*On the Advantage over a Random Assignment**

Johan Håstad,[1]'† S. Venkatesh[2]'‡
[1]*Nada, KTH, SE-100 44 Stockholm, Sweden;*
e-mail: johanh@nada.kth.se
[2]*DIMACS Center, CoRE Building, Rutgers University, 96 Frelinghuysen Road, Piscataway, New Jersey 08854;*
e-mail: venkat@dimacs.rutgers.edu

Received 19 July 2002; accepted 6 November 2003; received in final form 15 May 2004
DOI 10.1002/rsa.20031
Published online 23 June 2004 in Wiley InterScience (www.interscience.wiley.com).

ABSTRACT: We initiate the study of a new measure of approximation. This measure compares the performance of an approximation algorithm to the random assignment algorithm. This is a useful measure for optimization problems where the random assignment algorithm is known to give essentially the best possible polynomial time approximation. In this paper, we focus on this measure for the optimization problems Max-Lin-2 in which we need to maximize the number of satisfied linear equations in a system of linear equations modulo 2, and Max-k-Lin-2, a special case of the above problem in which each equation has at most k variables. The main techniques we use, in our approximation algorithms and inapproximability results for this measure, are from Fourier analysis and derandomization. © 2004 Wiley Periodicals, Inc. Random Struct. Alg., 25: 117–149, 2004

Keywords: linear system of equations; inapproximability; PCP; approximation algorithms; satisfiability

---

Correspondence to: J. Håstad
*This work was done while the authors were at the Institute for Advanced Study, Princeton, NJ, USA during the academic year 2000–2001. A preliminary version of this work appeared in the Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002.
†Supported in part by the Göran Gustafsson Foundation and NSF Grant CCR-9987077.
‡Supported by a joint IAS–DIMACS postdoctoral fellowship.

1. INTRODUCTION

Given any optimization problem, one can ask if there is an efficient algorithm that finds the optimal solution. The theory of NP-completeness allows us to prove that many explicit problems do not allow for efficient algorithms assuming that NP contains difficult problems. In particular, it has been known for a long time that many natural optimization problems are NP-hard. If we assume that P $\neq$ NP, none of these problems have a worst case polynomial time algorithm finding optimal solutions.

Given the evidence that it is probably hard to develop efficient algorithms that give optimal solutions for these problems, it is natural to ask if it is possible to design efficient algorithms for these problems that give reasonably good solutions for all instances. Usually, an algorithm is said to be a c-approximation algorithm for a maximization problem if it, for each instance, produces a solution whose objective value is at least OPT/c, where OPT is the global optimum. A similar definition can be given for minimization problems. The theory of approximability of NP-hard problems has been an active area of research in the past decade. Many new approximation algorithms have been designed for a variety of optimization problems. Many inapproximability results have also been proved based on plausible complexity theory assumptions.

Consider any algorithm that is designed to solve a maximization problem. A general criterion to evaluate its performance is

$$\frac{OPT - X}{ALG - X}$$

where OPT is the optimum, ALG is the objective value of the solution output by the algorithm and X is a parameter to be chosen. In the usual definition, X is chosen to be zero since in most optimization problems, all feasible solutions have nonnegative values. However, there are interesting variants that have been studied. For example, there are optimization problems like quadratic programming where the value of the optimum solution could be negative. In such cases, choosing X to be the minimum possible value of a feasible solution is an appropriate choice. Such a definition was used by Bellare and Rogaway [8] in their inapproximability result for quadratic programming.

Another possible variant is to fix a polynomial time algorithm A, and let X be the value of the solution obtained by A. If A is randomized, we can let X be the expected value of the solution obtained by A. Such a definition compares the performance of any algorithm with a fixed and known polynomial time algorithm A. This definition can particularly lead to useful information for many optimization problems if A represents a broad class of algorithms that is known to give the best approximation algorithm for a wide range of optimization problems.

In this paper, we focus on a class of optimization problems called constraint satisfaction problems. A constraint satisfaction problem has an underlying Boolean predicate P. An instance of this problem is given by a collection of constraints C and the goal is to find an assignment to the variables that maximizes the number of satisfied constraints in C under the predicate P. Håstad [14] has shown that for many constraint satisfaction problems like Max-Lin-p, in which we are required to maximize the number of satisfied equations in a system of linear equations modulo a prime p, and Max-E3-Sat, in which we are required to maximize the number of satisfied clauses in a 3-CNF formula, the random

assignment algorithm essentially yields the best possible approximation ratio. This makes the study of a new measure of approximation that compares the performance of an algorithm for a constraint satisfaction problem with the random assignment algorithm interesting. We study this measure specifically for Max-Lin-2 and its special case Max-k-Lin-2 where each equation contains at most k variables. We also get similar results for other problems and point out some connections to the question of learning parity with noise. In a related work, Alon, Gutin, and Krivelevich [2] recently studied a new measure for approximation algorithms called the domination ratio. They obtain deterministic polynomial time algorithms that achieve domination ratio $1 - o(1)$ for the partition problem and domination ratio $\Omega(1)$ for the Max-Cut problem. Some of the techniques used by them are similar to those in this paper.

