aligned}$$

and the problem WEIGHTED MAX ONES($\mathcal{F}|_{0,1}$) is poly-APX-hard. □

Lemma 6.15 MAX ONES($\{\text{NAND}_k\}$) *is* poly-APX-*hard for every* $k \geq 2$.

Proof: We reduce from MAX CLIQUE, which is known to be poly-APX-hard. Given a graph G, construct a MAX ONES($\{f\}$) instance consisting of a variable for every vertex in G and the constraint f is applied to every subset of k vertices in G which does not induce a clique. It may be verified that the optimum number of ones in any satisfying assignment to the instance created in this manner is $\max\{k-1, \omega(G)\}$, where $\omega(G)$ is the size of the largest clique in G. Given a solution to the MAX ONES($\{f\}$) instance with $l \geq k$ ones, the set of vertices corresponding to the variables set to one form a clique of size l. If $l < k$, output any singleton vertex. Thus in all cases we obtain a clique of size at least $l/(k-1)$ vertices. Thus given an r-approximate solution to the MAX ONES($\{\text{NAND}_k\}$) problem, we can find a $(k-1)r$ approximate solution to MAX CLIQUE. Thus MAX CLIQUE is A-reducible to MAX ONES($\{\text{NAND}_k\}$). □

Phase 1: $\mathcal{F}$ implements $\mathcal{F}|_0$.

Lemma 6.16 *If* $\mathcal{F} \subseteq \mathcal{F}'$ *for some* $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{2\text{CNF}}, \mathcal{F}_{\text{WN}}\}$ *but* $\mathcal{F} \not\subseteq \{\mathcal{F}_1, \mathcal{F}_{2\text{A}}, \mathcal{F}_{\text{WP}}\}$ *then there exists a constraint in* $\mathcal{F}$ *that is not C-closed constraint.*

Proof: Notice that a C-closed 0-valid constraint is also 1-valid. Thus if $\mathcal{F}$ is 0-valid, then the non 1-valid constraint is not C-closed.

Next we claim that a C-closed weakly positive constraint f is also weakly negative. To do so, consider the constraint $\bar{f}$ given by $\bar{f}(x) = f(\bar{x})$. Notice that for a C-closed constraint $f = \bar{f}$. Suppose $f(x) = \bigwedge_j C_j(x)$ where the C_j's are weakly positive clauses. Then $\bar{f}(x)$ can be described as $\bigwedge_j \bar{C}_j(x)$ (where $\bar{C}_j(x) = C_j(\bar{x})$). But in this representation $\bar{f}$ (and thus f) is seen to be a weakly negative constraint, thereby verifying our claim. Thus if $\mathcal{F}$ is weakly negative but not weakly positive, the non weakly-positive constraint is the non C-closed constraint.

Finally we consider the case when f is a 2CNF formula. Again define $\bar{f}(x) = f(\bar{x})$ and $f'(x) = f(x)\bar{f}(x)$. Notice that $f' = f$ if f is C-closed. Again consider the CNF representation of $f = \bigwedge_j C_j(x)$ where the $C_j(x)$'s are clauses of f of length 2. Then $f'(x)$ can be expressed as $\bigwedge_j (C_j(x) \bigwedge \bar{C}_j(x))$. But $C_j \bigwedge \bar{C}_j$ are affine constraints of width 2! Thus f' and hence f is an affine width-2 constraint. Thus if $\mathcal{F}$ is 2CNF but not width-2 affine, the non width-2 affine constraint is the non C-closed constraint. □

Lemma 4.7 along with Lemma 6.4 suffice to prove that $\mathcal{F}$ implements $\mathcal{F}|_0$. We now move on to Phase 2.

<u>Phase 2:</u> From $\mathcal{F}|_0$ to $\mathcal{F}|_{0,1}$.

Recall that if $\mathcal{F}$ has a non 0-valid constraint, then by Lemmas 6.16, 4.7 and 6.4 it implements an existential one constraint and thus $\mathcal{F}|_{0,1}$. Thus all lemmas in this Phase assume $\mathcal{F}$ is 0-valid.

Lemma 6.17 *If f is 0-valid and not weakly positive, then $\{f\}|_0$ either perfectly implements* NAND_k *for some $k \geq 2$ or* $\text{OR}_{2,1}$ *or* XNOR.

Proof: Let $C = \neg x_1 \bigvee \cdots \bigvee \neg x_p \bigvee y_1 \bigvee \cdots \bigvee y_q$ be a maxterm in f with more than one negation i.e. $p \geq 2$. Since f is not weakly positive, Lemma 4.20 shows that such a maxterm exists. Substituting a 0 in place of variables $y_1, y_2, \ldots, y_q$, and existentially quantifying over all variables not in C, we get a constraint g such that $\neg x_1 \bigvee \neg x_2 \bigvee \cdots \bigvee \neg x_p$ is a maxterm in g. Consider an unsatisfying assignment s for g with the smallest number of 1's and let k denote the number of 1's in s; we know $k > 0$ since the original constraint is 0-valid. W.l.o.g. assume that s assigns value 1 to the variables $x_1, x_2, \ldots, x_k$ and 0 to the remaining variables. It is easy to see that by fixing the variables $x_{k+1}, x_{k+2}, \ldots, x_p$ to 0, we get a constraint $g' = (\neg x_1 \bigvee \neg x_2 \bigvee \cdots \bigvee \neg x_k)$. If $k > 1$, then this perfectly implements the constraint $\text{NAND}_k(x_1, \ldots, x_k)$ and we are done.

Otherwise $k = 1$, i.e. there exists an unsatisfying assignment s which assigns value 1 to exactly one of the x_i's, say x_1. Now consider a satisfying assignment s' which assigns 1 to x_1 and has a minimum number of 1's among all assignments which assign 1 to x_1. The existence of such an assignment follows from C being a maxterm in g. For instance, the assignment $1^{p-1}0$ is a satisfying assignment which satisfies such a property. W.l.o.g. assume that $s' = 1^i 0^{p-i}$. Thus the constraint g looks as follows:

	x_1	x_2	$x_3 ... x_i$	$x_{i+1} ... x_p$	$g()$
s_1	0	0	00...0	00...0	1
s_2	1	0	00...0	00...0	0
$s' = s_3$	1	1	11...1	00...0	1
s_4	0	1	__..._	00...0	?

Existential quantification over the variables $x_3, x_4, \ldots, x_i$ and fixing the variables x_{i+1} through x_p to 0 yields a constraint g' which is either $\text{OR}_{2,1}(x_2, x_1)$ or $\text{XNOR}(x_1, x_2)$. The lemma follows. □

Now we consider the case where we can implement the function XNOR and show that in this case we can either perfectly implement NAND or $\text{OR}_{2,1}$. In the former case we are done and for the latter case we show in Lemma 6.19 that WEIGHTED MAX ONES($\mathcal{F}|_1$) is AP-reducible to WEIGHTED MAX ONES($\mathcal{F} \cup \{\text{OR}_{2,1}\}$).

Lemma 6.18 *If f is 0-valid but not affine then $\{f\}|_0 \cup \{\text{XNOR}\}$ perfectly implements either* NAND *or the constraint* $\text{OR}_{2,1}$.

Proof: Corollary 4.16 shows that if f is not affine then there exist two satisfying assignments s_1 and s_2 such that $s_1 \oplus s_2$ is not a satisfying assignment for f. Reorder the variables such that $Z(s_1) \cap Z(s_2) = \{x_1, \ldots, x_p\}$, $Z(s_1) \cap O(s_2) = \{x_{p+1}, \ldots, x_q\}$, $O(s_1) \cap Z(s_2) = \{x_{q+1}, \ldots, x_r\}$ and $O(s_1) \cap O(s_2) = \{x_{r+1}, \ldots, x_k\}$. Using the fact that f is 0-valid, we find that f looks as follows:

	$x_1 \ldots x_p$	$x_{p+1} \ldots x_q$	$x_{q+1} \ldots x_r$	$x_{r+1} \ldots x_k$	$g(\mathbf{x})$
	00...0	00...0	00...0	00...0	1
s_1	00...0	00...0	11...1	11...1	1
s_2	00...0	11...1	00...0	11...1	1
$s_1 \oplus s_2$	00...0	11...1	11...1	00...0	0

Consider the collection of constraints:

1. $f(0, \ldots, 0, x_{p+1}, \ldots, x_k)$.
2. $\text{XNOR}(x, x_i)$ for $i \in Z(s_1) \cap O(s_2)$.
3. $\text{XNOR}(y, x_i)$ for $i \in O(s_1) \cap Z(s_2)$.
4. $\text{XNOR}(z, x_i)$ for $i \in O(s_1) \cap O(s_2)$.

Existentially quantifying over the variables $x_{p+1}, \ldots, x_k$ we obtain an implementation of a constraint $h(x, y, z)$ such that $h(000) = h(011) = h(101) = 1$ and $h(110) = 0$. Furthermore, by restricting more of the variables in (1) above to 0, we get a perfect implementation of any constraint in $\{h\}|_0$. Using Claim 6.22 again we get that $\{h\}|_0$ can implement either NAND or $\text{OR}_{2,1}$, and thus we are done. □

Finally we show how to use $\text{OR}_{2,1}$ constraints.

Lemma 6.19 *If $\mathcal{F}$ is 0-valid then* Weighted Max Ones($\mathcal{F}|_1$) *AP-reduces to* Weighted Max Ones($\mathcal{F} \cup \{\text{OR}_{2,1}\}$).

Proof: We show something stronger, namely, Weighted Max Ones($\mathcal{F} \cup \{T\}$) AP-reduces to Weighted Max Ones($\mathcal{F} \cup \{\text{OR}_{2,1}\}$). This suffices since T is an existential one constraint and thus $\mathcal{F} \cup \{T\}$ can perfectly implement $\mathcal{F}|_1$.

Given an instance $\mathcal{I}$ of Weighted Max Ones($\mathcal{F} \cup \{T\}$) construct an instance $\mathcal{I}'$ of Weighted Max Ones($\mathcal{F} \cup \{\text{OR}_{2,1}\}$) as follows. The variable set of $\mathcal{I}'$ is the same as that of $\mathcal{I}$. Every constraint from $\mathcal{F}$ in $\mathcal{I}$ is also included in $\mathcal{I}'$. The only remaining constraints are of the form $T(x_i)$ for some variables x_i. We simulate this constraint in $\mathcal{I}'$ with $n - 1$ constraints of the form $\text{OR}_{2,1}(x_j, x_i)$ (i.e., $\neg x_j \bigvee x_i$) for every $j \in [n]$, $j \neq i$. Every non-zero solution to the resulting instance $\mathcal{I}'$ is also a solution to $\mathcal{I}$, since the solution must have $x_i = 1$ or else have $x_j = 0$ for every $j \neq i$. Thus the resulting instance of Max Ones($\mathcal{F} \cup \{\text{OR}_{2,1}\}$) has the same objective function and the same feasible space and is hence at least as hard as the original problem. □

This concludes Phase 2.

Phase 3: $\mathcal{F}|_{0,1}$ implements NAND.

Lemma 6.20 *If f is not weakly positive, then $\{f\}|_{0,1}$ perfectly implements either* XOR *or* NAND.

Proof: Let $C = (\neg x_1 \bigvee \cdots \bigvee \neg x_p \bigvee y_1 \bigvee \cdots \bigvee y_q)$ be a maxterm in f with more than one negation i.e. $p \geq 2$. Substituting a 1 for variables $x_3, \ldots, x_p$, a 0 for variables $y_1, \ldots, y_q$, and existentially quantifying over all variables not in C, we get a constraint f' such that $f'(11) = 0$, $f'(01) = f'(10) = 1$ (These three properties follow from the definition of a maxterm). Depending on whether $f'(00)$ is 0 or 1 we get the function XOR or NAND, respectively. □

Lemma 6.21 *If g is a non-affine constraint, then $\{g, \text{XOR}\}|_{0,1} \overset{p}{\Longrightarrow}$* NAND.

Proof: Again it suffices to consider $\{g, \text{XOR}, \text{XNOR}\}|_{0,1}$. Let g be of arity k. By Lemma 4.15 we find that there must exist assignments s_1, s_2 and s_3 satisfying g such that $s_1 \oplus s_2 \oplus s_3$ does not satisfy g. Partition the set $[k]$ into up to eight equivalence classes $S_{b_1 b_2 b_3}$ for $b_1, b_2, b_3 \in \{0, 1\}$ such that for any index $i \in S_{b_1 b_2 b_3}$, $(s_j)_i = b_j$ for every $j \in \{1, 2, 3\}$. (Refer to Figure 1 below.)

