A Complete Classification of the Approximability of Maximization Problems Derived from Boolean Constraint Satisfaction

Sanjeev Khanna* Madhu Sudan† David P. Williamson‡

Abstract

In this paper we study the approximability of boolean constraint satisfaction problems. A problem in this class consists of some collection of "constraints" (i.e., functions $f : \{0,1\}^k \rightarrow \{0,1\}$); an instance of a problem is a set of constraints applied to specified subsets of n boolean variables. Schaefer earlier studied the question of whether one could find in polynomial time a setting of the variables satisfying all constraints; he showed that every such problem is either in P or is NP-complete. We consider optimization variants of these problems in which one either tries to maximize the number of satisfied constraints (as in MAX 3SAT or MAX CUT) or tries to find an assignment satisfying all constraints which maximizes the number of variables set to 1 (as in MAX CUT or MAX CLIQUE). We completely classify the approximability of all such problems. In the first case, we show that any such optimization problem is either in P or is MAX SNP-hard. In the second case, we show that such problems fall precisely into one of five classes, assuming P $\neq$ NP: solvable in polynomial-time, approximable to within constant factors in polynomial time (but no better), approximable to within polynomial factors in polynomial time (but no better), not approximable to within any factor but decidable in polynomial time, and not decidable in polynomial time. This result proves formally for this class of problems two results which to this point have only been empirical observations; namely, that NP-hard problems in MAX SNP always turn out to be MAX SNP-hard, and that there seem to be no natural maximization problems approximable to within polylogarithmic factors but no better.

1 Introduction

In this paper, we study the approximability of optimization versions of boolean constraint satisfaction problems (CSPs). A boolean CSP consists of a collection $\mathcal{F}$ of boolean functions $f : \{0,1\}^k \rightarrow \{0,1\}$ called *constraints*. An instance of such a problem is a set of "constraint applications". Each application is a constraint drawn from $\mathcal{F}$ and applied to a specified subset of n boolean variables.

The decision version of a boolean constraint satisfaction problem asks whether there is an assignment to the variables such that all constraint applications are satisfied (that is, for each application the specified boolean function evaluates to 1 on the given subset of variables). For a collection of constraints $\mathcal{F}$, we call this problem SAT($\mathcal{F}$). Thus, for example, 3SAT is a decision version of boolean CSP with the constraint functions $f_1(x,y,z) = x \vee y \vee z$, $f_2(x,y,z) = \bar{x} \vee y \vee z$, and so on. Schaefer [17] studied the decision version of these problems and proved a remarkable result: for every such problem, either it is in P or it is NP-complete. This dichotomy is especially interesting in light of Ladner's theorem [12], which states that if P$\neq$NP, then there exist infinitely many problems of complexity between P and NP-complete. Thus although problems SAT($\mathcal{F}$) could in principle display a wide range of complexity, they in fact fall into distinct and quite separate classes (assuming P$\neq$NP). An additional property of Schaefer's result is that his characterization of the problems in P is compact. He gives six classes of functions, and if all functions in $\mathcal{F}$ fall entirely within any one of these classes, then SAT($\mathcal{F}$) is in P, otherwise it is NP-complete.

In this paper, we consider two different maximization versions of SAT($\mathcal{F}$) and completely classify the approximability of all such problems. In so doing, we find that these optimization problems also fall into distinct and separate classes, bypassing the many intermediate levels of approximability which are possible in principle. As we describe later, this classification proves formally

*sanjeev@research.bell-labs.com. Fundamental Mathematics Research Department, Bell Labs, 700 Mountain Avenue, NJ 07974. This work was performed when the author was at the Department of Computer Science, Stanford University, Stanford, CA 94305. He was supported by a Schlumberger Foundation Fellowship, an OTL grant, and NSF Grant CCR-9357849.

†madhu@watson.ibm.com. IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

‡dpw@watson.ibm.com. IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

for these problems some results which to this point have only been empirical observations. Furthermore, our classification of the problems also has a compact description. For both types of maximization versions of SAT($\mathcal{F}$), we refine Schaefer's classes, and the level of approximability of a problem for a given $\mathcal{F}$ is determined by which of these classes contain $\mathcal{F}$.

In the first maximization version of SAT($\mathcal{F}$) that we consider, for each instance of a problem we are also given a nonnegative weight w_i for each constraint application i, and we must try to find an assignment to the variables which maximizes the weight of the satisfied constraint applications. For any set of constraints $\mathcal{F}$, we call this associated maximization problem MAX CSP($\mathcal{F}$), and we call the class of all such problems MAX CSP. It follows almost immediately from its definition that MAX CSP is contained in the well-studied class MAX SNP. Conversely, it also contains many of its complete problems. For example, MAX 3SAT and MAX CUT can be cast as MAX CSP($\mathcal{F}$) problems. We show that each problem MAX CSP($\mathcal{F}$) is either solvable exactly in polynomial time or is MAX SNP-hard. Thus there is no problem in this class which has an approximation scheme but is not solvable in polynomial time. This result has been obtained independently by Creignou [4]; however our result is stronger in certain technical senses which we discuss later.

In the second maximization version of SAT($\mathcal{F}$) that we consider, for each instance of the problem we are also given a nonnegative weight w_i for each boolean variable, and we must try to find a boolean assignment of maximum weight that satisfies all constraint applications. For any set of constraints $\mathcal{F}$, we call this associated maximization problem MAX ONE($\mathcal{F}$); for example, MAX CUT and MAX CLIQUE can be cast as MAX ONE($\mathcal{F}$) problems. We show that each problem MAX ONE($\mathcal{F}$) must fall into one of five classes, assuming P $\neq$ NP: first, it is solvable exactly in polynomial time; second, it can be approximated to within some constant factor but no better; third, it can be approximated to within some factor that is polynomial in the number of variables, but no better; fourth, it is NP-complete to find a satisfying assignment of non-zero value; fifth, it is NP-complete to find any satisfying assignment.