1.1. Our Results

In this paper, we focus on the following optimization problem: We are given a system of m linear equations modulo 2 in n variables, together with positive weights w_i, $1 \leq i \leq m$. To avoid degenerate cases, we state our results for the case that $m \geq n$. The goal is to output an assignment to the variables that maximizes the total weight of the satisfied equations. We use $\{+1, -1\}$-notation for Boolean values with -1 corresponding to true. In this notation, addition modulo 2 is multiplication and we write equation i as

$$\prod_{j \in \alpha_i} x_j = b_i.$$

where each α_i is a subset of $[n]$ and $b_i \in \{+1, -1\}$.

We consider two cases: Max-k-Lin-2, in which each α_i is of size at most k and Max-Lin-2, the general case without any restrictions. If W is the total weight of all equations, that is, $W = \sum_{i=1}^{m} w_i$, our performance measure is given by

$$\max_L \frac{SAT[OPT(L)] - W/2}{SAT[ALG(L)] - W/2}, \tag{1}$$

where L is an instance, $SAT[OPT(L)]$ denotes the total weight of equations satisfied by the optimal solution and $SAT[ALG(L)]$ denotes the total weight of equations satisfied by the solution output by the algorithm ALG. We note that (1) compares the performance of an approximation algorithm for Max-Lin-2 with the random assignment algorithm. It measures how much of the gap between the optimal solution and the solution of the random assignment algorithm is recovered by an approximation algorithm. Such a definition can be given for any constraint satisfaction problem.

Throughout the discussion, we assume that each equation in our system is nontrivial. In other words, we do not allow equations with $\alpha = \emptyset$. We also assume that $\alpha_i \neq \alpha_{i'}$ for $i \neq i'$. If this is not the case, it can be achieved as follows. If the left-hand side of two equations are equal, they can be merged to one equation with the appropriate weight. If the right-hand sides are the same, the new weight is the sum of the original weights, and, if they are different, then the new weight is the difference of the original weights and the right-hand side is that of the equation with higher weight.

As a starting point, let us first indicate how we can obtain a nonnegative objective

value. We assign values to the variables sequentially and simplify the system of equations as we go along. When we are about to give a value to x_j, we consider all equations reduced to the form $x_j = b$ for a constant b. We choose a value for x_j satisfying at least half (in the weighted sense) of these equations. It is easy to see that the total weight of the satisfied equations is at least the total weight of the falsified equations. By being careful, a m-approximation algorithm can be obtained as follows.

Pick the equation with largest weight. We construct a solution that satisfies this equation and at least half (in the weight sense) of the rest of the equations. This is sufficient to ensure that we have a m-approximation algorithm.

Let us assume, without loss of generality, that it is the first equation and that α_1 contains the variable x_1. For any α_i, $i > 1$, that contains x_1, replace the ith equation

$$\prod_{j \in \alpha_i} x_j = b_i$$

by

$$\prod_{j \in \alpha_i \Delta \alpha_1} x_j = b_i b_1,$$

where Δ denotes the symmetric difference. The resulting $m - 1$ equations now do not contain the variable x_1. Now obtain, as described above, an assignment to the variables $x_2, \ldots, x_n$ such that at least half (by weight) of the remaining $m - 1$ equations are satisfied and finally give x_1 a value to satisfy the first equation.

Is it possible to improve the algorithm by picking another equation and repeating the variable elimination step described above? The problem is that the first variable elimination step could create pairs of equations with equal weights and with the same left-hand side but with different values on the right-hand side. These equations would then, as discussed above, cancel each other, possibly resulting in an empty system of equations. If, however, the chosen equation is satisfied by the optimal assignment, the situation is easy to control. This is the basis of one of our approximation algorithms.

Previous results give some bounds for our current measure. In particular, using Håstad's results [14], it can be shown, for $k \geq 3$, that it is hard to approximate Max-k-Lin-2 (and hence Max-Lin-2) within c for every $c > 1$ unless NP = P and within $(\log m)^c$ for some constant $c > 0$ unless NP $\subseteq$ DTIME$[m^{O(\log \log m)}]$.

1.1.1. Bounded Size Equations. We first consider the case when each equation has at most k variables for some small k. We start with a randomized approximation scheme for our new measure.

Theorem 1.1. *Consider Max-k-Lin-2. There exists a fixed constant $c > 1$ such that the following holds: For any $k \in O(\log n)$, there is a randomized polynomial time algorithm that, with probability at least 3/4, outputs an assignment that gives an approximation ratio at most $c^k\sqrt{m}$.*

By running the algorithm several times and outputting the best assignment, the probability 3/4 of outputting an assignment of the desired quality can be made exponentially close to 1.

For small k, this algorithm does much better than the simple m-approximation algorithm. The algorithm is simple and it just repeatedly tries random assignments. The proof of correctness of this algorithm uses a result from Fourier analysis that relates the L_4-norm of any function that has all its Fourier support on sets of size at most k to its L_2 norm.

We improve on the inapproximability results mentioned above by using a slightly stronger assumption.

Theorem 1.2. *Unless NP* $\subseteq$ *DTIME*$[2^{(\log m)^{O(1)}}]$, *for all* $k \geq 3$ *and* $\epsilon > 0$, *there is no algorithm that approximates Max-k-Lin-*2 *within* $2^{(\log m)^{1-\epsilon}}$ *and runs in time* $2^{(\log m)^{O(1)}}$.