	S_{000}	S_{001}	S_{010}	S_{011}	S_{100}	S_{101}	S_{110}	S_{111}	$g(\mathbf{x})$
s_1	0...0	0...0	0...0	0...0	1...1	1...1	1...1	1...1	1
s_2	0...0	0...0	1...1	1...1	0...0	0...0	1...1	1...1	1
s_3	0...0	1...1	0...0	1...1	0...0	1...1	0...0	1...1	1
$s_1 \oplus s_2 \oplus s_3$	0...0	1...1	1...1	0...0	1...1	0...0	0...0	1...1	0

Figure 1: Partition of inputs to g

W.l.o.g. assume that $S_{000} = \{1, \ldots, p\}$ and $S_{111} = \{q+1, \ldots, k\}$. Notice that the assignment of a variable in $S_{b_1 b_2 b_3}$ under assignment $s_1 \oplus s_2 \oplus s_3$ is also fixed (to $b_1 \oplus b_2 \oplus b_3$). Now consider the collection of constraints

1. $g(0, \ldots, 0, x_{p+1} \ldots, x_q, 1, \ldots, 1)$.
2. XNOR(x, x_i) for $i \in S_{001}$.
3. XNOR(y, x_i) for $i \in S_{010}$.
4. XNOR(z, x_i) for $i \in S_{011}$.
5. XOR(z, x_i) for $i \in S_{100}$.
6. XOR(y, x_i) for $i \in S_{101}$.
7. XOR(x, x_i) for $i \in S_{110}$.

By existentially quantifying over the variables $x_{p+1}, \ldots, x_q$ we perfectly implement a constraint $h(x, y, z)$ with the following properties: $h(000) = h(011) = h(101) = 1$ and $h(110) = 0$. Furthermore, by restricting more variables in condition (1) above, we can actually implement any function in the set $\{h\}|_{0,1}$. Claim 6.22 now shows that for any such function h, the set $\{h\}|_0$ perfectly implements either $\text{OR}_{2,1}$ or NAND. In the latter case we are done. In the former case, notice that the constraints $\text{OR}_{2,1}(x, z)$ and $\text{XOR}(z, y)$ perfectly implement the constraint $\text{NAND}(x, y)$ so in this case too we are done (modulo Claim 6.22). □

Claim 6.22 *If h is ternary function such that $h(000) = h(011) = h(101) = 1$ and $h(110) = 0$, then $\{h\}|_0 \overset{p}{\Longrightarrow}$ NAND or $\{h\}|_0 \overset{p}{\Longrightarrow} \text{OR}_{2,1}$.*

x \ yz	00	01	11	10
0	1	-	1	A
1	B	1	-	0

Figure 2: Truth-table of the constraint $h(x, y, z)$

Proof:

Figure 2 describes the truth table for the function h. The undetermined values of interest to us are indicated in the table by A and B. The following analysis shows that for every possible value of A and B, we can perfectly implement either NAND or $\mathrm{OR}_{2,1}$

$$\begin{aligned} A = 0 &\implies \exists\, x\ h(x, y, z) = \neg y \bigvee z \\ B = 0 &\implies \exists\, y\ h(x, y, z) = \neg x \bigvee z \\ A = 1, B = 1 &\implies h(x, y, 0) = \neg x \bigvee \neg y \end{aligned}$$

Thus in each case we perfectly implement either the constraint NAND or $\mathrm{OR}_{2,1}$. □

6.3.3 Remaining cases

We now prove that if $\mathcal{F}$ is not strongly decidable, then deciding if there exists a non-zero solution is NP-hard. This is shown in Lemma 6.23. The last of the hardness results, claiming that finding a feasible solution is NP-hard if $\mathcal{F}$ is not 0-valid or 1-valid or 2cnf or weakly positive or weakly negative or linear, follows directly from Schaefer's theorem (Theorem 2.10).

Lemma 6.23 *If $\mathcal{F} \not\subseteq \mathcal{F}'$, for any $\mathcal{F}' \in \{\mathcal{F}_{\mathrm{S0}}, \mathcal{F}_1, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then the problem of finding solutions of non-zero value to a given instance of (unweighted)* Max Ones$(\mathcal{F})$ *is NP-hard.*

Proof: Assume, for simplicity, that all constraints of $\mathcal{F}$ have arity k. Given a constraint $f : \{0,1\}^k \to \{0,1\}$ and an index $i \in [k]$, let $f{\downarrow_i}$ be the constraint mapping $\{0,1\}^{k-1}$ to $\{0,1\}$ given by

$$f{\downarrow_i}(x_1, \ldots, x_k) \stackrel{\mathrm{def}}{=} f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_k) \wedge f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_k).$$

Let $\mathcal{F}'$ be the set of constraints defined as follows:

$$\mathcal{F}' \stackrel{\mathrm{def}}{=} \mathcal{F} \cup \{f{\downarrow_i} \ \mid\ f \in \mathcal{F}, i \in [k]\}.$$

We will show that deciding Sat$(\mathcal{F}')$ is NP-hard, and that the problem of deciding Sat$(\mathcal{F}')$ reduces to finding non-zero solutions to Max Ones$(\mathcal{F})$.

First observe that $\mathcal{F}' \not\subseteq \mathcal{F}''$, for any $\mathcal{F}'' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$. In particular it is not 0-valid, since $\mathcal{F}$ is not strongly 0-valid. Hence, once again applying Schaefer's result, we find that deciding Sat$(\mathcal{F}')$ is NP-hard.

Given an instance of Sat$(\mathcal{F}')$ on n variables $\mathbf{x}$ with m constraints $\mathbf{C}$, with $C_1, \ldots, C_{m'} \in \mathcal{F}$ and $C_{m'+1}, \ldots, C_m \in \mathcal{F}' \setminus \mathcal{F}$, consider the instance of Max Ones$(\mathcal{F})$ defined on variable set

$$w_1, \ldots, w_{k+1}, y_1, \ldots, y_n, z_1, \ldots, z_n$$

with the following constraints:

1. Let f be a non-1-valid constraint in $\mathcal{F}$. We introduce the constraint $f(w_1, \ldots, w_k)$.
2. For every constraint $C_i(v_{i_1}, \ldots, v_{i_k})$, $1 \leq i \leq m'$, we introduce two constraints $C_i(y_{i_1}, \ldots, y_{i_k})$ and $C_i(z_{i_1}, \ldots, z_{i_k})$.
3. For every constraint $C_i(v_{i_1}, \ldots, v_{i_{k-1}})$, $m' + 1 \leq i \leq m$, we introduce $2(n + k + 1)$ constraints. For simplicity of notation, let $C_i(v_{i_1}, \ldots, v_{i_{k-1}}) == g(1, v_{i_1}, \ldots, v_{i_{k-1}}) \wedge g(0, v_{i_1}, \ldots, v_{i_{k-1}})$ where $g \in \mathcal{F}$. The $2(n + k + 1)$ constraints are:
   - $g(w_j, y_{i_1}, \ldots, y_{i_{k-1}})$, for $1 \leq j \leq k + 1$.
   - $g(z_j, y_{i_1}, \ldots, y_{i_{k-1}})$, for $1 \leq j \leq n$.
   - $g(w_j, z_{i_1}, \ldots, z_{i_{k-1}})$, for $1 \leq j \leq k + 1$.
   - $g(y_j, z_{i_1}, \ldots, z_{i_{k-1}})$, for $1 \leq j \leq n$.

We now show that the instance of Max Ones($\mathcal{F}$) created above has a non-zero satisfying assignment if and only if the instance of Sat($\mathcal{F}'$) has a satisfying assignment. Let $s = s_1 s_2 \ldots s_k$ be a satisfying assignment for the non 1-valid constraint f chosen above. First if $v_1, \ldots, v_n$ form a satisfying assignment to the instance of Sat($\mathcal{F}'$), then we claim that the assignment $w_j = s_j$ for $1 \leq j \leq k$, $w_{k+1} = 1$ and $y_j = z_j = v_j$ for $1 \leq j \leq n$ is a satisfying assignment to the instance of Max Ones($\mathcal{F}$) which has at least one 1 (namely w_{k+1}). Conversely, let some non-zero setting $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n, z_1, \ldots, z_n$ satisfy the instance of Max Ones($\mathcal{F}$). W.l.o.g. assume that one of the variable $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n$ is a 1. Then we claim that the setting $v_j = z_j$, $1 \leq j \leq n$ satisfies the instance of Sat($\mathcal{F}'$). It is easy to see that the constraints $C_i(v_{i_1}, \ldots, v_{i_k})$, $1 \leq i \leq m'$, are satisfied. Now consider a constraint $C_i(v_{i_1}, \ldots, v_{i_{k-1}}) = g(0, v_{i_1}, \ldots, v_{i_{k-1}}) \wedge g(1, v_{i_1}, \ldots, v_{i_{k-1}})$. Since at least one of the variables in the set $w_1, \ldots, w_k$ is a 0 and at least one of the variables in the set $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n$ is 1, we know that both $g(0, z_{i_1}, \ldots, z_{i_{k-1}})$ and $g(1, z_{i_1}, \ldots, z_{i_{k-1}})$ are satisfied and hence $C_i(v_{i_1}, \ldots, v_{i_{k-1}}) = 1$. Thus the reduced instance of Max Ones($\mathcal{F}$) has a non-zero satisfying assignment if and only if the instance of Sat($\mathcal{F}'$) is satisfiable. □

7 Classification of Min CSP

7.1 Preliminary results

We start with a simple equivalence between the complexity of the (Weighted) Min CSP problem for a function family and the family of functions obtained by complementing the 0's and 1's in its domain. Recall that for a function f, we defined f^- to be the function $f^-(\mathbf{x}) = f(\mathbf{1} - \mathbf{x})$, and for a function family $\mathcal{F}$, we defined $\mathcal{F}^- = \{f^- \mid f \in \mathcal{F}\}$.

Proposition 7.1 *For every constraint family $\mathcal{F}$,* (Weighted) Min CSP($\mathcal{F}$) *is AP-reducible to* (Weighted) Min CSP($\mathcal{F}^-$).

Proof: The reduction substitutes every constraint $f(\mathbf{x})$ from $\mathcal{F}$ with the constraint $f^-(\mathbf{x})$ from $\mathcal{F}^-$. A solution for the latter problem is converted into a solution for the former one by complementing the value of each variable. The transformation preserves the cost of the solution. □

Proposition 7.2 *If $\mathcal{F}$ is decidable then* Weighted Min CSP($\mathcal{F}$) *is in* poly-APX *and is AP-reducible to* Min CSP($\mathcal{F}$).

Proof: Given an instance $\mathcal{I}$ of WEIGHTED MIN ONES($\mathcal{F}$) with constraints $C_1, \ldots, C_m$ sorted in order of decreasing weight $w_1 \geq \cdots \geq w_m$. Let j be the largest index such that the constraints $C_1, \ldots, C_j$ are simultaneously satisfiable. Notice that j is computable in polynomial time and an assignment $\mathbf{a}$ satisfying $C_1, \ldots, C_j$ is computable in polynomial time. Then the solution $\mathbf{a}$ is an m-approximate solution to $\mathcal{I}$, since every solution must fail to satisfy at least one of the constraints $C_1, \ldots, C_{j+1}$ and thus have an objective of at least w_{j+1}, while $\mathbf{a}$ achieves an objective of at most $\sum_{i=j+1}^{m} w_i \leq m w_{j+1}$. Thus we conclude that WEIGHTED MIN CSP($\mathcal{F}$) is in poly-APX. The second part of the proposition follows by Lemma 3.11. □

7.2 Containment Results (Algorithms) for MIN CSP

We now show the containment results described in Theorem 2.13. Most results described here are simple containment results which follow easily from the notion of a "basis". The more interesting result here is a constant factor approximation algorithm for IHS-B which is presented in Lemma 7.3.

Recall that the classes contained in PO have already been dealt with in Section 5.1. We now move on to APX-containment results.

Lemma 7.3 *If $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{IHS}}$, then* WEIGHTED MIN CSP($\mathcal{F}$) $\in$ APX.

Proof: By Propositions 3.4 and 7.1 it suffices to prove the lemma for the problem WEIGHTED MIN CSP(IHS-B), where IHS-$B = \{\mathrm{OR}_k | k \in [B]\} \cup \{\mathrm{OR}_{2,1}, F\}$. We will show that for every B, WEIGHTED MIN CSP(IHS-B) is $B+1$-approximable.