The central idea of our proofs is a new concept we call an *implementation*. Given a set of constraints $\mathcal{F}$, we show that if $\mathcal{F}$ has certain properties, then it can be used to enforce other constraint functions f. We show that under suitable conditions, implementations can be composed, so that the constraints of $\mathcal{F}$ can be used to implement the constraints of other problems (such as MAX CUT or MAX CLIQUE) whose approximability is well known. The central difficulty of the proofs is showing that this can be done in an exhaustive way for all possible sets of constraints $\mathcal{F}$. Our definition of an implementation here is inspired by the notion of a gadget in Bellare et al. [3] and we unify their many definitions (they have different definitions for every f and $\mathcal{F}$ that they consider) into a single one. Our definition has in turn been used by Trevisan et al. [18] to derive improved hardness of approximation results and improved approximation algorithms.

Our results prove formally for these classes of problems some results about approximability which to this point have only been empirical observations. For example, the study of MAX SNP has revealed so far that every NP-hard MAX SNP problem is also hard to approximate to within some constant factor. Our result on MAX CSP serves as a formal basis for this empirical observation. Similarly, in the search for polynomial-time approximation algorithms, optimization problems so far either have exact algorithms, or approximation schemes, or constant or (poly)logarithmic or polynomial approximation algorithms – but this list is virtually exhaustive. There have been no "natural" problems that are approximable to within intermediate factors, such as $2^{\log^\epsilon n}$ or $\log\log n$ and no better. In addition, for many natural optimization problems the best known approximation algorithm guarantees logarithmic factor approximability, and yet none of them is a *maximization* problem. Once again, our results show that these observations are not simply due to a lack of knowledge, but have some formal basis.

One of the original motivations for this work was to find some simple rules which characterize the approximability of any given optimization problem. However the very general question, "Given an optimization problem, determine its approximability" is undecidable by Rice's Theorem (cf. [15], page 62)[1]. Hence we turned to restricted classes of uniformly presented optimization problems and this allowed us to achieve our goal. A natural next step in this research agenda is to broaden the classes of problems covered by this approach. Khanna et al. [11] have already extended this line of research to minimization problems, obtaining a complete classification for MIN CSP and MIN ONES. Perhaps one of the most interesting directions to extend this work is to study the approximability of these problems over a non-boolean domain. To begin with, a study over a domain of size 3 itself seems to require new techniques. In an elegant paper, Feder and Vardi [5] highlight many inherent problems in obtaining an analog of Schaefer's result over non-boolean domains. Other possible research directions include: (1) extending the function families that are studied (to include, say, functions of bounded range or functions of unbounded arity); (2) placing restrictions

[1] Here we are assuming the optimization problem is being presented by an arbitrary Turing machine which solves the optimization problem. More detailed study of the decidability of the problem, when the presentation of the optimization problem is more restricted is carried on in Merkle and the references there [14].

on the nature of the interaction between constraints and variables (such as bounding the number of times a variable can appear in a constraint). One such restriction which has been explored by Khanna and Motwani [8] is the case where the interaction graph of the constraint applications and the variables is planar.

Due to space limitations, we will focus our attention on presenting the result for the problems MAX ONE($\mathcal{F}$). In Section 2, we present some definitions and state our main results. We also state the ways in which our result for MAX CSP($\mathcal{F}$) strengthens that of Creignou. Section 3 defines implementations and states some basic properties of implementations. Section 4 outlines the proof of our result for MAX ONE($\mathcal{F}$). Details of omitted results and proofs may be found in the full version of this paper [10].

2 Definitions and Main Results

We begin with some definitions. A constraint f is as defined above, and a constraint application is a pair $(f,(i_1,\ldots,i_k))$, where the $i_j \in [n]$ indicate to which k of the n boolean variables the constraint is applied. We require that $i_j \neq i_{j'}$ for $j \neq j'$. While the 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 $C = (f,(i_1,\ldots,i_k))$ 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 part of C".

For a given set of constraints $\mathcal{F}$, SAT($\mathcal{F}$), MAX CSP($\mathcal{F}$), and MAX ONE($\mathcal{F}$) are as defined above. If for a given instance of MAX CSP($\mathcal{F}$) (MAX ONE($\mathcal{F}$)) the weights $w_i = 1$ for all i, we call this an *unweighted* instance of MAX CSP($\mathcal{F}$) (MAX ONE($\mathcal{F}$)). An instance which is not unweighted is *weighted*. We define the class of problems MAX CSP (MAX ONE) to be the set of all problems in MAX CSP($\mathcal{F}$) (MAX ONE($\mathcal{F}$)) taken over all possible sets of constraints $\mathcal{F}$.

We now need some definitions from the theory of approximation algorithms. Given an NPO (NP Optimization) problem Π and a function $\alpha : \mathcal{Z}^+ \to \mathcal{Z}^+$ (with $\alpha(\cdot) \geq 1$), we say that an algorithm A is an *α-approximation algorithm* for Π if for every instance $\mathcal{I}$ of Π of size n, A produces, in time polynomial in n, a solution s to $\mathcal{I}$ of value in the range $[\mathrm{OPT}(\mathcal{I})/\alpha(n), \alpha(n)\mathrm{OPT}(\mathcal{I})]$. We say Π is *α-approximable* if such an algorithm exists. We define APX to be the class of all NPO problems which have constant-factor approximation algorithms, and poly-APX to be the class of NPO problems which have polynomial-factor approximation algorithms.