The proof of Theorem 1.2 shows that the same result also applies to the special case of Max-k-Lin-2 where each equation contains exactly k variables. The proof of this result is similar to the proof of the result by Håstad that shows that it is NP-hard to approximate (in the old-fashioned sense, taking ratios of the number of satisfied equations) Max-E3-Lin-2 within $2 - \epsilon$ for any $\epsilon > 0$. It is based on the correspondence between approximation problems and probabilistically checkable proofs. One important difference between the two proofs is that we cannot use long codes in our result as they are too long. We make use of split codes defined recently by Khot [12].

1.1.2. The General Case. We now consider the general case in which there is no restriction on the number of variables in each equation. We start with the inapproximability results.

Theorem 1.3. *There exists a constant* $\gamma > 0$ *such that it is NP-hard to approximate Max-Lin-*2 *within* m^{γ}.

This proof uses an idea from derandomization and in particular it is based on the "walk on expanders" construction. If we allow randomization, we can get a stronger inapproximability result.

Theorem 1.4. *For any* $\epsilon > 0$, *unless NP* $\subseteq$ *RP*, *there is no randomized polynomial time algorithm that*, *with probability at least* $\frac{1}{2}$, *outputs an assignment for Max-Lin-*2 *with an approximation ratio at most* $m^{\frac{1}{2} - \epsilon}$.

This result uses a straightforward sampling technique. It gives evidence that no approximation algorithm is likely to achieve an approximation ratio much better than $m^{1/2}$ in general. For equations with few unknowns, Theorem 1.1 shows that one can almost do this well on such instances. Theorem 1.4 does not apply to the situation when each equation has only a constant number of variables. However, it does not use the full power of the general case since each equation in the proof of the theorem has only $O(\log n)$ variables.

The best upper bound we can show for the general case is rather poor.

Theorem 1.5. *For any* $c > 0$, *there is a randomized polynomial time algorithm that*, *with probability* 3/4, *outputs an assignment for Max-Lin-*2 *with approximation ratio at*

most $\frac{m}{c \log m}$ *. There is also, for any* $c > 0$, *a deterministic approximation algorithm that approximates Max-Lin-*2 *within* $\frac{m}{c}$.

The main idea is to use randomization and extend the greedy algorithm described in the beginning of Section 1.1.

1.2. Some Algorithmic Implications of Our Work

The measure we study for Max-Lin-2 has connections to some well-studied algorithmic questions.

1.2.1. Approximation Algorithms for Max-Lin-2. Consider the standard measure of approximation. It is known that for Max-k-Lin-2, the random assignment algorithm achieves approximation ratio 2. Håstad showed that it is NP-hard to get an algorithm with approximation ratio $2 - \epsilon$ for any constant $\epsilon > 0$. Thus the only question is exactly how close to 2 we can get.

Theorem 1.6. *For any* $c > 0$, *there is a randomized polynomial time algorithm for Max-Lin-*2 *that achieves approximation ratio* $2 - \sqrt{\frac{c \log m}{m}}$

We view this result as a step toward a better understanding of the correct value of ϵ as a function of m. Such results have been proved for problems like Max-Clique by Engebretsen and Holmerin [11] and Khot [13].

1.2.2. Learning Parity with Noise. Recently, the problem of solving parity with noise has been studied [10]. In this model, a random solvable system of linear equations in n variables is generated, and then the right-hand side of each equation is changed with probability p where $p < \frac{1}{2}$. The task is to reconstruct the solution to the original system. If the number of equations, m, is sufficiently large ($m > c_p n$ turns out to be enough) the solution is, with high probability, unique and can be found in exponential time. No algorithm, even for unbounded m, running faster than $2^{O(n/\log n)}$ is known and a major open problem is whether this can be improved.

In our performance measure, the best solution would get a value around $(1 - 2p)m$ while the second best solution would, with high probability, get a value that can be bounded by $O(\sqrt{nm})$. Thus, an efficient approximation algorithm for Max-Lin-2 with an approximation ratio of $m^{1/2-\epsilon}$ for $\epsilon > 0$ would imply that for $m \in \omega(n^{1/(2\epsilon)})$ the optimal solution could be recovered efficiently. However, Theorem 1.4 implies that such an efficient approximation algorithm for Max-Lin-2 is unlikely unless NP $\subseteq$ RP.

1.3. Organization of the Paper

In the next section, we describe some background from proof systems and some ideas from Fourier analysis that are used in this paper. We then give proofs of the theorems mentioned in the introduction in Section 3. We end with some conclusions in Section 4.

2. BACKGROUND

We start by recalling some definitions of proof systems used in this paper. Since the concepts are hopefully familiar to many readers of this paper, we keep the discussion brief and refer the reader to [7, 14] for more details.

2.1. Proof Systems

A central player in a proof system is a probabilistic Turing machine V called a *verifier*. The goal of the verifier is to decide whether to accept or reject an input string x. The verifier takes x as input and tosses random coins r. In addition, it has access to an oracle π.

Definition 2.1. *An* oracle *is a function* $\Sigma^* \mapsto \{0, 1\}^*$.