Given an instance $\mathcal{I}$ of WEIGHTED MIN CSP(IHS-B) on variables $x_1, \ldots, x_n$ with constraints $C_1, \ldots, C_m$ with weights $w_1, \ldots, w_m$, we create a linear program on variables $y_1, \ldots, y_n$ (corresponding to the Boolean variables $x_1, \ldots, x_n$) and variables $z_1, \ldots, z_m$ (corresponding to the constraints $C_1, \ldots, C_m$). For every constraint C_j in the instance $\mathcal{I}$ we create a LP constraint using the following transformation rules:

$$\begin{array}{lclcrcl} C_j & : & x_{i_1} \bigvee \cdots \bigvee x_{i_k}, \text{ for } k \leq B & \to & z_j + y_{i_1} + \cdots + y_{i_k} & \geq & 1 \\ C_j & : & \neg x_{i_1} \bigvee x_{i_2} & \to & z_j + (1 - y_{i_1}) + y_{i_2} & \geq & 1 \\ C_j & : & \neg x_{i_1} & \to & z_j + (1 - y_{i_1}) & \geq & 1 \end{array}$$

In addition we add the constraints $0 \leq z_j, y_i \leq 1$ for every i, j. It may be verified that any integer solution to the above LP corresponds to an assignment to the MIN CSP problem with the variable z_j set to 1 if the constraint C_j is not satisfied. Thus the objective function for the LP is to minimize $\sum_j w_j z_j$.

Given any feasible solution vector $y_1, \ldots, y_n, z_1, \ldots, z_m$ to the LP above, we show how to obtain a 0/1 vector $y''_1, \ldots, y''_n, z''_1, \ldots, z''_m$ that is also feasible such that $\sum_j w_j z''_j \leq (B+1) \sum_j w_j z_j$.

First we set $y'_i = \min\{1, (B+1) y_i\}$ and $z'_j = \min\{1, (B+1) z_j\}$. Observe that the vector $y'_1, \ldots, y'_n$, $z'_1, \ldots, z'_m$ is also feasible and gives a solution of value at most $(B+1) \sum_j w_j z_j$. We now show how to get an *integral* solution whose value is at most $\sum_j w_j z'_j \leq (B+1) \sum_j w_j z_j$. For this part we first set $y''_i = 1$ if $y'_i = 1$ and $z''_j = 1$ if $z'_i = 1$. Now we remove every constraint in the LP that is made redundant. Notice in particular that every constraint of type (1) is now redundant (either z''_j or one of the y''_i's has already been set to 1 and hence the constraint will be satisfied by any assignment to the remaining variables). We now observe that, on the remaining variables, the LP constructed above reduces to the following

$$\begin{array}{rrcl} \textsf{Minimize} & \sum_j w_j z_j & & \\ \textsf{Subject to} & y_{i_2} - y_{i_1} + z_j & \geq & 0 \\ & y_{i_2} + z_j & \geq & 1 \\ & -y_{i_1} + z_j & \geq & 0 \end{array}$$

with the y_i''s and z_j''s forming a feasible solution to the above LP. Notice further that every z_j occurs in at most one constraint above. Thus the above LP represents s-t min cut problem, and therefore has an optimal integral solution. We set z_j'''s and y_i'' to such an integral optimal solution. Notice that the solution thus obtained is integral and satisfies $\sum_j w_j z_j'' \leq \sum_j w_j z_j' \leq (B+1)\sum_j w_j z_j$. □

Lemma 7.4 *For any family $\mathcal{F} \subseteq \mathcal{F}_{2\mathrm{A}}$,* Weighted Min CSP($\mathcal{F}$) *A-reduces to* Min CSP(XOR).

Proof: First we will argue that the family $\mathcal{F}' = \{\mathrm{XOR}, T, F\}$ perfectly implements $\mathcal{F}$. By Proposition 3.4 it suffices to implement the basic width-2 affine functions: namely, the functions XOR, XNOR, T and F. Every function except XNOR is already present in $\mathcal{F}'$ and by Proposition 3.3 XOR perfectly implements XNOR.

We conclude by observing that the family {XOR} is neither 0-valid nor 1-valid and hence, by Lemma 5.7, Weighted Min CSP($\mathcal{F}'$) A-reduces to Weighted Min CSP(XOR). Finally the weights can be removed using Proposition 7.2. □

The following lemmas show reducibility to Min 2CNF Deletion, Nearest Codeword and Min Horn Deletion.

Lemma 7.5 *For any family $\mathcal{F} \subseteq \mathcal{F}_{2\mathrm{CNF}}$, the family* $\{\mathrm{OR}, \mathrm{NAND}\} \overset{p}{\Longrightarrow} \mathcal{F}$ *and hence* Weighted Min CSP($\mathcal{F}$)$\leq_{\mathrm{A}}$ Min 2CNF Deletion.

Proof: Again it suffices to consider the basic constraints of $\mathcal{F}$ and this is some subset of

$$\{\mathrm{OR}_{2,0}, \mathrm{OR}_{2,1}, \mathrm{OR}_{2,2}, T, F\}.$$

The family $\{\mathrm{OR}, \mathrm{NAND}\}$ contains the first and the third function. Since it contains a non 0-valid function, a non 1-valid function and a non C-closed function, it can also implement T and F (by Lemma 4.6. This leaves the function $\mathrm{OR}_{2,1}$ which is implemented by the constraints $\mathrm{NAND}(x, z_{\mathrm{AUX}})$ and $\mathrm{OR}(y, z_{\mathrm{AUX}})$ (on the variables x and y). The A-reduction now follows from Lemma 3.10. □

Lemma 7.6 *For any family $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{A}}$, the family* $\{\mathrm{XOR}_3, \mathrm{XNOR}_3\}$ *perfectly implements every function in $\mathcal{F}$. and thus* Weighted Min CSP($\mathcal{F}$) $\leq_{\mathrm{A}}$ Nearest Codeword.

Proof: It suffices to show implementation of the basic affine constraints, namely, constraints of the form XNOR_p and XOR_q for every $p, q \geq 1$. We focus on the former type as the implementation of the latter is analogous. First, we observe that the constraint $\mathrm{XNOR}(x_1, x_2)$ is perfectly implemented by the constraints $\{\mathrm{XNOR}_3(x_1, x_2, z_1), \mathrm{XNOR}_3(x_1, x_2, z_2), \mathrm{XNOR}_3(x_1, x_2, z_3), \mathrm{XNOR}_3(z_1, z_2, z_3)\}$ Next, the constraint $F(x_1)$ can be perfectly implemented by $\{\mathrm{XNOR}(x_1, z_1), \mathrm{XNOR}(x_1, z_2), \mathrm{XNOR}(x_1, z_3), \mathrm{XNOR}_3(z_1, z_2, z_3)\}$ Finally, the constraint $\mathrm{XNOR}_p(x_1, \ldots, x_p)$ for any $p > 3$ can be implemented as follows. We introduce the following set of constraints using the auxiliary variables $z_1, z_2, ..., z_{p-2}$ and the set of constraints:

$$\{\mathrm{XNOR}_3(x_1, x_2, z_1), \mathrm{XNOR}_3(z_1, x_3, z_2), \mathrm{XNOR}_3(z_2, x_4, z_3), \ldots, \mathrm{XNOR}_3(z_{p-2}, x_{p-1}, x_p)\}$$

□

Lemma 7.7 *For any family $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WP}}$, we have $\{\mathrm{OR}_{3,1}, T, F\} \overset{p}{\Longrightarrow} \mathcal{F}$ and thus* Weighted Min CSP($\mathcal{F}$) $\leq_{\mathrm{A}}$ Min Horn Deletion.

Proof: As usual, it suffices to perfectly implement every function in the basis $\{\mathrm{OR}_k \mid k \geq 1\} \cup \{\mathrm{OR}_{k,1} \mid k \geq 1\}$. The constraint $\mathrm{OR}(x, y)$ is implemented by the constraints $\mathrm{OR}_{3,1}(a, x, y)$ and $T(a)$. $\mathrm{OR}_{2,1}(x, y)$ is implemented by $\mathrm{OR}_{3,1}(x, y, a)$ and $F(a)$. The implementation of $\mathrm{OR}_3(x, y, z)$ is $\mathrm{OR}(x, a)$ and $\mathrm{OR}_{3,1}(a, y, z)$ (the constraint $\mathrm{OR}(x, a)$, in turn, may be implemented with the already shown method). Thus every k-ary constraint, for $k \leq 3$ can be perfectly implemented by the family $\{\mathrm{OR}_{3,1}, T, F\}$). For $k \geq 4$, we use the textbook reduction from Sat to 3Sat (see e.g. [19, Page 49]) and we observe that when applied to k-ary weakly positive constraints it yields a perfect implementation using only 3-ary weakly positive constraints. □

To conclude this section we describe the trivial approximation algorithms for Nearest Codeword and Min Horn Deletion. They follow easily from Proposition 7.2 and the fact that both families are decidable.

Corollary 7.8 (to Proposition 7.2) Min Horn Deletion *and* Nearest Codeword *are in* poly-APX.

7.3 Hardness Results (Reductions) for Min CSP

Lemma 7.9 (**APX-hardness**) *If $\mathcal{F} \not\subseteq \mathcal{F}'$, for $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}\}$, and then* Min CSP($\mathcal{F}$) *is* APX*-hard.*

Proof: The proof essentially follows from Lemma 5.8 in combination with Proposition 3.7. We show that for every $\mathcal{F}$ Max CSP($\mathcal{F}$) AP-reduces to Min CSP($\mathcal{F}$). Let $\mathcal{I}$ be an instance of Max CSP($\mathcal{F}$) on n variables and m constraints. Let $\mathbf{x}'$ be a solution satisfying m/k constraints that can be found in polynomial time (by Proposition 3.7). Let $\mathbf{x}''$ be an r-approximate solution to the same instance $\mathcal{I}$ viewed as an instance of Min CSP($\mathcal{F}$). If $\textsc{opt}$ is the optimum solution to the maximization problem $\mathcal{I}$, then $\mathbf{x}''$ satisfies at least $m - r(m - \textsc{opt}) = r\textsc{opt} - (r-1)m$ constraints. Thus the better of the two solutions is an r'-approximate solution to the instance $\mathcal{I}$ of Max CSP($\mathcal{F}$), where

$$\begin{aligned} r' &\leq \frac{\textsc{opt}}{\max\{m/k, r\textsc{opt} - (r-1)m\}} \\ &\leq \frac{((r-1)k+1)\textsc{opt}}{(r-1)k(m/k) + r\textsc{opt} - (r-1)m} \\ &= \frac{1 + (r-1)k}{r} \\ &\leq 1 + (r-1)k \end{aligned}$$

Thus Max CSP($\mathcal{F}$) AP-reduces to Min CSP($\mathcal{F}$). The lemma follows from the APX-hardness of Max CSP($\mathcal{F}$) (Lemma 5.8). □

Lemma 7.10 (**Min UnCut-hardness**) *If $\mathcal{F} \not\subseteq \mathcal{F}'$, for $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}, \mathcal{F}_{\mathrm{IHS}}\}$, and $\mathcal{F} \subseteq \mathcal{F}_{2\mathrm{A}}$ then* Min CSP($\mathcal{F}$) *is* Min UnCut*-hard.*

Proof: Recall that Min UnCut-hardness requires that Min CSP(XOR) be A-reducible to Min CSP($\mathcal{F}$).

Let $f \in \mathcal{F}$. Consider (all) the minimally dependent sets of f. By Lemma 4.22 all such sets are of cardinality at most 2. For a minimally dependent set $\{i, j\}$ let

$$f_{i,j}(x_i, x_j) \stackrel{\text{def}}{=} \exists x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k \text{ s.t. } f(x_1, \ldots, x_k).$$

By Lemma 4.17 all the $f_{i,j}$'s are affine and thus must be one of the functions $T(x_i)$, $F(x_i)$ $\text{XOR}(x_i, x_j)$ or $\text{XNOR}(x_i, x_j)$. Furthermore f can be expressed as the conjunction of $f_{i,j}$'s over all the minimally dependent sets. It follows that there exist i, j such that $f_{i,j}(x_i, x_j) = \text{XOR}(x_i, x_j)$. (Otherwise f would be a conjunction of T, F and XNOR functions, all of which are in $\mathcal{F}_{\text{IHS}}$, and thus f would also be in $\mathcal{F}_{\text{IHS}}$.) Thus we conclude that f implements XOR and by Lemma 3.10 we conclude that MIN CSP(XOR) is A-reducible to MIN CSP($\mathcal{F}$) as desired. □

For the MIN 2CNF DELETION-hardness proof, we need the following three simple lemmas.

Lemma 7.11 *If f is a* 2CNF *function which is not width-2 affine, then* $f \overset{p}{\Longrightarrow} \text{OR}_{2,l}$ *for some* $l \in \{0, 1, 2\}$.