We also need to define what it means to be hard to approximate a problem Π to within a factor of α. For a function $\alpha : \mathcal{Z}^+ \to \mathcal{Z}^+$ with $\alpha(\cdot) \geq 1$, an NP maximization problem Π is hard to approximate to within a factor of α if there exists a polynomial time reduction f from SAT to Π which maps instances of SAT of length n to instances of Π of length $l(n)$ and for every n and for any two instances ϕ_1, ϕ_2 of size n of SAT such that $\phi_1 \in$ SAT and $\phi_2 \notin$ SAT, $\mathrm{OPT}(f(\phi_1))/\mathrm{OPT}(f(\phi_2)) \geq \alpha(l(n))$. Thus a problem Π is APX-hard if there exists a constant function $\alpha_\Pi > 1$ such that Π is hard to approximate to within α_Π. A problem Π is poly-APX-hard if there exists an $\epsilon > 0$ such that Π is hard to approximate to within n^ϵ. A problem is APX-complete (poly-APX-complete) if it is in APX (poly-APX) and is APX-hard (poly-APX-hard).

It is usual to define completeness for approximation classes in terms of reducibility, rather than the hardness of approximation of the problem. However, Khanna et al. [9] have shown that these two notions are equivalent provided the right approximation preserving reductions are used. We will not go into these definitions here, and refer the reader to their paper for details.

We now describe the main constraint classes that are identified by Schaefer's and our results. We say a constraint f is *0-valid* (*1-valid*) if $f(\bar{0}) = 1$ ($f(\bar{1}) = 1$). It is *weakly positive* (*weakly negative*) if it can be expressed in conjunctive normal form, with all the disjuncts having at most one negated literal (positive literal). A constraint f is *affine* if it can be expressed as a conjunction of linear equalities over GF(2). And, finally, a constraint f is *2CNF* if it can be expressed in conjunctive normal form with all disjuncts having at most two literals.

The above six constraint classes can now be used to describe Schaefer's result. In what follows we use phrases such as "$\mathcal{F}$ is 0-valid" to imply that "every function $f \in \mathcal{F}$ is 0-valid". We stress that when we say something like "$\mathcal{F}$ is 0-valid or 1-valid", we mean that "every function in $\mathcal{F}$ is 0-valid or every function in $\mathcal{F}$ is 1-valid".

Theorem 2.1 [Schaefer [17]] For any constraint set $\mathcal{F}$, SAT($\mathcal{F}$) is either in P or is NP-complete. Furthermore, SAT($\mathcal{F}$) is in P if and only if $\mathcal{F}$ is 0-valid or 1-valid or weakly positive or weakly negative or affine or 2CNF.

We now provide the definitions required to state our main classification result for MAX CSP. For starters, observe that for the approximability of MAX CSP($\mathcal{F}$) does not change by removing (or adding) functions from $\mathcal{F}$ which are not satisfiable. Hence, given a constraint set $\mathcal{F}$, we define the constraint set $\mathcal{F}'$ to be the set of constraints f in $\mathcal{F}$ which are satisfiable. We also need to define one more class of constraints before we can give our result: we say a constraint f of arity k is *2-monotone* if there exist indices $i_1,\ldots,i_p \subset \{1,\ldots,k\}$ and $j_1,\ldots,j_q \subset \{1,\ldots,k\}$ such that $f(X_1,\ldots,X_k) = (X_{i_1} \wedge \cdots \wedge X_{i_p}) \vee (\bar{X}_{j_1} \wedge \cdots \wedge \bar{X}_{j_q})$.

Theorem 2.2 (MAX CSP **Classification**) For every constraint set $\mathcal{F}$, the problem MAX CSP($\mathcal{F}$) is always either in P or is APX-complete. Furthermore, it is in P if and only if $\mathcal{F}'$ is 0-valid or 1-valid or 2-monotone.

As stated previously, Theorem 2.2 was independently discovered by Creignou [4]. One fundamental point of difference between our result and hers is that we do not allow the use of variable replication in a constraint application. This is enforced by our definition of constraint application which insists that the indices $i_1, \ldots, i_k$ must be distinct. Our theorem shows that this does not ultimately matter, but this is not obvious a priori. For instance, a problem whose approximability has often been studied is the MAX EXACT kSAT problem: Given a collection of clauses of length exactly k, satisfy as many as possible. This problem is known to be approximable to within $1 + 2^{-k}/(1 - 2^{-k})$. However this problem cannot be captured as a MAX CSP problem under Creignou's notion of constraint application. The related problem that she can capture is MAX kSAT: Given a collection of clauses of length *at most* k, satisfy as many as possible. The best known approximation for this problem is only slightly smaller than 4/3 [19, 6, 7, 2]. Thus replications do end up altering the approximability of optimization problems and we take care to study the approximability of problems without the use of replications.

To give our result for MAX ONE, we need to give two more classes of constraints. We say a constraint is *affine with width* 2 if it can be expressed as a conjunction of linear equalities over GF(2) with at most two variables per equality constraint. A constraint f is *strongly 0-valid* if it is satisfied by any assignment with less than or equal to 1 ones. In the theorem below, we use the term "decidable" to mean that the decision version SAT($\mathcal{F}$) is in P.

Theorem 2.3 (MAX ONE **Classification**) For every constraint set $\mathcal{F}$, MAX ONE($\mathcal{F}$) is either solvable exactly in P or 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 MAX ONE($\mathcal{F}$) is in P.
2. Else if $\mathcal{F}$ is affine then MAX ONE($\mathcal{F}$) is APX-complete.
3. Else if $\mathcal{F}$ is strongly 0-valid or weakly negative or 2CNF then MAX ONE($\mathcal{F}$) is poly-APX complete.
4. Else if $\mathcal{F}$ is 0-valid then SAT($\mathcal{F}$) is decidable in P but finding a solution of positive value is NP-hard.
5. Else finding any feasible solution to MAX ONE($\mathcal{F}$) is NP-hard.