The verifier specifies a string $s \in \Sigma^*$ and the oracle returns a number of bits as the answer. For us, it is convenient to view the oracle as holding a *proof* and the verifier as specifying a location for which oracle returns the corresponding bit(s) of the proof as the answer. At the end of the computation, the verifier either accepts or rejects x. We say that the verifier *accepts* if it outputs 1 (written as $V^\pi(x, r) = 1$) and that it *rejects* if it outputs 0.

Definition 2.2. *Let c and s be real numbers such that $1 \geq c > s \geq 0$. We say that a language L has a* Probabilistically Checkable Proof (PCP) *with soundness s and completeness c if there exists a probabilistic polynomial time verifier V such that*

- *For $x \in L$, there exists an oracle π such that $\Pr_r[V^\pi(x, r) = 1] \geq c$.*
- *For $x \notin L$, for every oracle π $\Pr_r[V^\pi(x, r) = 1] \leq s$.*

We have the famous PCP-theorem that tells us that there are very efficient PCPs for any language in NP.

Theorem 2.3 [4]. *There is a universal integer q such that any language in NP has a PCP with soundness* 1/2 *and completeness* 1, *where V uses logarithmic number of random bits in the size of the input and makes at most q nonadaptive accesses to the oracle, each answered by a single bit.*

The number of bits accessed by the verifier can be reduced to 3, but this pushes the soundness towards 1, although it remains a constant below 1. Moreover, the 3 queries to the oracle are nonadaptive. When the verifier reads 3 bits nonadaptively, the acceptance condition on each fixed random string can be written as a CNF formula with at most 3 literals in each clause. By adding dummy variables it is not difficult to ensure that each clause is of length exactly 3, and we call such a formula an E3-CNF formula. This leads to the following variant of the PCP theorem that is convenient for us.

Theorem 2.4 [4]. *Let L be a language in NP and x be a string. There is a universal constant $\tau < 1$ for which the following holds: For every language L in NP, there is an algorithm that, given an input string x, runs for time polynomial in $|x|$ and produces a*

E3-CNF formula $\varphi_{x,L}$ such that if $x \in L$, then $\varphi_{x,L}$ is satisfiable while if $x \notin L$, no assignment to the variables of $\varphi_{x,L}$ satisfies more than a fraction τ of the clauses.

We next describe a two-prover one-round interactive proof system. The verifier in such a proof system has access to two oracles but has the limitation that it can only ask one question of each oracle and that both questions have to be produced before either of them is answered. Though we do not limit the answer size of the oracles, the verifier will not read more than polynomial number of bits since it runs in polynomial time. We call the two oracles P_1 and P_2 and the two questions q_1 and q_2. Since the oracles are only accessed through these questions we refer to the fact that V accepts as $V(x, r, P_1(q_1), P_2(q_2)) = 1$.

Definition 2.5. *Let c and s be real numbers such that $1 \geq c > s \geq 0$. We say that a language L has a* two-prover one-round proof system *with soundness s and completeness c if there exists a probabilistic polynomial time verifier V with two oracles that, on input x produces, without interacting with its oracles, two strings q_1 and q_2, such that*

- *For $x \in L$ there are two oracles P_1 and P_2 such that $\Pr_r[V(x, r, P_1(q_1), P_2(q_2)) = 1] \geq c$.*
- *For $x \notin L$, for any two oracles P_1 and P_2, $\Pr_r[V(x, r, P_1(q_1), P_2(q_2)) = 1] \leq s$.*

Note that the questions q_1 and q_2 are in both cases the only questions V asks the oracles. $P_1(q_1)$ depends on x, but may not depend on q_2 and similarly $P_2(q_2)$ is independent of q_1.

On many occasions, it is convenient to think of P_1 and P_2 as two actual dynamic provers rather than oracles or proofs. They are infinitely powerful and are cooperating. They can make any agreement before the interaction with V starts but then they cannot communicate during the run of the protocol. Thus it makes sense to ask P_1 and P_2 for the same information in different contexts.

Provers are, in general, allowed to be both history-dependent and randomized. Since we only consider one-round protocols, there is no history and hence the question whether the provers are history-dependent plays no role. As with randomization, it can[1] be seen that, for any x, the provers P_1 and P_2 maximizing $\Pr_r[V(x, r, P_1(q_1), P_2(q_2)) = 1]$ can be made deterministic without decreasing the acceptance probability. When proving the existence of good strategies for the provers we will, however, allow ourselves to design probabilistic strategies, which then, in principle, can be converted to deterministic strategies.

Improving the soundness of a two-prover proof system can be done by parallel repetition where V repeats his random choices to choose u independent pairs of questions $(q_1^{(i)}, q_2^{(i)})_{i=1}^u$ and sends $(q_1^{(i)})_{i=1}^u$ to P_1 and $(q_2^{(i)})_{i=1}^u$ to P_2, all at once. V then receives u answers from each prover and accepts if it would have accepted in all u protocols given each individual answer. One could naively hope that the soundness of this parallel

[1]Fix an optimal strategy, which might be randomized, of P_1. Now, for each q_2, P_2 can consider all possible coin tosses r of V producing q_2, compute q_1 and then, since the strategy of P_1 is fixed, exactly calculate the probability that V would accept for each possible answer. P_2 then answers with the lexicographically first string achieving the maximum. This gives an optimal deterministic strategy for P_2. We can then proceed to make P_1 deterministic by the symmetric approach.

protocol would be s^u; but this is not true in general, and we have to rely on the parallel repetition theorem of Raz [16].