Proof: For $i, j \in [k]$, let

$$f_{i,j}(x_i, x_j) \stackrel{\text{def}}{=} \exists x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k \text{ s.t. } f(x_1, \ldots, x_k).$$

Recall that f can be expressed as the conjunction of $f_{i.j}$'s over all its maxterms and by Lemma 4.21, all the maxterms of f's have at most 2 literals in them. Thus $f(x_1, \ldots, x_k)$ can be expressed as $\bigwedge_{i,j \in [k]} f_{i,j}(x_i, x_j)$. It follows that some $f_{i,j}$ must be one of the functions $\text{OR}_{2,0}$, $\text{OR}_{2,1}$ or $\text{OR}_{2,2}$ (all other functions on 2 variables are affine). Thus existentially quantifying over all variables other than x_i and x_j, f perfectly implements $\text{OR}_{2,l}$ for some $l \in \{0, 1, 2\}$. □

Lemma 7.12 *If* $f \in \mathcal{F}_{\text{2CNF}}$ *is not in IHS-B, then* $f \overset{p}{\Longrightarrow} \text{XOR}$.

Proof: Once again we use the fact that f can be expressed as $\bigwedge_{i,j \in [k]} f_{i,j}(x_i, x_j)$, where $f_{i,j}$ is the function obtained from f by existentially quantifying over all variables other than x_i and x_j. It follows that one of the $f_{i,j}$'s must be NAND or XOR, since all the other functions on two variables are in IHS-$B+$. In the latter case we are done, else we use the fact that f is not in IHS-$B-$ to conclude that f perfectly implements OR or XOR. In the latter case again we are done else we use the fact that f perfectly implements both the functions NAND and OR, and that $\text{NAND}(x, y)$ and $\text{OR}(x, y)$ perfectly implement $\text{XOR}(x, y)$, to conclude that in this case too, the function f perfectly implements XOR. □

Lemma 7.13 *If f is the function* $\text{OR}_{2,l}$ *for some* $l \in \{0, 1, 2\}$ *then* $\{f, \text{XOR}\} \overset{p}{\Longrightarrow} \{\text{OR}, \text{NAND}\}$.

Proof: The lemma follows from the fact that the function XOR essentially allows us to negate literals. For example, given the function $\text{OR}_{2,1}(x, y)$ and XOR, the applications $\text{OR}_{2,1}(x, z_{\text{AUX}})$ and $\text{XOR}(z_{\text{AUX}}, y)$ perfectly and strictly implement the function $\text{NAND}(x, y)$. Other implementations are obtained similarly. □

Lemma 7.14 (MIN 2CNF DELETION-hardness) *If* $\mathcal{F} \not\subseteq \mathcal{F}'$, *for* $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{\text{2M}}, \mathcal{F}_{\text{IHS}}, \mathcal{F}_{\text{2A}}\}$, *and* $\mathcal{F} \subseteq \mathcal{F}_{\text{2CNF}}$ *then* MIN CSP($\mathcal{F}$) *is* MIN 2CNF DELETION-*hard.*

Proof: By Lemmas 7.11 and 7.12, $\mathcal{F}$ implements one of the functions $\text{OR}_{2,l}$ for $l \in \{0, 1, 2\}$ and the function XOR. By Lemma 7.13 this suffices to implement the family $\{\text{NAND}, \text{OR}\}$. Thus by Lemma 3.10 we conclude that MIN CSP($\{\text{OR}, \text{NAND}\}$) A-reduces to MIN CSP($\mathcal{F}$). □

Lemma 7.15 *If $\mathcal{F} \subseteq \mathcal{F}_{\rm A}$ but $\mathcal{F} \not\subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{\rm 2M}, \mathcal{F}_{\rm IHS}, \mathcal{F}_{\rm 2A}\}$, then* Min CSP($\mathcal{F}$) *is* Nearest Codeword*-hard.*

Proof: By Lemma 4.18 we know that in this case $\mathcal{F}$ perfectly implements the constraint $x_1 \oplus \cdots \oplus x_p = b$ for some $p \geq 3$ and some $b \in \{0,1\}$. Thus the family $\mathcal{F} \cup \{T, F\}$ implements the functions $x \oplus y \oplus z = 0, x \oplus y \oplus z = 1$. Thus Nearest Codeword =Min CSP($\{x \oplus y \oplus z = 0, x \oplus y \oplus z = 1\}$ is A-reducible to Min CSP($\mathcal{F} \cup \{F, T\}$). Since $\mathcal{F}$ is neither 0-valid nor 1-valid, we can use Lemma 5.7 to conclude that Min CSP($\mathcal{F}$) is Nearest Codeword-hard. □

The next lemma describes the best known hardness of approximation for the Nearest Codeword problem. The result relies on an assumption stronger than NP $\neq$ P.

Lemma 7.16 ([2]) *For every $\epsilon > 0$,* Nearest Codeword *is hard to approximate to within a factor of $\Omega(2^{\log^{1-\epsilon} n})$, unless* NP *has deterministic algorithms running in time $n^{\log^{O(1)} n}$.*

Proof: The required hardness of the nearest codeword problem is shown by Arora et al. [2]. The nearest codeword problem, as defined in Arora et al., works with the following problem: Given a $m \times n$ matrix A and a m-dimensional vector b, find an n-dimensional vector x which minimizes the Hamming distance between Ax and b. Thus this problem can be expressed as a Min CSP problem with m affine constraints over n-variables. The only technical point to be noted is that these constraints have unbounded arity. In order to get rid of such long constraints, we replace a constraint of the form $x_1 \oplus \cdots \oplus x_l = 0$ into $l - 2$ constraints $x_1 \oplus x_2 \oplus z_1 = 0$, $z_1 \oplus x_3 \oplus z_2 = 0$, etc. on auxiliary variables $z_1, \ldots, z_{l-3}$. (The same implementation was used in Lemma 7.6.) This increases the number of constraints by a factor of at most n, but does not change the objective function. Thus if M represents the number of constraints in the new instance of the problem, then the approximation hardness which is $2^{\log^{1-\epsilon} m}$ can be expressed as $2^{\frac{1}{2}\log^{1-\epsilon} M}$ which is still growing faster than, say, $2^{\log^{1-2\epsilon} M}$. Since the result of [2] holds for every positive ϵ, we still get the desired result claimed above. □

It remains to see the Min Horn Deletion-hard case. We will have to draw some non-trivial consequences from the fact that a family is not IHS-B.

Lemma 7.17 *Assume $\mathcal{F} \not\subseteq \mathcal{F}_{\rm IHS}$ and either $\mathcal{F} \subseteq \mathcal{F}_{\rm WP}$ or $\mathcal{F} \subseteq \mathcal{F}_{\rm WN}$. Then $\mathcal{F}$ contains a function that is not C-closed.*

Proof: Let f be a C-closed function in $\mathcal{F}_{\rm WP}$ ($\mathcal{F}_{\rm WN}$). We claim that all of f's maxterms must be of the form $T(x_i)$, $F(x_i)$ or ${\rm OR}_{2,1}(x_i, x_j)$. If not, then since f is C-closed, the maxterm involving the complementary literals is also a maxterm of f, but the complementary maxterm is not weakly positive (and by Lemma 4.20 every maxterm of f must be weakly positive). But if all of f's maxterms are of the form $T(x_i)$, $F(x_i)$ or ${\rm OR}_{2,1}(x_i, x_j)$, then f is in IHS-B. The lemma follows from the fact that $\mathcal{F} \not\subseteq \mathcal{F}_{\rm IHS}$. □

Lemma 7.18 *If f is a weakly positive function not expressible as IHS-B+, then $\{f, T, F\} \stackrel{p}{\Longrightarrow}$ ${\rm OR}_{3,1}$. If f is a weakly negative function not expressible as IHS-B-, then $\{f, T, F\} \stackrel{p}{\Longrightarrow} {\rm OR}_{3,2}$.*

Proof: Let f be a weakly positive function. By Lemma 4.20 all maxterms of f are weakly positive. Since f is not IHS-$B+$, f must have a maxterm of the form $(\neg x_1 \bigvee x_2 \bigvee \cdots \bigvee x_p)$, for some $p \geq 3$. We first show that $\{f, F\}$ can perfectly implement the function XNOR. To get the former, consider the function

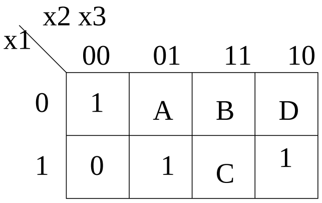
$$f_1(x_1, x_2) \stackrel{\rm def}{=} \exists x_{p+1}, \ldots, x_k \text{ s.t. } f(x_1, x_2, 0^{p-2}, x_{p+1}, \ldots, x_k).$$

The function f_1 satisfies the properties $f_1(10) = 0$, $f_1(00) = f_1(11) = 1$. Thus f_1 is either the function XNOR or $\mathrm{OR}_{2,1}$. Notice that the constraints $f(x_1, \ldots, x_k)$ and $F(x_i)$, $i \in \{3, \ldots, p\}$ perfectly implement f_1. Thus $\{f, F\}$ perfectly implement either the function XNOR or $\mathrm{OR}_{2,1}$. In the former case, we have the claim and in the latter case we use the fact that the constraints $\mathrm{OR}_{2,1}(x, y)$ and $\mathrm{OR}_{2,1}(y, x)$ perfectly implement $\mathrm{XNOR}(x, y)$.

Next, we show how the family $\{f, T, F, \mathrm{XNOR}\}$ (and hence $\{f, T, F\}$) can perfectly implement $\mathrm{OR}_{2,1}$. To do so, we consider the function

$$f_2(x_1, x_2, x_3) \stackrel{\text{def}}{=} \exists x_{p+1}, \ldots, x_k \text{ s.t. } f(x_1, x_2, x_3, 0^{p-3}, x_{p+1}, \ldots, x_k).$$

Again $\{f, F\}$ implement f_2 perfectly. By the definition of a maxterm, we find that f_2 satisfies the following properties: $f_2(100) = 0$ and $f_2(000) = f_2(110) = f_2(101) = 1$. Figure 3 gives the truth table for f_2, where the unknown values are denoted by A, B, C and D. If $C = 0$ then restricting $x_1 = 1$ gives the constraint $\mathrm{XOR}(x_2, x_3)$. But notice that XOR is not a weakly positive function and by Lemma 4.19 every function obtained by setting some of the variables in a weakly positive function to constants and existentially quantifying over some other subset of variables is a weakly positive function. Thus $C = 1$. If $D = 1$, we implement the function $\mathrm{OR}_{2,1}(x_1, x_2)$ by the constraints $f_2(x_1, x_2, x_3)$ and $F(x_3)$. Else we have $D = 0$, and the constraints $f_2(x_1, x_2, x_3)$ and $\mathrm{XNOR}(x_1, x_3)$ implement the constraint $\mathrm{OR}_{2,1}(x_2, x_1)$.



x1 \ x2 x3	00	01	11	10
0	1	A	B	D
1	0	1	C	1

Figure 3: Truth-table of the constraint f_2

Finally we conclude by observing that the constraints $f_2(x, z^1, z^2)$, $\mathrm{OR}_{2,1}(z^1, y)$ and $\mathrm{OR}_{2,1}(z^2, z)$, perfectly implement the constraint $\mathrm{OR}_{3,1}(x, y, z)$.

This completes the proof for the first part. The proof if f is weakly negative is similar. □

Lemma 7.19 (The Min Horn Deletion**-hard Case)** *If $\mathcal{F} \not\subseteq \mathcal{F}'$, for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}, \mathcal{F}_{\mathrm{IHS}}, \mathcal{F}_{2\mathrm{A}}, \mathcal{F}_{2\mathrm{CNF}}\}$, and either $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WP}}$ or $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WN}}$, then* Weighted Min CSP$(\mathcal{F})$ *is* Min Horn Deletion*-hard.*

Proof: From Lemma 7.18 we have that either Min CSP$(\{\mathrm{OR}_{3,1}, T, F\}$ or Min CSP$(\{\mathrm{OR}_{3,2}, T, F\}$ is A-reducible to Min CSP$(\mathcal{F})$. Furthermore, since $\mathcal{F}$ is not 0-valid or 1-valid we have that Min CSP$(\mathcal{F} \cup \{T, F\})$ is A-reducible to Min CSP$(\mathcal{F})$. The lemma follows by an application of Proposition 7.1 which shows that the problems Min CSP$(\{\mathrm{OR}_{3,1}, T, F\})$ A-reduces to Min CSP$(\{\mathrm{OR}_{3,2}, T, F\})$. □

To show the hardness of Min Horn Deletion we define a variant of the "label cover" problem. The original definition from [2] used a different objective function. Our variant is similar to one used by Amaldi and Kann [1] under the name Total Label Cover.