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 is the same as MAX CSP($\{X \oplus Y\}$). We will then have to reduce this problem to MAX CSP($\mathcal{F}$). The main technique we use to do this is to "implement" the constraint $X \oplus Y$ 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 $\vec{X} = \{X_1, X_2, ..., X_p\}$ and $\vec{Y} = \{Y_1, Y_2, ..., Y_q\}$ is called an α-implementation of a constraint $f(\vec{X})$ for a positive integer α iff the following conditions are satisfied:

(a) no assignment of values to $\vec{X}$ and $\vec{Y}$ can satisfy more than α constraints from $C_1, \ldots, C_m$.

(b) for any assignment of values to $\vec{X}$ such that $f(\vec{X})$ is true, there exists an assignment of values to $\vec{Y}$ such that precisely α constraints are satisfied,

(c) for any assignment of values to $\vec{X}$ such that $f(\vec{X})$ is false, no assignment of values to $\vec{Y}$ can satisfy more than $(\alpha - 1)$ constraints.

An implementation which satisfies the following additional property is called a *strict* α-implementation.

(d) for any assignment to $\vec{X}$ which does not satisfy f, there always exists an assignment to $\vec{Y}$ such that precisely $(\alpha - 1)$ constraints are satisfied.

A collection of m constraints is a *perfect* implementation of f if it is an m-implementation of f. 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 refer to the set $\vec{X}$ as the *constraint variables* and the set $\vec{Y}$ as the *auxiliary variables*.

A constraint f 1-implements itself strictly and perfectly. While properties (a)-(c) have perhaps been used implicitly elsewhere, property (d) is more strict (hence the name), but turns out to be critical in composing implementations together. The following lemma shows that the implementations of constraints compose together, if they are strict or perfect.

Lemma 3.2 If $\mathcal{F}_f$ strictly (perfectly) implements a constraint f, and $\mathcal{F}_g$ strictly (perfectly) implements a constraint $g \in \mathcal{F}_f$, then $(\mathcal{F}_f \setminus \{g\}) \cup \mathcal{F}_g$ strictly (perfectly) implements the constraint f.

Proof: Let $C_1, \ldots, C_{m_1}$ be constraint applications from $\mathcal{F}_f$ on variables $\vec{X}, \vec{Y}$ giving an α_1-implementation of f with $\vec{X}$ being the constraint variables. Let $C'_1, \ldots, C'_{m_2}$ be constraint applications from $\mathcal{F}_g$ on variable set $\vec{X}', \vec{Z}'$ yielding an α_2-implementation of g. 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 $(\{\mathcal{F}_f \setminus \{g\}) \cup \mathcal{F}_g$ on a set of variables $\vec{X}, \vec{Y}, \vec{Z}'_1, \ldots, \vec{Z}'_\beta$ as follows: We include the constraint applications $C_{\beta+1}, \ldots, C_{m_1}$ on variables $\vec{X}, \vec{Y}$ and for every constraint application C_j on variables $\vec{V}_j$ (which is a subset of variables from $\vec{X}, \vec{Y}$) we place the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$ on variable set $\vec{V}_j, \vec{Z}'_j$ with $\vec{Z}'_j$ being the auxiliary variables.

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

Consider any assignment to $\vec{X}$ satisfying f. Then any assignment to $\vec{Y}$ 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 $\vec{Z}'_j$ 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 $\vec{X}, \vec{Y}$, 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 (a). Property (c) is achieved similarly.

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

We now consider the case when the α_1- and α_2-implementations satisfy property (d) and show that in this case also the collection of constraints above satisfies property (b). Given an assignment to $\vec{X}$ satisfying f there exists an assignment to $\vec{Y}$ satisfying α_1 constraints from $C_1, \ldots, C_{m_1}$. Say this assigment 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 $\vec{X}, \vec{Y}$, there exists an assignment to $\vec{Z}'_j$ 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 $\vec{Z}'_j$ 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 $\vec{Y}, \vec{Z}'_1, \ldots, \vec{Z}'_\beta$ 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 (b). A similar analysis can be used to show property (d). ■

The following lemma shows a simple monotonicity property of implementations (proof omitted).

Lemma 3.3 For integers α, α' with $\alpha \leq \alpha'$, if $\mathcal{F}$ α-implements f then $\mathcal{F}$ α'-implements f. Furthermore strictness and perfectness are preserved under this transformation.

The next lemma now shows how to use perfect implementations for showing hardness of weighted MAX ONE problems.

Lemma 3.4 Given constraint sets $\mathcal{F}_1, \mathcal{F}_2$, such that the weighted MAX ONE$(\mathcal{F}_2)$ problem has a $\alpha(n)$-approximation algorithm and every constraint $f \in \mathcal{F}_1$ can be perfectly implemented by the constraint set $\mathcal{F}_2$, then there exist constants c, d such that the weighted MAX ONE$(\mathcal{F}_1)$ problem has a $\alpha(cn^d)$-approximation algorithm.

Proof: Let $l = |\mathcal{F}_1|$ and k be the maximum arity of any constraint $f \in \mathcal{F}_1$. Let K be the largest number of auxiliary variables used in perfectly implementing any constraint $f \in \mathcal{F}_1$ by $\mathcal{F}_2$. Notice that K is a finite constant for any fixed $\mathcal{F}_1$, $\mathcal{F}_2$.