Theorem 2.6 [16]. *For all integers d and $s < 1$, there exists $c_{d,s} < 1$ such that given a two-prover one-round proof system with soundness s and answer sizes bounded by d, then, for all integers u, the soundness of u protocols run in parallel is bounded by $c_{d,s}^u$.*

Since we do not limit the answer size of the provers they can of course misbehave by sending long answers which always cause V to reject. Thus, by answer size, we mean the maximal answer size in any interaction where V accepts.

2.2. Important Tool; Fourier Transforms

We study functions $f: \{-1, 1\}^n \to \Re$ mapping $\{-1, 1\}^n$ into the set of real numbers. A key tool for us is the discrete Fourier transform. For any $\alpha \subseteq [n]$, we have the corresponding basis function

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i.$$

A general function can be expanded as

$$f(x) = \sum_\alpha \hat{f}_\alpha \chi_\alpha(x),$$

where $\hat{f}_\alpha$ are the Fourier coefficients defined by

$$\hat{f}_\alpha = 2^{-n} \sum_x f(x) \chi_\alpha(x).$$

Parseval's equality tells us that

$$\sum_\alpha \hat{f}_\alpha^2 = 2^{-n} \sum_x f(x)^2$$

and for a Boolean function, whose range is $\{1, -1\}$, this sum is 1.

2.3. Max-Lin-2

Fourier transforms have been, and are also in this paper, important for the reason that they are useful in the analysis of certain probabilistically checkable proofs. In the present paper, Fourier transforms play an even more central role because the very problem we study is stated very naturally in terms of the Fourier transform. In order to see this, let us define the linear system formally. Remember that we are in $\{-1, 1\}$-notation and hence exclusive-or is in fact multiplication.

Definition 2.7. *A linear system L with m equations in n variables is given by subsets of $[n]$ $(\alpha_i)_{i=1}^m$, bits $(b_i)_{i=1}^m$ and positive weights $(w_i)_{i=1}^m$. The ith equation is*

$$\prod_{j\in\alpha_i} x_j = (-1)^{b_i}.$$

For an assignment $x^0 = (x_j^0)_{j=1}^n$ to the variables, let $P_L(x^0)$ be the total weight of all satisfied equations, and let $N_L(x^0)$ be the total weight of all falsified equations. The objective value $W(L, x^0)$ is defined as $P_L(x^0) - N_L(x^0)$. In the problem Max-Lin-2 we are given L, and we need to find an x^0 that maximizes $W(L, x^0)$.

The measure we used in the introduction subtracted instead of $N_L(x^0)$ half the total weight but it is easy to check that the new measure is exactly twice the measure used in the introduction. Since we are interested in ratios of objective values this change of scale is immaterial. We have the following lemma.

Lemma 2.8. $W(L, x^0) = \sum_{i=1}^m (-1)^{b_i}\chi_{\alpha_i}(x^0)w_i$.

Proof. Note that, by definition,

$$(-1)^{b_i}\prod_{j\in\alpha_i} x_j^0 = (-1)^{b_i}\chi_{\alpha_i}(x^0)$$

and this is 1 if the ith equation is satisfied and -1 otherwise. The lemma follows. ∎

In other words, we are given the Fourier coefficients of a function, and we want to find the maximum of the function.

The only reason we do not have a general function mapping $\{-1, 1\}^n$ to the real numbers is that we require the number of (nonzero) Fourier coefficients to be bounded by a parameter m which is usually much smaller than the maximal value 2^n.

A particular case that we are interested in is the case where each equation depends on at most k variables.

Definition 2.9. *An instance of Max-Lin-2 is said to belong to Max-k-Lin-2 if $|\alpha_i| \le k$ for $1 \le i \le m$.*

It turns out that functions that have the support of the Fourier transform on small sets have special properties and to capture one of these let us study the L_p-norm denoted by $\|f\|_p$ and defined by

$$\|f\|_p = (2^{-n}\sum |f(x)|^p)^{1/p}.$$

It is not difficult to see that $\|f\|_p$ is an increasing function in p. The key result we use is that for functions whose Fourier transform is supported on small sets, we have a reverse inequality proved in several contexts, but the following is Theorem 6 from [9]. A similar result can also be obtained using Theorem 5 from [6].

Theorem 2.10 (Bonami [9]). *For any integer k and $p \geq 2$, let f be a function which has its Fourier support on sets of size at most k; then $\|f\|_p \leq (p-1)^{k/2}\|f\|_2$.*

2.4. The Long Code and the Split Code

In PCP theory, many results use the long code defined by Bellare, Goldreich, and Sudan [7]. Let $\mathcal{G}_t$ be the set of all functions mapping $\{-1, 1\}^t$ to $\{-1, 1\}$.