Definition 7.20 (Total Label Cover$_p$)
Instance: *An instance is described by sets $\mathcal{R}$, $\mathcal{Q}$ and $\mathcal{A}$ and by p functions (given by their tables) $Q_1, \ldots, Q_p : \mathcal{R} \to \mathcal{Q}$ and a function* Acc $: \mathcal{R} \times (\mathcal{A})^p \to \{0, 1\}$.

Feasible solutions: *A solution is a collection of p functions $A_1, \ldots, A_p : \mathcal{Q} \rightarrow 2^{\mathcal{A}}$. The solution is feasible if for every $R \in \mathcal{R}$, there exists $a_1 \in A_1(Q_1(R)), \ldots, a_p \in A_p(Q_p(R))$ such that* $\textsc{Acc}(R, a_1, \ldots, a_p) = 1$.

Objective: *The objective is to minimize $\sum_{i=1}^{p} \sum_{q \in Q} |A_i(q)|$.*

In the appendix, we show how results from interactive proofs imply the hardness of approximating Min Label-Cover to within a factor of $2^{\log^{1-\epsilon} n}$. We now use this result to show that hardness of Min Horn Deletion.

Lemma 7.21 *For every $\epsilon > 0$,* Min Horn Deletion *is* NP*-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$.*

Proof: Let p be such that Min Label-Cover$_p$ is NP-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$. (By Lemma A.3 such a p exists.) We now reduce Min Label-Cover$_p$ to Min Horn Deletion.

Let $(Q_1, \ldots, Q_p, \textsc{Acc})$ be an instance of Min Label-Cover$_p$, where $Q_i : \mathcal{R} \rightarrow \mathcal{Q}$ and $\textsc{Acc} : \mathcal{R} \times (\mathcal{A})^p \rightarrow \{0,1\}$. For any $R \in \mathcal{R}$, we define $Acc(R) = \{(a_1, \ldots, a_p) : V(R, a_1, \ldots, a_p) = 1\}$.

We now describe the reduction. For any $R \in \mathcal{R}$, $a_1, \ldots, a_p \in \mathcal{A}$, we have a variable $v_{R,a_1,\ldots,a_p}$ whose intended meaning is the value of $\textsc{Acc}(R, a_1, \ldots, a_p)$. Moreover, for every $i \in [p]$, $Q \in \mathcal{Q}$, and $a \in A_i$ we have a variable $x_{i,Q,a}$, with the intended meaning being that its value is 1 if and only if $a \in A_i(Q)$. For any $x_{i,Q,a}$ we have the weight-one constraint $\neg x_{i,q,a}$. The following constraints (each with weight $(p \times |\mathcal{Q}| \times |\mathcal{A}|)$) enforce the variables to have their intended meaning. Due to their weight, it is never convenient to contradict them.

$$\begin{array}{ll} \forall R \in \mathcal{R} : & \bigvee_{(a_1,\ldots,a_p) \in Acc(R)} v_{R,a_1,\ldots,a_p} \\ \forall R \in \mathcal{R}, a_1, \ldots, a_p \in \mathcal{A}, i \in [p] : & v_{R,a_1,\ldots,a_p} \Rightarrow x_{i,Q_i(R),a_i} \end{array}$$

The constraints of the first kind can be perfectly implemented with OR_3 and $\mathrm{OR}_{3,1}$ (see Lemma 7.7). It can be checked that this is an AP-reduction from Min Label-Cover$_p$ to Min Horn Deletion and thus the lemma follows. □

8 Min Ones Classification

8.1 Preliminaries: Min Ones vs. Min CSP

We start with the following easy relation between Min CSP and Min Ones problems. Recall that a family $\mathcal{F}$ is decidable if membership in $\textsc{Sat}(\mathcal{F})$ is decidable in polynomial time.

Proposition 8.1 *For any decidable constraint family $\mathcal{F}$,* Weighted Min Ones$(\mathcal{F})$ *AP-reduces to* Weighted Min CSP$(\mathcal{F} \cup \{F\})$.

Proof: Let $\mathcal{I}$ be an instance of Weighted Min Ones$(\mathcal{F})$ over variables $x_1, \ldots, x_n$ with weights $w_1, \ldots, w_n$. Let $w_{\max}$ be the largest weight. We construct an instance $\mathcal{I}'$ of Weighted Min CSP$(\mathcal{F} \cup \{F\})$ by leaving the constraints of $\mathcal{I}$ (each with weight $nw_{\max}$), and adding a constraint $F(x_i)$ of weight w_i for any $i = 1, \ldots, n$. Notice that whenever $\mathcal{I}$ is feasible, the optimum value for $\mathcal{I}$ equals the optimum value for $\mathcal{I}'$. Given a r-approximate solution to $\mathbf{x}$ to $\mathcal{I}'$, we check to see if $\mathcal{I}$

is feasible and if so find any feasible solution $\mathbf{x}'$ and output solution (from among $\mathbf{x}$ and $\mathbf{x}'$) that achieves a lower objective. It is clear that the solution is at least an r-approximate solution if $\mathcal{I}$ is feasible. □

Reducing a Min CSP problem to a Min Ones problem is slightly less general.

Proposition 8.2 *For any function f, let f' and f'' denote the functions $f'(\mathbf{x}, y) = \mathrm{OR}(f(\mathbf{x}), y)$ and $f''(\mathbf{x}, y) = \mathrm{XOR}(f(\mathbf{x}), y)$ respectively. If constraint families $\mathcal{F}$ and $\mathcal{F}'$ are such that for every $f \in \mathcal{F}$, f' or f'' is in $\mathcal{F}'$, then* Weighted Min CSP$(\mathcal{F})$ *AP-reduces to* Weighted Min Ones$(\mathcal{F}')$.

Proof: Given an instance $\mathcal{I}$ of Weighted Min CSP$(\mathcal{F})$ we create an instance $\mathcal{I}'$ of Weighted Min Ones$(\mathcal{F}')$ as follows: For every constraint C_j we introduce an auxiliary variable y_j. The variable takes the same weight as the constraint C_j in $\mathcal{I}$. The original variables are retained with weight zero. If the constraint $C_j(\mathbf{x}) \bigvee y_j$ is a constraint of $\mathcal{F}'$ we apply that constraint, else we apply the constraint $C_j(\mathbf{x}) \oplus y = 1$. Given an assignment to the variables of $\mathcal{I}$, notice that by setting $y_j = \neg C_j$, we get a feasible solution to $\mathcal{I}'$ with the same objective value; conversely, a feasible solution to $\mathcal{I}'$ when projected onto the variables $\mathbf{x}$ gives a solution with the same value to the objective function of $\mathcal{I}$. This shows that the optimum value to $\mathcal{I}'$ equals that of $\mathcal{I}$ and that an r-approximate solution to $\mathcal{I}'$ projects to give an r-approximate solution to $\mathcal{I}$. □

Finally the following easy proposition is invoked at a few places.

Proposition 8.3 *If $\mathcal{F} \Longrightarrow f$, then $\mathcal{F}^- \Longrightarrow f^-$.*

8.2 Containment Results for Min Ones

Lemma 8.4 (PO **ccontainment)** *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{\mathrm{WN}}, \mathcal{F}_{\mathrm{2A}}\}$, then* Weighted Min Ones$(\mathcal{F})$ *is solvable exactly in polynomial time.*

Proof: Follows from Lemma 6.5 and from the observation that for any family $\mathcal{F}$, solving Weighted Min Ones$(\mathcal{F})$ to optimality reduces to solving Weighted Max Ones$(\mathcal{F}^-)$ to optimality. □

Lemma 8.5 *If $\mathcal{F} \subseteq \mathcal{F}'$ for $\mathcal{F}' \in \{\mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{IHS}}\}$, then* Weighted Min Ones$(\mathcal{F})$ *is in* APX.

Proof: For the case $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{2CNF}}$, a 2-approximate algorithm is given by Hochbaum et al. [25].

Consider now the case $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{IHS}}$. From Proposition 3.4 it is sufficient to consider only basic IHS-B constraints. Since IHS-$B-$ constraints are weakly negative, we will restrict to basic IHS-$B+$ constraints. We use linear-programming relaxations and deterministic rounding. Let k be the maximum arity of a function in $\mathcal{F}$, we will give a k-approximate algorithm. Let $\phi = \{C_1, \ldots, C_m\}$ be an instance of Weighted Min Ones$(\mathcal{F})$ over variable set $X = \{x_1, \ldots, x_n\}$ with weights $w_1, \ldots, w_n$. The following is an integer linear programming formulation of finding the minimum weight satisfying assignment for ϕ.

$$\begin{array}{lll}
\textsf{Minimize} & \sum_i w_i y_i & \\
\textsf{Subject to} & & \\
& y_{i_1} + \ldots + y_{i_h} \geq 1 & \forall (x_{i_1} \bigvee \ldots \bigvee x_{i_h}) \in \phi \\
& y_{i_1} - y_{i_2} \geq 0 & \forall (x_{i_1} \bigvee \neg x_{i_2}) \in \phi \qquad\qquad \text{(SCB)} \\
& y_i = 0 & \forall \neg x_i \in \phi \\
& y_i = 1 & \forall x_i \in \phi \\
& y_i \in \{0, 1\} & \forall i \in \{1, \ldots, n\}
\end{array}$$

Consider now the linear programming relaxation obtained by relaxing the $y_i \in \{0,1\}$ constrains into $0 \leq y_i \leq 1$. We first find an optimum solution $\mathbf{y}^*$ for the relaxation, and then we define a 0/1 solution by setting $y_i = 0$ if $y_i^* < 1/k$, and $y_i = 1$ if $y_i^* \geq 1/k$. It is easy to see that this rounding increases the cost of the solution at most k times and that the obtained solution is feasible for (SCB). □

Lemma 8.6 *For any $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{A}}$,* Weighted Min Ones($\mathcal{F}$) *is A-reducible to* Nearest Codeword.

Proof: From Lemmas 7.6 and 3.9 we have that Weighted Min Ones($\mathcal{F}$) is A-reducible to Weighted Min Ones($\{\mathrm{XNOR}_3, \mathrm{XOR}_3\}$). From Proposition 8.1, we have that Weighted Min Ones($\mathcal{F}$) A-reduces to Weighted Min CSP($\{\mathrm{XOR}_3, \mathrm{XNOR}_3, F\}$). Notice further that the family $\{\mathrm{XNOR}_3, \mathrm{XOR}_3\}$ can implement F (by Lemma 4.6). Thus we have that we have that Weighted Min Ones($\mathcal{F}$) A-reduces to Weighted Min CSP($\{\mathrm{XOR}_3, \mathrm{XNOR}_3, \}$) = Nearest Codeword. □

Lemma 8.7 *For any $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WP}}$,* Weighted Min Ones($\mathcal{F}$) *AP-reduces to* Min Horn Deletion*.*

Proof: Follows from the following sequence of assertions:

(1) $\{\mathrm{OR}_{3,1}, T, F\}$ perfectly implements $\mathcal{F}$ (Lemma 7.7).

(2) Weighted Min Ones($\mathcal{F}$) AP-reduces to Weighted Min Ones($\{\mathrm{OR}_{3,1}, T, F\}$) (Lemma 3.9).

(3) Weighted Min Ones($\{\mathrm{OR}_{3,1}, T, F\}$) AP-reduces to Weighted Min CSP($\{\mathrm{OR}_{3,1}, T, F\}$) = Min Horn Deletion (Proposition 8.1).

□

Proposition 8.8 *If $\mathcal{F}$ is decidable then* Min Ones($\mathcal{F}$) *is in* poly-APX*.*

Proof: The proposition follows immediately from the fact that in this case it is easy to determine if the input instance is feasible and if so, if the optimum value is zero. If so we output the $\mathbf{0}$ as the solution, else we output any feasible solution. Since the objective is at least 1 and the solution has value at most n, this is an n-approximate solution. □

8.3 Hardness Results for Min Ones

We start by considering the hardest problems first. The case when $\mathcal{F}$ is not decidable is immediate. We move to the case where $\mathcal{F}$ may be 1-valid, but not in any other of Schaefer's easy classes.