Given an instance $\mathcal{I}$ of MAX ONE$(\mathcal{F}_1)$ with m constraints $C_1, \ldots, C_m$ on n variables $X_1, \ldots, X_n$, with n real non-negative weights $w_1, \ldots, w_n$, we create an instance $\mathcal{I}'$ of MAX ONE$(\mathcal{F}_2)$ as follows: $\mathcal{I}'$ has the variables $X_1, \ldots, X_n$ of $\mathcal{I}$ and in addition "auxiliary" variables $\{Y_i^j\}_{i=1,j=1}^{m,K}$. The weights corresponding to $X_1, \ldots, X_n$ is $w_1, \ldots, w_n$ (same as in $\mathcal{I}$) and the auxiliary variables Y_i^j have weight zero. The constraints of $\mathcal{I}'$ perfectly implement the constraints of $\mathcal{I}$. In particular the constraint $f_i(X_{i_1}, \ldots, X_{i_k})$ of $\mathcal{I}$ is implemented by a collection of constraints from $\mathcal{F}_2$ (as dictated by the perfect implementation of f_i by $\mathcal{F}_2$) on the variables $(X_{i_1}, \ldots, X_{i_k}, Y_i^1, \ldots, Y_i^K)$.

By the definition of perfect implementations, it is clear that the every feasible solution to $\mathcal{I}$ can be extended (by some assignment to the Y variables) into a feasible solution to $\mathcal{I}'$. Alternately, every solution to $\mathcal{I}'$ immediately projects on to a solution of $\mathcal{I}$. Furthermore, the value of the objective function is exactly the same (by our choice of weights). Thus a β-approximate solution to $\mathcal{I}'$ gives a β-approximate solution to $\mathcal{I}$.

It remains to study this approximation as a function of the instance size. Observe that the instance size of $\mathcal{I}'$ is much larger. Let N denote the number of variables in $\mathcal{I}'$. Then N is upper bounded by $Km + n$, where m is the number of constraints in $\mathcal{I}$. But m, in turn, is at most ln^k. Thus $N \leq (K+1)ln^k$, implying that an $\alpha(N)$-approximate solution to $\mathcal{I}'$, gives an $\alpha((K+1)ln^k)$-approximate solution to $\mathcal{I}$. Thus an $\alpha(N)$-approximation algorithm for the weighted MAX ONE($\mathcal{F}_2$) problem yields an $\alpha(cn^d)$-approximation for the weighted MAX ONE($\mathcal{F}_1$)-problem, for $c = (K+1)l$ and $d = k$. ■

Corollary 3.5 If weighted MAX ONE($\mathcal{F}_1$) is APX-hard and $\mathcal{F}_2$ perfectly implements every constraint in $\mathcal{F}_1$, then weighted MAX ONE($\mathcal{F}_2$) is APX-hard. Similarly, if weighted MAX ONE($\mathcal{F}_1$) is poly-APX-hard and $\mathcal{F}_2$ perfectly implements every constraint in $\mathcal{F}_1$, then weighted MAX ONE($\mathcal{F}_2$) is poly-APX-hard.

4 The Classification Theorem for MAX ONE

In this section, we establish Theorem 2.3. We use the following shorthand notation for the eight constraints classes of importance. Let $\mathcal{F}_1$ denote the class of 1-valid constraints, $\mathcal{F}_2$ the weakly positive constraints, $\mathcal{F}_3$ the affine width-2 constraints, $\mathcal{F}_4$ the affine constraints, $\mathcal{F}_5$ the strongly 0-valid constraints, $\mathcal{F}_6$ the weakly negative constraints, $\mathcal{F}_7$ the 2CNF constraints and $\mathcal{F}_8$ the 0-valid ones. Theorem 2.3 can be restated as follows. For a constraint set $\mathcal{F}$ if i is the smallest index such that $\mathcal{F} \subset \mathcal{F}_i$, then if $i \in \{1,2,3\}$ then MAX ONE($\mathcal{F}$) $\in$ P, if $i = 4$ then MAX ONE($\mathcal{F}$) is APX-complete, if $i \in \{5,6,7\}$ then MAX ONE($\mathcal{F}$) is poly-APX-complete, if $i = 8$ then SAT($\mathcal{F}$) is in P but MAX ONE($\mathcal{F}$) is not approximable and if no such i exists then finding any satisfying assignment for MAX ONE($\mathcal{F}$) in NP-hard.

4.1 Preliminaries

In this subsection, we prove a few preliminary lemmas that we will need in the proof of the theorem, particularly in Cases 2 and 3. We first show that in these cases, it is essentially equivalent for us to consider the weighted or unweighted MAX ONE($\mathcal{F}$) problem.

We begin with a slightly stronger definition of polynomial-time solvability of SAT($\mathcal{F}$) that we will need. We then show that given this stronger form of SAT($\mathcal{F}$) that insofar as APX-hardness and poly-APX-hardness are concerned, the weighted and unweighted cases of MAX ONE($\mathcal{F}$) are equivalent. We conclude by showing that in Cases 2 and 3 the stronger form of SAT($\mathcal{F}$) holds.

Definition 4.1 We say that a constraint satisfaction problem SAT($\mathcal{F}$) is *strongly decidable* if given m constraints on n variables $X_1, \ldots, X_n$ *and an index* $i \in \{1, \ldots, n\}$, there exists a polynomial time algorithm which decides if there exists an assignment to $X_1, \ldots, X_n$ satisfying all m constraints and additionally satisfying the property $X_i = 1$.

Lemma 4.2 For every strongly decidable constraint set $\mathcal{F}$, for every ϵ of the form $1/l$ for some positive integer l and for every non-decreasing function $\alpha : \mathcal{Z}^+ \to \mathcal{Z}^+$, α-approximating the weighted MAX ONE($\mathcal{F}$) problem reduces to α'-approximating the (unweighted) MAX ONE($\mathcal{F}$) problem, where $\alpha'(n) = \frac{\alpha(\sqrt{\epsilon n})}{(1+\epsilon)}$.

Proof Sketch: The basic idea of the proof is that we can replicate variables many times (in proportion to its weight) and then for every replicated copy of a variable, we place all the constraints that were placed on the original copy of the variable. If the original weights are polynomially bounded then the new instance still has polynomial size.