Definition 2.11. *Let $x \in \{-1, 1\}^t$. Its long code is a vector of length 2^{2^t} with its entries corresponding to elements in $\mathcal{G}_t$ and the value corresponding to $f \in \mathcal{G}_t$ is $f(x)$.*

The set $\{-1, 1\}^t$ has no special significance and the long code can be defined for any range M of x with the set of all Boolean functions on M, denoted $\mathcal{G}_M$, taking the place of $\mathcal{G}_t$.

It is obvious from the definition that the long code is indeed very long. The previous results have made good use of the huge amount of information contained in these codes. In some situations, like the one in this paper, a much shorter but closely related code turns out to be a good alternative and these are the split codes defined by Khot [12]. Split codes are defined for product spaces.

Definition 2.12. *Let*

$$M = M_1 \times M_2 \times \cdots \times M_t.$$

The split code of an element $x = (x_1, x_2, \ldots x_t) \in M$ is indexed by all tuples $g = (g_1, g_2, \ldots, g_t)$ where $g_i \in \mathcal{G}_{M_i}$ and the corresponding value is

$$\prod_{i=1}^{t} g_i(x_i).$$

Thus, split codes belong to the vector space of all real-valued functions on $\mathcal{G} = \mathcal{G}_{M_1} \times \cdots \times \mathcal{G}_{M_t}$. We can define an inner product on this space by

$$\langle B_1, B_2 \rangle = \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} B_1(g) B_2(g).$$

Definition 2.13. *Let $\beta = (\beta_1, \ldots, \beta_t)$, $\beta_i \subseteq M_i$. We define a character χ_β as*

$$\chi_\beta(g) = \prod_{i=1}^{t} \prod_{x \in \beta_i} g_i(x).$$

Under the inner product defined above, the set of all characters χ_β form an orthonormal basis for the product space. For an arbitrary function B, we can write

$$B(g) = \sum_{\beta} \hat{B}_{\beta}\chi_{\beta}(g), \tag{2}$$

where

$$\hat{B}_{\beta} = 2^{-\sum_{i=1}^{t}|M_i|} \sum_{g} B(g)\chi_{\beta}(g). \tag{3}$$

Folding was designed in [7] as a mechanism forcing a table that is supposed to be a long code to at least respect negation. In particular this mechanism prevents a table from being biased. The mechanism nicely extends to split codes.

Definition 2.14. *A split code B is folded over true if*

$$B(g_1, \ldots, g_t) = (-1)^{s_1+\cdots+s_t}B((-1)^{s_1}g_1, \ldots, (-1)^{s_t}g_t) \tag{4}$$

for all $g_1, \ldots, g_t$ and for all $s_1, \ldots, s_t \in \{0, 1\}$.

Folding is implemented by choosing, for each set of 2^t inputs,

$$B((-1)^{s_1}g_1, \ldots, (-1)^{s_t}g_t), \qquad s_i \in \{0, 1\}^t,$$

only one representative to be stored in the table. Suppose for concreteness that $B(g_1, \ldots, g_t)$ is chosen. Whenever the value of any other input in the set is needed, $B(g_1, \ldots, g_t)$ is read and then possibly negated to satisfy (4).

The following lemma states a useful property of folded tables.

Lemma 2.15. *If a table B is folded over true, then $\hat{B}_{\beta} = 0$ for all $\beta = (\beta_1, \ldots, \beta_t)$ such that $|\beta_i|$ is even for some i.*

Proof. If $|\beta_i|$ is even then any term in (3) is canceled by the term obtained by changing g_i to $-g_i$ while keeping the other g_j unchanged. ∎

Split codes are crucial to our PCP construction as they offer a less lengthy alternative to long codes. The split code on M is only of length $2^{|M_1|+|M_2|+\cdots+|M_t|}$ while a long code on M would be of length $2^{|M_1|\times|M_2|\times\cdots\times|M_t|}$.

2.5. Notation

All logarithms in this paper are to the base 2. We use $|\cdot|$ both to denote the cardinality of any set and the absolute value of real numbers. The meaning would be clear from the context.

3. PROOFS OF THEOREMS

3.1. Proof of Theorem 1.1

We are given an instance L of Max-k-Lin-2, and for brevity let $f(x) = W(L, x)$. Our task is to find a value x^0 such that $f(x^0) \geq c^{-k} m^{-1/2} \max_x f(x)$ with high probability. This is done by repeatedly picking uniformly random values for x^0, and this is formalized in Algorithm 1.

Algorithm 1: Let $N = 16 \cdot 3^k m$. Pick N assignments $x^{(i)}$, $1 \leq i \leq N$, independently and uniformly at random. Output the assignment $x^{(\ell)}$ with the property $f(x^{(\ell)}) = \max_j f(x^{(j)})$ for $1 \leq j \leq N$.

To show that Algorithm 1 achieves an approximation ratio as claimed, define X to be the random variable $f(x)$ when x is chosen randomly. We claim that X has the following properties:

1. $E(X) = 0$.
2. $E(X^2) = \sum_{i=1}^m w_i^2 =^{def} \sigma^2$.
3. $|X| \leq \sqrt{m}\sigma$.
4. $E(X^4) \leq 3^{2k}(E(X^2))^2 = 3^{2k}\sigma^4$.