Lemma 8.9 *If $\mathcal{F} \not\subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{2\mathrm{CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then* Weighted Min Ones($\mathcal{F}$) *is hard to approximate to within any factor, and* Min Ones($\mathcal{F}$) *is* poly-APX*-hard.*

Proof: We first show how to handle the weighted case. The hardness for the unweighted case will follow easily. Consider a function $f \in \mathcal{F}$ which is not weakly positive. For such an f, there exists assignments $\mathbf{a}$ and $\mathbf{b}$ such that $f(\mathbf{a}) = 1$ and $f(\mathbf{b}) = 0$ and $\mathbf{a}$ is zero in every coordinate where $\mathbf{b}$ is zero. (Such a input pair exists for every non-monotone function f and every monotone function is also weakly positive.) Now let f' be the constraint obtained from f by restricting it to inputs where $\mathbf{b}$ is one, and setting all other inputs to zero. Then f' is a satisfiable function which is not 1-valid. We can now apply Schaefer's theorem [42] to conclude that Sat($\mathcal{F} \cup \{f'\}$) is hard to decide. We now reduce an instance of deciding Sat($\mathcal{F} \cup \{f'\}$) to approximating Weighted Min CSP($\mathcal{F}$). Given an instance $\mathcal{I}$ of Sat($\mathcal{F} \cup \{f'\}$) we create an instance which has some auxiliary variables $W_1, \ldots, W_k$ which are all supposed to be zero. This in enforced by giving them very large weights. We now replace every occurrence of the constraint f' in $\mathcal{I}$ by the constraint f on the corresponding variables with the W_i's in place which were set to zero in f to obtain f'. It is clear that if a "small" weight solution exists to the resulting Weighted Min CSP problem, then $\mathcal{I}$ is satisfiable, else it is not. Thus we conclude it is NP-hard to approximate Weighted Min CSP to within any bounded factors.

For the unweighted case, it suffices to observe that by using polynomially bounded weights above, we get a poly-APX hardness. Further one can get rid of weights entirely by replicating variables. □

We may now restrict our attention to function families $\mathcal{F}$ that are 2CNF or affine or weakly positive or weakly negative or 0-valid. In particular, by the containment results shown in the previous section, in all such cases the problem Weighted Min Ones($\mathcal{F}$) is in poly-APX. We now give a weight-removing lemma which allow us to focus on showing the hardness of the weighted problems.

Lemma 8.10 *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_{\text{2CNF}}, \mathcal{F}_{\text{A}}, \mathcal{F}_{\text{WP}}, \mathcal{F}_{\text{WN}}, \mathcal{F}_0\}$, then* Weighted Min Ones($\mathcal{F}$) *AP-reduces to* Min Ones($\mathcal{F}$).

Proof: By Lemma 3.11 it suffices to verify that Weighted Min Ones($\mathcal{F}$) is in poly-APX in all cases. If $\mathcal{F}$ is weakly negative or 0-valid, then this follows from Lemma 8.4. If $\mathcal{F}$ is 2CNF then this follows from Lemma 8.5. If $\mathcal{F}$ is affine or weakly positive, then it A-reduces to Nearest Codeword or MinHornDeletion respectively which are in poly-APX by Corollary 7.8. □

Before dealing with the remaining cases, we prove one more lemma that is useful in dealing with Min Ones problems.

Lemma 8.11 *For every constraint family $\mathcal{F}$ such that $\mathcal{F} \cup \{F\}$ is decidable,* Weighted Min Ones($\mathcal{F} \cup \{F\}$) *AP-reduces to* Weighted Min Ones($\mathcal{F}$).

Proof: Given an instance $\mathcal{I}$ of Weighted Min Ones($\mathcal{F} \cup \{F\}$) on n variables $x_1, \ldots, x_n$ with weights $w_1, \ldots, w_n$ we create an instance $\mathcal{I}'$ of Weighted Min Ones($\mathcal{F}$), on the variables $x_1, \ldots, x_n$ using all the constraints of $\mathcal{I}$ that are from $\mathcal{F}$; and for every variable variable x_i such that $F(x_i)$ is a constraint of $\mathcal{I}$, we increase the weight of the variable x_i to $nw_{\max}$ where $w_{\max}$ is the maximum of the weights $w_1, \ldots, w_n$. As in Lemma 8.1 we observe that if $\mathcal{I}$ is feasible, then the optima for $\mathcal{I}$ and $\mathcal{I}'$ are equal and given an r-approximate solution to $\mathcal{I}'$ we can find an r-approximate solution to $\mathcal{I}$. Furthermore, since $\mathcal{F} \cup \{F\}$ is decidable, we can decide whether or not $\mathcal{I}$ is feasible. □

We now deal with the affine problems.

Lemma 8.12 *If $\mathcal{F}$ is affine but not width-2 affine or 0-valid then* Min Ones(XOR_3) *is AP-reducible to* Weighted Min Ones($\mathcal{F}$).

Proof: Notice that since $\mathcal{F}$ is affine, so is $\mathcal{F}^-$. Furthermore, $\mathcal{F}^-$ is neither width-2 affine nor 1-valid. Thus by Lemma 6.10 $\mathcal{F}^-$ perfectly implements either the family $\{\text{XNOR}_3\}$ or the family $\{\text{XOR}, \text{XNOR}_4\}$. Thus, by applying Proposition 8.3, we get that $\mathcal{F}$ implements either XOR_3 or the family $\{\text{XOR}, \text{XNOR}_4\}$. In the former case, we are done (by Lemma 3.9). In the latter case, notice that the constraints $\text{XNOR}_4(x_1, x_2, x_3, x_5)$ and $\text{XOR}(x_4, x_5)$ perfectly implement the constraint $\text{XOR}_4(x_1, x_2, x_3, x_5)$. Thus we conclude that Weighted Min Ones(XOR_4) is AP-reducible to Weighted Min Ones($\mathcal{F}$). Finally we use Lemma 8.11 to conclude that the family Weighted Min Ones($\mathcal{F}$)($\{\text{XOR}\}|_0$) is AP-reducible to Weighted Min Ones($\mathcal{F}$). The lemma follows from the fact that $\text{XOR}_3 \in \{\text{XOR}_4\}|_0$. □

Lemma 8.13 *If $\mathcal{F}$ is affine but not width-2 affine or 0-valid then, for every $\epsilon > 0$,* MIN ONES($\mathcal{F}$) *is* NEAREST CODEWORD*-hard and hard to approximate to within a factor of $\Omega(2^{\log^{\epsilon} n})$.*

Proof: Follows from the following sequence of reductions:

$$\begin{array}{ll} \multicolumn{2}{l}{\text{NEAREST CODEWORD}} \\ = & \text{WEIGHTED MIN CSP}(\{\text{XOR}_3, \text{XNOR}_3\}) \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\{\text{XOR}_4, \text{XNOR}_4\}) \text{ (using Proposition 8.2)} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\{\text{XOR}_3, \text{XOR}\}) \text{ (see below)} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\text{XOR}_3) \text{ (using Lemma 8.11)} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\mathcal{F}) \text{ (using Lemmas 8.12 and 3.9)} \\ \leq_{\text{AP}} & \text{MIN ONES}(\mathcal{F}) \text{ (using Lemma 8.10.)} \end{array}$$

The second reduction above follows by combining Lemma 3.9 with the observation that the family $\{\text{XOR}_3, \text{XOR}\}$ perfectly implement the functions XOR_4 and XNOR_4 as shown next. The constraints $\text{XOR}_3(u, v, w)$ and $\text{XOR}_3(w, x, y)$ perfectly implement the constraint $\text{XNOR}_4(u, v, x, y)$; the constraints $\text{XOR}_4(u, v, w, x)$ and $\text{XOR}(w, y)$ perfectly implement $\text{XOR}_4(u, v, x, y)$. The hardness of approximation of NEAREST CODEWORDLemma 7.16. □

Lemma 8.14 *If $\mathcal{F}$ is weakly positive and not IHS-B (nor 0-valid) then* MIN ONES($\mathcal{F}$) *is* MIN HORN DELETION*-hard, and hence hard to approximate within $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$.*

Proof: Follows from the following sequence of reductions:

$$\begin{array}{ll} \multicolumn{2}{l}{\text{MIN HORN DELETION}} \\ = & \text{WEIGHTED MIN CSP}(\{\text{OR}_{3,1}, T, F\} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\{\text{OR}_{4,1}, \text{OR}_2, \text{OR}_{2,1}\}) \text{ (Using Proposition 8.2.)} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\{\text{OR}_{3,1}, T, F\}) \text{ (Using Lemmas 7.7 and 3.9.)} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\mathcal{F} \cup \{T, F\}) \text{ (Using Lemmas 7.18 and 3.9.)} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\mathcal{F} \cup \{F\}) \text{ (Using Lemma 4.6 to perfectly implement } T\text{.)} \\ \leq_{\text{AP}} & \text{WEIGHTED MIN ONES}(\mathcal{F}) \text{ (Using Lemma 8.11.)} \\ \leq_{\text{AP}} & \text{MIN ONES}(\mathcal{F}) \text{ (Using Lemma 8.10.)} \end{array}$$

The hardness of approximation follows from Lemma 7.21. □

Lemma 8.15 MIN ONES(OR) *is* APX*-hard.*

Proof: We reduce VERTEX COVER to MIN ONES(OR). Given a graph G on n vertices, we construct an instance of MIN ONES(OR) on n variables $x_1, \ldots, x_n$. For every edge between vertex i and j of G, we create a constraint $\text{OR}(x_i, x_j)$. We notice that there is a one-to-one correspondence between an assignment to the variables and vertex covers in G (with variables assigned 1 corresponding to vertices in the cover) and the minimum vertex cover minimizes the sum of the variables. The lemma follows from the fact that VERTEX COVER is APX-hard [39, 3]. □

Lemma 8.16 (APX-hardness) *If $\mathcal{F} \not\subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{\text{WN}}, \mathcal{F}_{2\text{A}}\}$, then* MIN ONES($\mathcal{F}$) *is* APX*-hard.*

Proof: We mimic the proof of Lemma 6.14. We assume that $\mathcal{F}$ is not affine – the case where $\mathcal{F}$ is affine is shown to be NEAREST CODEWORD-hard in Lemma 8.13. By Lemma 8.10 it suffices to show that WEIGHTED MIN ONES($\mathcal{F}$) is APX-hard; and by Lemma 8.11 it suffices to show that WEIGHTED MIN ONES($\mathcal{F} \cup \{F\}$) is APX-hard. Since $\mathcal{F} \cup \{F\}$ is not 0-valid or 1-valid or C-closed it implements every function in $\mathcal{F} \cup \{T, F\}$ and thus every function in $\mathcal{F}|_{0,1}$. We now shift focus on to the family $(\mathcal{F}|_{0,1})^-$. Furthermore $(\mathcal{F}|_{0,1})^-$ is neither weakly positive nor affine and thus by Lemmas 6.20 and 6.21 it implements NAND. Using Proposition 8.3 we get that $\mathcal{F}_{0,1}$ implements OR. Using Lemma 8.15 we get that WEIGHTED MIN ONES(OR) is APX-hard. Thus we conclude that WEIGHTED MIN ONES($\mathcal{F}$) is APX-hard. □

Acknowledgments

We thank Mihir Bellare, Nadia Creignou, Oded Goldreich, and Jean-Pierre Seifert for useful discussions. We thank an anonymous referee for pointing out numerous errors in a previous version of this paper.

References

[1] E. AMALDI AND V. KANN. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1-2):181–210, 1995.

[2] S. ARORA, L. BABAI, J. STERN AND Z. SWEEDYK. The hardness of approximate optima in lattices, codes, and systems of linear equations. *Journal of Computer and System Sciences*, 54(2):317–331, 1997.

[3] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY. Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[4] S. ARORA AND S. SAFRA. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[5] S. ARORA AND M. SUDAN. Improved low degree testing and its applications. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, 4-6 May 1997.

[6] T. ASANO, T. ONO, AND T. HIRATA. Approximation algorithms for the maximum satisfiability problem. In Rolf G. Karlsson and Andrzej Lingas, editors, *SWAT '96, 5th Scandinavian Workshop on Algorithm Theory*, volume 1097 of Lecture Notes in Computer Science, pages 100–111, Reykjavik, Iceland, 3-5 July 1996. Springer.

[7] M. BELLARE, S. GOLDWASSER, C. LUND AND A. RUSSELL. Efficient probabilistically checkable proofs and applications to approximation. *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 294–304, San Diego, California, 16-18 May 1993.

[8] M. BELLARE, O. GOLDREICH, AND M. SUDAN. Free bits, PCPs, and nonapproximability – towards tight results. *SIAM Journal on Computing*, 27(3):804-915, June 1998.

[9] R. Boppana and M. Haldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32(2), 180–196, 1992.

[10] D.P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, New York, 1994.

[11] N. Creignou. A Dichotomy Theorem for Maximum Generalized Satisfiability Problems. *Journal of Computer and System Sciences*, 51(3): 511–522, 1995.