The only issue to be taken care of is that the weights of the original instance need not be polynomially bounded. To take care of this, we first find the largest weight element which is set to 1 in some satisfying assignment. Here we use the strong decidability of the family $\mathcal{F}$ to find this element. Having done so, we can essentially ignore elements of larger weight and also afford to round up the weight of all smaller elements to integral multiples of ϵ/n times the weight of this element. This only increases the contribution of any assignment by a factor of $(1+\epsilon)$. After this transformation, we are reduced to the polynomially bounded case as desired. ■

The ability to work with weighted problems in combination with Lemma 3.4 allows us to use existential quantification over auxiliary variables and the notion of perfect implementations of constraints.

As our examination will eventually show, there is really no essential difference in the approximability of the weighted and unweighted problems. For now we will satisfy ourselves by stating this conditionally.

Corollary 4.3 For any strongly decidable constraint set $\mathcal{F}$, the MAX ONE($\mathcal{F}$) problem is APX-hard if and only if the weighted MAX ONE($\mathcal{F}$) problem is APX-hard. Similarly, the MAX ONE($\mathcal{F}$) problem is poly-APX-hard if and only if the weighted MAX ONE($\mathcal{F}$) problem is poly-APX-hard.

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

Lemma 4.4 If $\mathcal{F} \subset \mathcal{F}_j$ for some $j \in \{1, \ldots, 7\}$, then $\mathcal{F}$ is strongly decidable.

Lastly we describe one more tool that comes in useful in creating reductions. This is the notion of implementing a property which falls short of being an implementation of an actual constraint. The target constraints in the following definitions are the constraints which force variables to being constants (either 0 or 1). However, sometimes we are unable to achieve this. So we end up implementing a weaker form which suffices for our applications. We next describe this property.

Definition 4.5 [Existential Zero (One)] A constraint set $\mathcal{F}$ can implement the existential zero (one) property if there exists a set of m constraints $f_1, \ldots, f_m$ over n variables $\vec{X}$ and an index $k \in \{1, \ldots, n\}$ such that the following hold:

- There exists an assignment $V_{k+1}, \ldots, V_n$ to $X_{k+1}, \ldots, X_n$ such that assigning zero (one) to the first k variables $X_1, \ldots, X_k$ satisfies all constraints.
- Conversely, every assignment satisfying all the constraints must make at least one of the variables in $X_1, \ldots, X_k$ zero (one).

Definition 4.6 Given a constraint f of arity k and a set $S \subset \{1, \ldots, k\}$, the constraint $f|_{(S,0)}$ is a constraint of arity $k - |S|$ given by $f|_{(S,0)}(X_1, \ldots, X_{k-|S|}) = f(X_1, 0, 0, X_2, \ldots, X_{k-|S|}, 0, 0)$, where the zeroes occur in the indices contained in S. For a constraint set $\mathcal{F}$, the 0-closure of $\mathcal{F}$, denoted $\mathcal{F}|_0$ is the set of constraints $\{f_{(S,0)} | f \in \mathcal{F}, S \subset \{1, \ldots, k\}\}$. (1-closure may be defined similarly.)

Notice that $\mathcal{F}|_0$ essentially implements every constraint that can be implemented by $\mathcal{F} \cup \{F\}$, except the constraint $\{F\}$, where F stands for the unary constraint "false". We define $\mathcal{F}|_1$ similarly. Then $\mathcal{F}|_{0,1} = \mathcal{F}|_0 \cup \mathcal{F}|_1$.

Lemma 4.7 If a constraint set $\mathcal{F}$ can implement the existential zero property, then $\mathcal{F}$ perfectly implements every constraint in the constraint set $\mathcal{F}|_0$. Similarly, if a constraint set $\mathcal{F}$ can implement the existential one property, then $\mathcal{F}$ perfectly implements every constraint in the constraint set $\mathcal{F}|_1$.

4.2 Proof of Main Theorem

4.2.1 Cases 1, 4, and 5

We now begin our proof of Theorem 2.3. We start with the sub-cases that are easier to prove and then move on to the more difficult sub-cases. Cases 4 and 5 of Theorem 2.3 are indirect and direct consequences of Schaefer's theorem, respectively; we omit their proofs. Case 1 is relatively simple, and we sketch its proof below.

Lemma 4.8 The weighted MAX ONE($\mathcal{F}$) problem is in P if each $\mathcal{F}$ is 1-valid or is weakly positive or is affine with width 2.

Proof Sketch: The first two cases are easy and are henceforth omitted. In the case that $\mathcal{F}$ is affine with width 2, we reduce the problem of finding a feasible solution to 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; 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 so, then for each connected component of the graph choose the larger weight side of the bipartition, and set the corresponding variables to one. ■

4.2.2 Case 3: The poly-APX-Complete Case

The proofs of Cases 2 and 3 are much more difficult. For space reasons, we omit the proof of Case 2 and concentrate on Case 3. We first show that the problems in this case are in poly-APX.

Lemma 4.9 If $\mathcal{F} \subset \mathcal{F}_i$ for some $i \in \{1, 2, 3, 4, 5, 6, 7\}$ then MAX ONE($\mathcal{F}$) can be approximated to within a factor of n.

Proof: Schaefer's results imply a polynomial-time algorithm to compute a feasible solution. If the feasible solution has at least one 1, we are done. Else, iteratively try setting every variable to one and computing a feasible solution. Note that if $\mathcal{F}$ is affine (or 2CNF), then the constraints obtained by restricting some variable to be 1 remains affine (or resp. 2CNF), and thus this new class is still decidable. Lastly, a strongly 0-valid constraint set remains 0-valid after this restriction and is still decidable. If the decision procedure gives no non-zero solution, then the optimum is zero, else we output a solution of value at least 1. ■

We now turn to showing that this class of problems is poly-APX-hard. Our goal will be to perfectly implement the constraint $\bar{X}_1 + \cdots + \bar{X}_k$, for some $k \geq 2$. The following lemma shows that this will imply poly-APX-hardness.