The first two properties are straightforward. The third property is shown by using Cauchy-Schwartz inequality as follows:

$$|X| \leq \sum_{i=1}^{m} w_i \leq \left(\sum_{i=1}^{m} 1 \right)^{1/2} \left(\sum_{i=1}^{m} w_i^2 \right)^{1/2} = m^{1/2}\sigma.$$

The fourth property follows from Theorem 2.10 with $p = 4$.

We now have the following lemma.

Lemma 3.1. *Consider any random variable X which takes values between $-m^{1/2}\sigma$ and $m^{1/2}\sigma$ and has the following properties*: (1) $E(X) = 0$, (2) $E(X^2) = \sigma^2$, *and* (3) $E(X^4) \leq 3^{2k}\sigma^4$. *Then*,

$$\Pr\left[X \geq \frac{\sigma}{8 \cdot 3^k} \right] \geq \frac{1}{8 \cdot 3^k m}.$$

Proof. Suppose not. Let us assume that

$$\Pr\left[X \geq \frac{\sigma}{8 \cdot 3^k} \right] < \frac{1}{8 \cdot 3^k m}.$$

Let $X_p = \max(0, X)$ and $X_n = -\min(0, X)$. From Hölder's inequality, we have that

$$E(X_n^2) \leq E(X_n^4)^{1/3} E(X_n)^{2/3}. \tag{5}$$

We now make the following claims using the assumptions of the lemma.

1. $E(X_n^4)^{1/3} \le 3^{2k/3}\sigma^{4/3}$.
2. $E(X_n)^{2/3} \le 2^{-1}3^{-2k/3}\sigma^{2/3}$.
3. $E(X_n^2) > \sigma^2/2$.

These claims clearly contradict (5), and thus we only have to establish these claims. The first is simply an assumption in the lemma. The second claim is established as follows:

$$E(X_n) = E(X_p) \le \sigma 8^{-1}3^{-k} + m^{1/2}\sigma \frac{1}{8 \cdot 3^k m} \le 4^{-1}3^{-k}\sigma,$$

and the third follows from $E(X^2) = E(X_p^2) + E(X_n^2)$ and

$$E(X_p^2) \le \sigma^2 8^{-1}3^{-k} + m\sigma^2 \frac{1}{8 \cdot 3^k m} < \sigma^2/2.$$

∎

From Lemma 3.1, it follows that when an assignment x' is picked uniformly at random, then with probability at least $\frac{1}{8 \cdot 3^k m}$, the approximation ratio is bounded from above by

$$\frac{\max_x f(x)}{f(x')} \le \frac{\sum_{i=1}^m w_i}{\sigma/8 \cdot 3^k} \le \frac{\sqrt{m}\sigma}{\sigma/8 \cdot 3^k} \le 8 \cdot 3^k \sqrt{m}$$

Algorithm 1 increases the probability of success to 3/4 by picking assignments independently and uniformly at random and outputting the best. Hence, Theorem 1.1 follows.

We believe that Theorem 1.1 is not too far from the true behavior of Algorithm 1, and an example where it does poorly can be constructed using the discussion in Section 1.2.2. To be more precise, start with a target solution x^0 and m random parities each of size k. Probabilistically choose the right-hand sides so that x^0 satisfies each equation with probability $1 - p$. Clearly, $f(x^{(0)}) \ge \Omega(m)$ with high probability.

Now look at any random assignment $x^{(i)}$ generated by Algorithm 1. With high probability $x^{(i)}$ agrees with x^0 in at most $(n + O(\sqrt{n \log n})/2$ coordinates. This implies that when generating an equation the probability that $x^{(i)}$ and x^0 want the same right hand side is at most $(1 + O((\log n)/n))^{k/2})/2$. Now, for $m \le n^s$, $s < k$ it is not difficult to see that with high probability we have $\max_i f(x^{(i)}) \le O(\sqrt{m \log n})$ while $f(x^{(0)}) \ge \Omega(m)$.

3.2. Proof of Theorem 1.2

A well-known approach to prove an inapproximability result for an optimization problem is to start with a language L in NP and design a PCP for L in which the acceptance criteria of the verifier closely mimics the optimization problem and this is also the approach used here. The construction of this PCP goes through three stages: In the first stage, using Theorem 2.4, checking membership in L is converted into solving a version of a GAP-SAT problem. Next, a two-prover protocol is designed for GAP-SAT. Finally, a

PCP is obtained starting from the two-prover protocol. We describe each stage of the reduction in detail below.

Start with a language L in NP and a string x. Apply the efficient algorithm stated in Theorem 2.4 to get a 3-CNF formula $\varphi_{x,L}$ which is satisfiable if $x \in L$ but has the property that no assignment satisfies more than a fraction τ of the clauses if $x \notin L$. Note that this reduction converts checking membership in L to solving a GAP-SAT problem of deciding if a formula is satisfiable or far from being satisfiable. Now consider the following two-prover protocol.

The 2-prover protocol: The protocol uses parameters t and u that will be fixed later.

Step 1: The verifier chooses tu random clauses from $\varphi_{x,L}$ and tu variables at random, one from each of the tu clauses chosen.

Step 2: The verifier asks prover P_1 for the assignment to each clause chosen. It asks prover P_2 for the assignment to each variable chosen.