[12] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1): 1–12, 25 February 1996.

[13] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. *SIAM Journal on Computing*, 28(5):1759-1782, 1999.

[14] P. Crescenzi and A. Panconesi. Completeness in approximation classes. *Information and Computation*, 93(2):241-262, August 1991.

[15] P. Crescenzi, R. Silvestri, and L. Trevisan. To weight or not to weight: Where is the question? In *Proceedings of the 4th IEEE Israel Symposium on Theory of Computing and Systems*, pages 68–77, 1996. Full version to appear in *Information and Computation.*

[16] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2): 151–174, February 1998.

[17] T. Feder and M. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1): 57–104, February 1999.

[18] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2): 268–292, 1996.

[19] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[20] N. Garg, V.V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

[21] M. Goemans and D. Williamson. New 3/4-approximation algorithms for MAX SAT. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

[22] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[23] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *37th Annual Symposium on Foundations of Computer Science*, pages 627–636, Burlington, Vermont, 14-16 October 1996. IEEE.

[24] J. Håstad. Some optimal inapproximability results. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 1–10, El Paso, Texas, 4-6 May 1997.

[25] D.S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62:69–83, 1993.

[26] H. B. Hunt III, M. V. Marathe, and R. E. Stearns. Generalized CNF satisfiability problems and non-efficient approximability (preliminary version). *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, pages 356–366, Amsterdam, The Netherlands, 28 June-1 July 1994. IEEE Computer Society Press.

[27] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? *38th Annual Symposium on Foundations of Computer Science*, pages 406–415, Miami Beach, Florida, 20-22 October 1997. IEEE.

[28] S. Khanna and R. Motwani. Towards a Syntactic Characterization of PTAS. *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 329–337, Philadelphia, Pennsylvania, 22-24 May 1996.

[29] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1): 164–191, February 1999.

[30] S. Khanna, M. Sudan and L. Trevisan. Constraint satisfaction: The approximability of minimization problems. *Proceedings, Twelfth Annual IEEE Conference on Computational Complexity*, pages 282–296, Ulm, Germany, 24-27 June 1997. IEEE Computer Society Press.

[31] S. Khanna, M. Sudan and D. P. Williamson. A complete classification of the approximability of maximization problems derived from Boolean constraint satisfaction. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 11–20, El Paso, Texas, 4-6 May 1997.

[32] P. Klein, A. Agarwal, R. Ravi and S. Rao. Approximation through multicommodity flow. *31st Annual Symposium on Foundations of Computer Science*, volume II, pages 726–737, St. Louis, Missouri, 22-24 October 1990. IEEE.

[33] P.N. Klein, S.A. Plotkin, S. Rao, and É. Tardos. Approximation algorithms for Steiner and directed multicuts. *Journal of Algorithms*, 22(2):241–269, February 1997.

[34] P. G. Kolaitis and M. N. Thakur. Approximation properties of NP minimization classes. *Journal of Computer and System Sciences*, 50(3):391-411, June 1995.

[35] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.

[36] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, September 1994.

[37] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In Svante Carlsson Andrzej Lingas, Rolf G. Karlsson, editor, *Automata, Languages and Programming, 20th International Colloquium*, volume 700 of Lecture Notes in Computer Science, pages 40–51, Lund, Sweden, 5-9 July 1993. Springer-Verlag.

[38] R. Panigrahy and S. Vishwanathan. An O(log* n) approximation algorithm for the asymmetric p-center problem. *Journal of Algorithms*, 27(2):259–268, May 1998.

[39] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, December 1991.

[40] E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4(2):133-157, 1994.

[41] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 475–484, El Paso, Texas, 4-6 May 1997.

[42] T. Schaefer. The complexity of satisfiability problems *Tenth Annual ACM Symposium on Theory of Computing*, pages 216–226, San Diego, California, 1-3 May 1978.

[43] L. Trevisan, G. Sorkin, M. Sudan and D. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6): 2074–2097, December 2000.

[44] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17(3):475-502, November 1994.

[45] D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25(6):1293–1304, December 1996.

A Hardness of Total Label Cover

Definition A.1 *$L \in \mathrm{MIP}_{c,s}[p,r,q,a]$ if there exists a polynomial time bounded probabilistic oracle machine V (verifier) such that on input $x \in \{0,1\}^n$, the verifier picks a random string $R \in \{0,1\}^{r(n)}$ and generates p queries $Q_1 = Q_1(x,R), \ldots, Q_p = Q_p(x,R) \in \{0,1\}^{q(n)}$ and sends query Q_i to prover Π_i and receives from prover Π_i an answer $A_i = A_i(Q_i) \in \{0,1\}^{a(n)}$ and then computes a verdict $\textsc{Acc}(x,R,A_1,\ldots,A_p) \in \{0,1\}$ with the following properties:*

Completeness: $x \in L \Rightarrow \exists A_1(\cdot),\ldots,A_p(\cdot)$ *such that* $\mathbf{E}_R[\textsc{Acc}(x,R,A_1,\ldots,A_p)] \geq c(n)$.

Soundness: $x \notin L \Rightarrow \forall A_1(\cdot),\ldots,A_p(\cdot), \quad \mathbf{E}_R[\textsc{Acc}(x,R,A_1,\ldots,A_p)] < s(n)$.

We say V is uniform if for every x and i, there exists $d_{x,i}$, s.t. for every query $Q_i \in \{0,1\}^{q(n)}$, $|\{R \in \{0,1\}^{r(n)} | Q_i(R) = Q_i\}| = d_{x,i}$. We say L is in $\textsc{uniform}$-$\mathrm{MIP}_{c,s}[p,r,q,a]$ *if there exists a uniform verifier V which places L in $\mathrm{MIP}_{c,s}[p,r,q,a]$.*

We use a recent result of Raz and Safra [41] (see also [5] for an alternate proof) which provides a strong $\textsc{uniform}$-MIP containment result for NP.

Lemma A.2 ([41, 5]) *For every $\epsilon > 0$, there exist constants p, c_1, c_2 and c_3 such that*

$$\mathrm{NP} \subseteq \textsc{uniform}\text{-}\mathrm{MIP}_{1,2^{-\log^{1-\epsilon} n}}[p, c_1 \log n, c_2 \log n, c_3 \log n].$$

Remark:

(1) The result shown by [41, 5] actually has smaller answer sizes, but this turns out to be irrelevant to our application below, so we don't mention their stronger result.

(2) The uniformity property is not mentioned explicitly in the above papers. However it can be verified from their proofs that this property does hold for the verifier constructed there.

The following reduction is essentially from [36, 7, 2].

Lemma A.3 *For every $\epsilon > 0$, there exists a $p = p_\epsilon$ such that Total Label Cover$_p$ is NP-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$.*

Proof: We use Lemma A.2. Let L be an NP-complete language and for $\epsilon > 0$, let p, c_1, c_2, c_3 be such that $L \in \text{UNIFORM-MIP}_{1,2^{-\log^{1-\epsilon/2} n}}[p, c_1 \log n, c_2 \log n, c_3 \log n]$ and let V be the verifier that shows this containment. Given an instance $x \in \{0,1\}^n$ of L, we create an instance of Total Label Cover$_p$ as follows: Set $Q_i(R)$ to be the query generated by V to prover Π_i on input x and random string R. For every $R, a_1, \ldots, a_p$ $\text{ACC}(R, a_1, \ldots, a_p)$ is 1 if V accepts the answers $a_1, \ldots, a_p$ on random string R.

Let $\mathcal{Q} = \{0,1\}^{c_2 \log n}$ denote the set of all possible queries and let $\mathcal{R}$ denote the space of all possible random strings (i.e., $\mathcal{R} = \{0,1\}^{c_1 \log n}$.) If $x \in L$, it is clear that there exists a feasible solution $A_1, \ldots, A_p$ such that for every query $q \in \mathcal{Q}$, and for every $i \in \{1, \ldots, p\}$, it is the case that $|A_i(q)| = 1$. Thus the value of the optimum solution is at most $p \cdot |\mathcal{Q}|$.

Now we claim for a given x, if the mapped instance of Total Label Cover has a solution of size $Kp|\mathcal{Q}|$ then there exist provers $\Pi_1, \ldots, \Pi_p$ such that V accepts with probability at least $K^{-1/p}/(p+1)^{p+1}$.

To see this let $\Pi_i(q)$ be a random element of $A_i(q)$. If $n_{i,q}$ denotes the cardinality of $A_i(q)$, then the probability that V accepts the provers response is given by

$$\frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}} \prod_i 1/n_{i,Q_i(R)}.$$

Define $\mathcal{R}_i$ to be $\{R \in \mathcal{R} | n_{i,Q_i(R)} \geq (p+1)K\}$. By Markov's inequality and the uniformity of the protocol $|\mathcal{R}_i|/|\mathcal{R}| \leq 1/(p+1)$.

Let $\mathcal{R}_0 = \mathcal{R} - \mathcal{R}_1 - \mathcal{R}_2 - \cdots - \mathcal{R}_p$. Then $|\mathcal{R}_0|/|\mathcal{R}| \geq 1/(p+1)$.

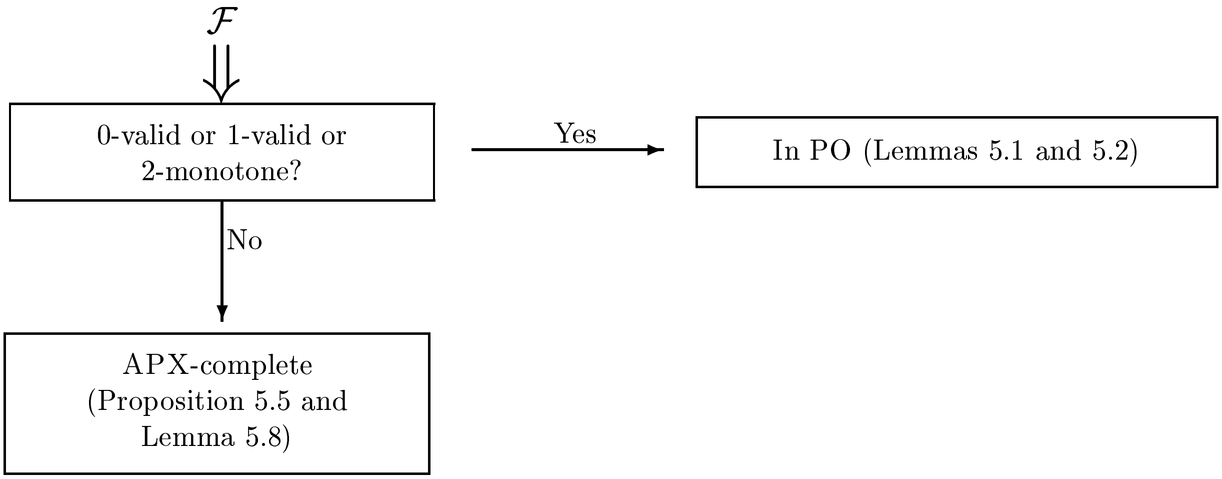
We go back to bounding the probability above:

$$\begin{aligned}
\frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}} \prod_i 1/n_{i,Q_i(R)} &\geq \frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}_0} \prod_i 1/n_{i,Q_i(R)} \\
&\geq \frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}_0} \prod_i 1/n_{i,Q_i(R)} \\
&\geq \frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}_0} (1/((p+1)K)^p) \\
&\geq K^{-1/p}/(p+1)^{p+1}.
\end{aligned}$$