Lemma 4.10 If $f = \bar{X}_1 + \cdots + \bar{X}_k$, then MAX ONE($\{f\}$) is poly-APX-hard.

Proof: We do a reduction from MAX CLIQUE, which is known to be poly-APX-hard [1]. Given a graph G, construct a MAX ONE($\{f\}$) 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 ONE($\{f\}$) 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 the existence of an α-approximation algorithm for the MAX ONE($\{f\}$) problem implies the existence of a $(k-1)\alpha$-approximation algorithm to the clique problem. The poly-APX-hardness of clique now implies the lemma. ■

The following lemma divides the remainder of the proof into two cases.

Lemma 4.11 If $\mathcal{F} \subset \mathcal{F}_i$ for some $i \in \{5, 6, 7\}$, but $\mathcal{F} \not\subset \mathcal{F}_j$ for any $j \in \{1, 2, 3, 4\}$, then either $\mathcal{F}$ perfectly implements every constraint in $\mathcal{F}|_{0,1}$ or $\mathcal{F}$ perfectly implements $\mathcal{F}|_0$ and every constraint in $\mathcal{F}$ is 0-valid.

We show that in the first case, we can perfectly implement $\bar{X} + \bar{Y}$. We will then turn to the case in which all constraints are 0-valid and show that we can either perfectly implement $\bar{X}_1 + \cdots + \bar{X}_k$ or an existential one. If we can implement an existential one, then we are in the same situation as the first case. This will complete the proof of poly-APX-hardness.

Recall that we have a constraint in $\mathcal{F}$ that is not weakly positive and a constraint that is not affine.

Case I : $\mathcal{F}$ perfectly implements every constraint in $\mathcal{F}|_{0,1}$.

Lemma 4.12 If f is not weakly positive, then the constraint set $\{f\}|_{0,1}$ perfectly implements either XOR or $\bar{X} + \bar{Y}$.

Proof: Let $C = (\bar{X}_1 + \cdots + \bar{X}_p + Y_1 + \cdots + Y_q)$ be a maxterm in f with more than one negation i.e. $p \geq 2$. Substituting a 1 in place of variables $\bar{X}_3, \bar{X}_4, \ldots, \bar{X}_p$, 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 f' such that $(\bar{X}_1 + \bar{X}_2)$ is a maxterm in f'.

By definition of maxterm, f' must be satisfied whenever $X_1 \oplus X_2 = 1$. Now if f' is also satisfied when $X_1 = X_2 = 0$, we get the constraint $\bar{X}_1 + \bar{X}_2$, else we get the constraint XOR(X_1, X_2). ■

Lemma 4.13 The constraint set $\{\text{XOR}\}$ can perfectly implement the constraint REP.

Lemma 4.14 Let g be a non-affine constraint. Then the constraint set $\{g, \text{REP}, \text{XOR}\}|_{0,1}$ can either perfectly implement the constraint $(X + \bar{Y})$ or $(\bar{X} + \bar{Y})$.

Proof: Since g is non-affine, we essentially have the following situation for three satisfying assignments s_1, s_2 and s_3 for g.

									$g()$
s_1	0	0	0	0	1	1	1	1	1
s_2	0	0	1	1	0	0	1	1	1
s_3	0	1	0	1	0	1	0	1	1
$s_1 \oplus s_2 \oplus s_3$	0	1	1	0	1	0	0	1	0
	0	X	Y	Z	$\bar{Z}$	$\bar{Y}$	$\bar{X}$	1	

(where each column may actually be repeated more than once). Fixing the above variables to 0's and 1's as shown in the last row, and using replicated copies of three variables X, Y and Z (and their negations using XOR), we get a constraint $h(X, Y, Z)$ with the truth-table in Figure 1.

x \ yz	00	01	11	10
0	1	B	1	A
1	C	1	D	0

Figure 1: Truth-table of the constraint $h(X, Y, Z)$

The undetermined values in the table are indicated by the parameters A, B, C and D. The following analysis shows that for every possible value of these parameters, we can indeed perfectly implement an OR constraint using the constants 0 and 1.

$$\begin{aligned}
A = 0 &\implies \exists\, X\ h(Y, Z) = Y + \bar{Z} \\
A = 1, B = 0 &\implies h(0, Y, Z) = \bar{Y} + Z \\
A, B = 1, C = 0 &\implies h(X, 0, Z) = X + \bar{Z} \\
A, B, C = 1, D = 0 &\implies h(X, Y, 1) = \bar{X} + \bar{Z} \\
A, B, C, D = 1 &\implies h(1, Y, Z) = Y + \bar{Z}
\end{aligned}$$

■

Lemma 4.15 The constraint set $\{X + \bar{Y}, \text{XOR}\}$ perfectly implements the constraint $\bar{X} + \bar{Y}$.

Proof: To perfectly implement $\bar{X} + \bar{Y}$, we create an auxiliary variable X'. We now add two constraints, namely $X' + \bar{Y}$, and XOR(X, X'). Clearly, all constraints are satisfied only if $\bar{X} + \bar{Y}$ is satisfied. ■

Thus in all cases we are able to implement the constraint $\bar{X} + \bar{Y}$.

Case II : $\mathcal{F}$ can perfectly implement all constraints in $\mathcal{F}|_0$ and all constraints are 0-valid.

We now show that either we can perfectly implement $\bar{X} + \bar{Y}$, or perfectly implement a 1. If the former occurs, we are done, and if the latter, we can reduce to the previous case.

Lemma 4.16 If f is 0-valid and not weakly positive, then the constraint set $\{f\}|_0$ either perfectly implements $\bar{X}_1 + \cdots + \bar{X}_k$ for some $k \geq 2$ or it perfectly implements $X + \bar{Y}$ or REP.