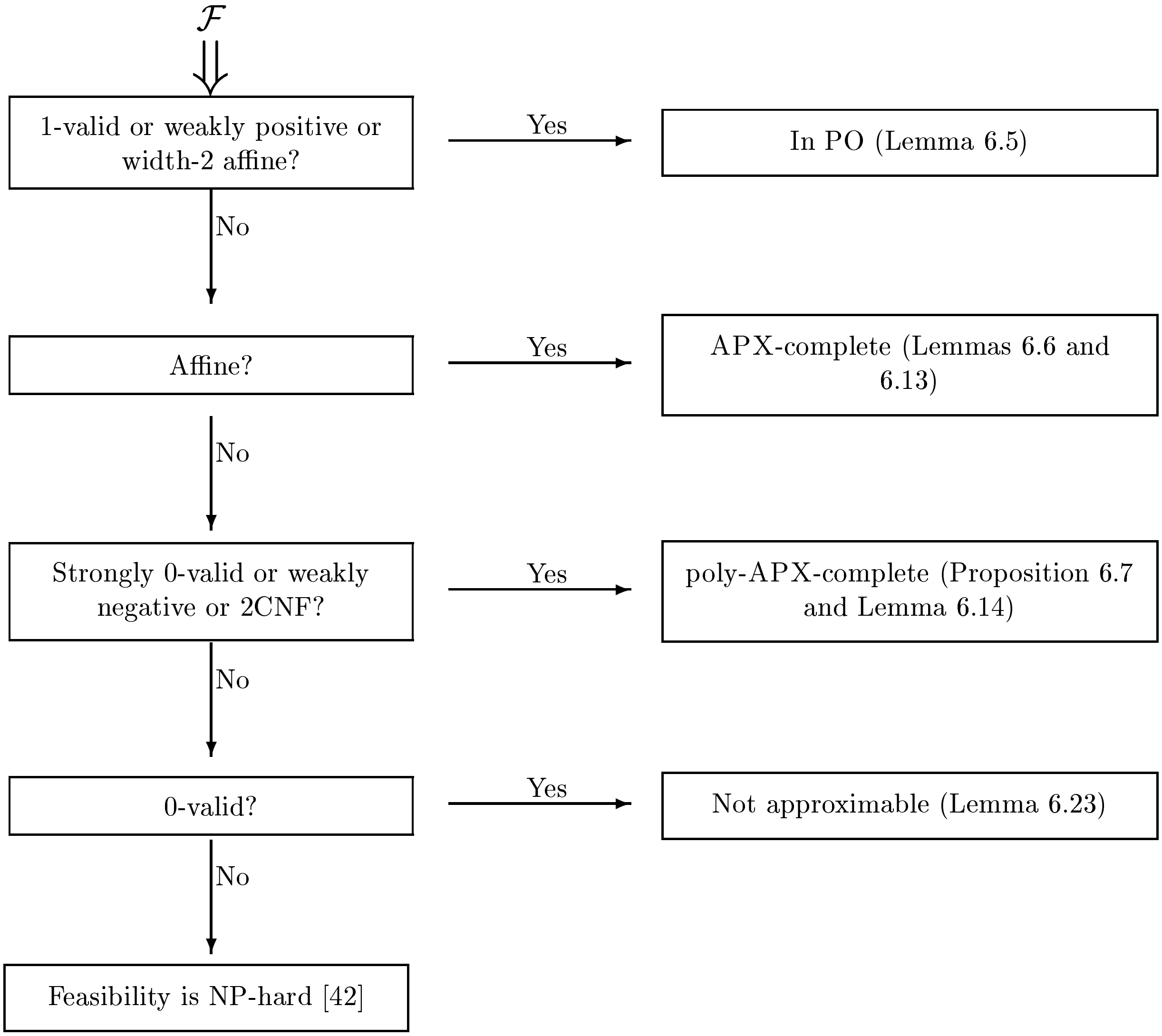
It follows that if $K = K(n)$ is less than $2^{\log^{1-\epsilon} n}$, then for sufficiently large n, $K^{-1/p}/(p+1)^{p+1}$ is greater than $2^{\log^{1-\epsilon/2} n}$. Thus a K-approximation algorithm for Total Label Cover$_p$ can be used to decide L. Thus Total Label Cover$_p$ is NP-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$. $\square$

B Schematic Representations of the Classification Theorems

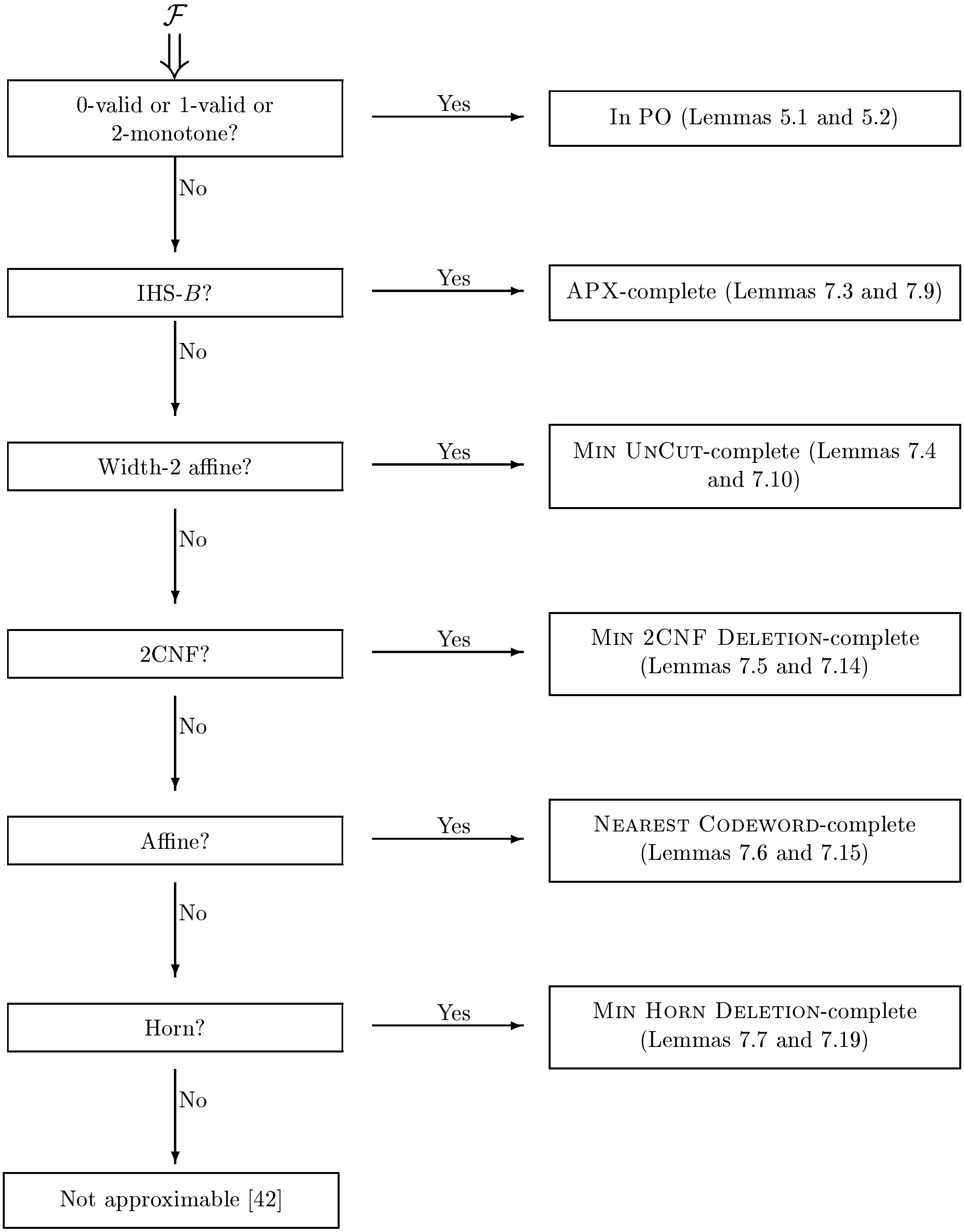
B.1 The Max CSP Classification


$\mathcal{F}$
0-valid or 1-valid or 2-monotone?
Yes
In PO (Lemmas 5.1 and 5.2)
No
APX-complete (Proposition 5.5 and Lemma 5.8)

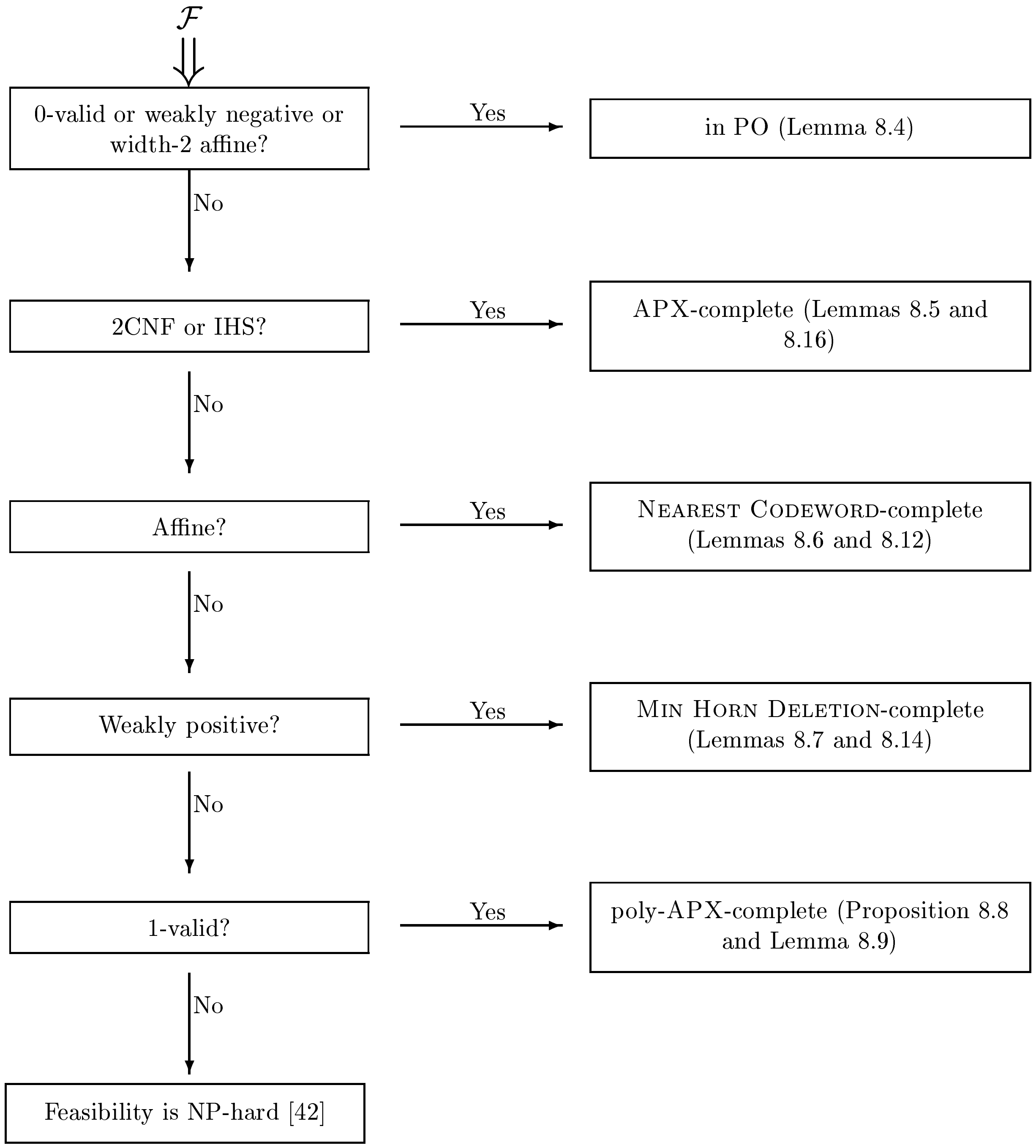
B.2 The Max Ones Classification


$\mathcal{F}$
1-valid or weakly positive or width-2 affine?
Yes
In PO (Lemma 6.5)
No
Affine?
Yes
APX-complete (Lemmas 6.6 and 6.13)
No
Strongly 0-valid or weakly negative or 2CNF?
Yes
poly-APX-complete (Proposition 6.7 and Lemma 6.14)
No
0-valid?
Yes
Not approximable (Lemma 6.23)
No
Feasibility is NP-hard [42]

B.3 The MIN CSP Classification


$\mathcal{F}$
0-valid or 1-valid or 2-monotone?
Yes
In PO (Lemmas 5.1 and 5.2)
No
IHS-B?
Yes
APX-complete (Lemmas 7.3 and 7.9)
No
Width-2 affine?
Yes
MIN UNCUT-complete (Lemmas 7.4 and 7.10)
No
2CNF?
Yes
MIN 2CNF DELETION-complete (Lemmas 7.5 and 7.14)
No
Affine?
Yes
NEAREST CODEWORD-complete (Lemmas 7.6 and 7.15)
No
Horn?
Yes
MIN HORN DELETION-complete (Lemmas 7.7 and 7.19)
No
Not approximable [42]

B.4 The Min Ones Classification


$\mathcal{F}$
0-valid or weakly negative or width-2 affine?
Yes
in PO (Lemma 8.4)
No
2CNF or IHS?
Yes
APX-complete (Lemmas 8.5 and 8.16)
No
Affine?
Yes
Nearest Codeword-complete (Lemmas 8.6 and 8.12)
No
Weakly positive?
Yes
Min Horn Deletion-complete (Lemmas 8.7 and 8.14)
No
1-valid?
Yes
poly-APX-complete (Proposition 8.8 and Lemma 8.9)
No
Feasibility is NP-hard [42]