Proof: Let $C = (\bar{X}_1 + \cdots + \bar{X}_p + Y_1 + \cdots + Y_q)$ be a maxterm in f with more than one negation i.e. $p \geq 2$ (such a maxterm exists since f is not weakly positive). 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 $(\bar{X}_1 + \bar{X}_2 + \cdots + \bar{X}_p)$ 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 0-valid. WLOG assume that s assigns value 1 to the variables $X_1, X_2, \ldots, X_k$ and 0's 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 $g' = (\bar{X}_1 + \bar{X}_2 + \cdots + \bar{X}_k)$. If $k > 1$, then this perfectly implements the constraint $(\bar{X}_1 + \cdots + \bar{X}_k)$ 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 easily follows from C being a maxterm in g. WLOG 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 $(X_1 + \bar{X}_2)$ or $\mathrm{REP}(X_1, X_2)$. The lemma follows. ■

If we can perfectly implement $X + \bar{Y}$, then the following lemma shows that we can essentially perfectly implement a 1, and thus we can reduce to Case I. We use the constraint function $T(X_i) = X_i$ to represent constraints $X_i = 1$.

Lemma 4.17 If $\mathrm{MAX\ ONE}(\mathcal{F} \cup \{X + \bar{Y}\})$ is α-approximable for some function α, then so is $\mathrm{MAX\ ONE}(\mathcal{F} \cup \{T\})$.

Proof: Given an instance $\mathcal{I}$ of $\mathrm{MAX\ ONE}(\mathcal{F} \cup \{T\})$ we construct an instance $\mathcal{I}'$ of $\mathrm{MAX\ ONE}(\mathcal{F} \cup \{X + \bar{Y}\})$ 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 $X_i = 1$ for some variables X_i (imposed by the constraint T). We simulate this constraint in $\mathcal{I}'$ with $n - 1$ constraints of the form $X_i + \bar{X}_j$ for every $j \in \{1, \ldots, n\}$, $j \neq i$. 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 every $X_j = 0$. Thus the resulting instance of $\mathrm{MAX\ ONE}(\mathcal{F} \cup \{X + \bar{Y}\})$ has the same objective function and the same feasible space and is hence at least as hard as the original problem. ■

Now by Lemma 4.16 the only remaining subcase is if we can perfectly implement REP. The following lemma shows that in this case we can either perfectly implement $\bar{X} + \bar{Y}$ or $X + \bar{Y}$. If we can do the former, we are done, and if the latter, we can use $X + \bar{Y}$ to perfectly implement the T constraint, and reduce to the previous case. Hence in either case we are finished.

Lemma 4.18 If f is 0-valid constraint and non-affine, then $\mathrm{MAX\ ONE}(\{f, \mathrm{REP}\})$ perfectly implements either the constraint $(\bar{X} + \bar{Y})$ or the constraint $(X + \bar{Y})$.

Proof: Schaefer [17] shows that if f is a non-affine constraint, 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. Using this fact and the fact that f is 0-valid, we essentially have the following situation:

					$g()$
	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
	00...0	$XX...X$	$YY...Y$	$ZZ...Z$	

Fixing the above variables to 0's as shown in the last row, and assigning replicated copies of three variables X, Y and Z, we get a constraint $h(X, Y, Z)$ with the truth-table in Figure 1. The lemma now follows using an analysis identical to the one used in Lemma 4.14. ■

Acknowledgments

Many thanks to Nadia Creignou, Oded Goldreich, Greg Sorkin and Luca Trevisan for their valuable comments.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the intractability of approximation problems. *Proceedings of the* 33rd IEEE FOCS, IEEE, 1992.

[2] T. Asano, T. Ono, and T. Hirata. Approximation algorithms for the maximum satisfiability problem. *Scandanavian Workshop on Algorithmic Theory* 96 *Proceedings*, Lecture Notes in Computer Science Vol. 1097, ed., Springer-Verlag, 1996.

[3] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCP and non-approximability — towards tight results. (Version 3). *ECCC Technical Report* number TR95-024, 1995.

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

[5] T. Feder and M. Vardi. Monotone monadic SNP and constraint satisfaction. *Proceedings of the* 25th ACM STOC, ACM, 1993.

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

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

[8] S. Khanna and R. Motwani. Towards a Syntactic Characterization of PTAS, pp. 329–337. *Proceedings of the* 28th ACM STOC, ACM, 1996.

[9] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On Syntactic versus Computational Views of Approximation. *Proceedings of the* 35th IEEE FOCS, IEEE, 1994, pp. 819–830.

[10] S. Khanna, M. Sudan, and D.P. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. *Electonic Colloquium on Computational Complexity*, Technical report no. TR96-062, 1996.

[11] S. Khanna, M. Sudan and L. Trevisan. Constraint Satisfaction: The Approximability of Minimization Problems. To appear in the *Proceedings of the 12th Annual IEEE Computational Complexity Conference (CCC), 1997.*

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

[13] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In Proceedings of International Colloquium on Automata, Languages and Programming, ICALP, pp. 40–51, 1993.

[14] W. Merkle. Structural properties of bounded relations with an application to NP optimization problems. *Technical Report*, Mathematical Institute, University of Heidelberg, (`http://math.uni-heidelberg.de/logic/`), December 1996.

[15] C. Papadimitriou. *Computational Complexity.* Addison Wesley Publishing Company, 1994.

[16] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43, pp. 425–440, 1991.

[17] T. Schaefer. The complexity of satisfiability problems *Proceedings of the* 10th ACM STOC, ACM, 1978.

[18] L. Trevisan, G. Sorkin, M. Sudan and D. Williamson. Gadgets, approximation and linear programming. *Proceedings of the* 37th IEEE FOCS, IEEE, 1996.

[19] M. Yannakakis, On the approximation of maximum satisfiability. *Journal of Algorithms*, vol. 17, pages 475–502, 1994.