Step 3: The verifier accepts if all the clauses are satisfied and the two assignments are consistent.

The two-prover protocol has completeness one. If $\varphi_{x,L}$ is satisfiable, P_1 and P_2 can agree on one satisfying assignment and answer accordingly. These answers are always consistent and satisfy any clauses picked.

The basic protocol with $t = u = 1$ has soundness $(2 + \tau)/3$ and hence, by Theorem 2.6, the soundness of this two-prover protocol is upper bounded by c_1^{tu} for some $c_1 < 1$. Thus, we have shown that L has a two-prover protocol with completeness 1 and soundness at most c_1^{tu}. We state these properties for future reference.

Lemma 3.2. *If $\varphi_{x,L}$ is constructed according to Theorem* 2.4, *then, if $x \in L$, there are strategies for P_1 and P_2 in the two-prover protocol to make the verifier always accept. If $x \notin L$, then, for some absolute constant $c_1 < 1$, no strategies for P_1 and P_2 can make the verifier accept with probability larger than c_1^{tu}.*

The next step of the reduction is to convert the two-prover protocol for L described above into a PCP.

From 2-prover protocol to PCP: We first describe the oracle (or the proof) that is expected by the verifier in the PCP. In other words, we describe the proof used to convince the verifier that a satisfiable $\varphi_{L,x}$ is indeed satisfiable.

The verifier expects two proofs A and B. Both of them correspond to split codes folded over true as described in Section 2.4. B is indexed by every possible question asked by the verifier to P_1 in the two-prover protocol. Each such question is grouped into t groups of u clauses and the split code of dimension t for the answer to this question is stored in B. Similarly, A is indexed by every possible question asked by the verifier to P_2 in the two-prover protocol. Each such question is grouped into t groups of u variables and a split code of dimension t for the answer to this question is stored in A.

For the answers of prover P_1, the set M_i, in the definition of the split code, is the set

of satisfying assignments of the clauses of group i. It is of cardinality at most 7^u. For the answers of prover P_2, we denote by N_i, an assignment to the chosen variables in group i. It is of cardinality at most 2^u. We have

$$M = M_1 \times M_2 \times \cdots \times M_t$$

and

$$N = N_1 \times N_2 \times \cdots \times N_t.$$

Using the projection operator from M_i to N_i which maps an assignment to the subassignment, we obtain a natural compound projection operator $\pi : M \to N$.

The PCP construction: We now describe the PCP that is used to derive the inapproximability result for Max-k-Lin-2. The verifier of this PCP uses a parameter $\epsilon < \frac{1}{2}$.

Step 1: The verifier picks t groups each containing u clauses chosen at random resulting in ut clauses $(C_1, C_2, \ldots, C_{ut})$. For each clause C_i, it chooses a variable $x_i \in C_i$ at random which are then similarly divided into t groups. This defines M and N. Let B be the supposed split code on M and A the supposed split code on N, both folded over true.

Step 2: The verifier then chooses random functions $f_i : N_i \to \{+1, -1\}$ and $g_i : M_i \to \{+1, -1\}$ for $1 \leq i \leq t$.

Step 3: The verifier also chooses $\mu_i : M_i \to \{+1, -1\}$ for $1 \leq i \leq t$ by setting for every $x \in M_i$,

$$\mu_i(x) = \begin{cases} 1 & \text{with probability } 1 - \epsilon, \\ -1 & \text{with probability } \epsilon. \end{cases}$$

Step 4: Define $h_i : M_i \to \{+1, -1\}$ for $1 \leq i \leq t$ by $h_i(x) = f_i(\pi(x))g_i(x)\mu_i(x)$.

Step 5: Let $g = (g_1, \ldots, g_t), f = (f_1, \ldots, f_t)$ and $h = (h_1, \ldots, h_t)$. The verifier reads $A(f)$, $B(g)$, and $B(h)$. The verifier accepts if and only if $A(f)B(g)B(h) = 1$.

We now do the completeness and the soundness analysis of our PCP construction. Computing the completeness is straightforward as shown below. To show that soundness of the PCP is small, our strategy is to prove that if, on the contrary, it is high, then we can extract strategies for the two provers in the two-prover protocol to convince the verifier in that protocol to accept with high probability. Using the very good soundness of the two-prover protocol stated above, we can conclude that the formula is satisfiable. This helps us obtain an upper bound on the soundness of the PCP. We now present the details.

Lemma 3.3. *The completeness of the PCP is at least* $\frac{1 + (1 - 2\epsilon)^t}{2}$.

Proof. If $\varphi_{x,L}$ is satisfiable, fix a satisfying assignment giving strategies for P_1 and P_2. As indicated above the written proof for the PCP is the split-codes of the answers by P_1 and P_2. We have

$$A(f)B(g)B(h) = 1 \Leftrightarrow \prod_{i=1}^{t} \mu_i(x_i) = 1,$$

where x_i is the ith component of the answer by P_1. It is not difficult to see that

$$\Pr\left[\prod_{i=1}^{t} \mu_i(x_i) = 1\right] = \frac{1 + (1 - 2\epsilon)^t}{2},$$

and the lemma follows. ∎

The key lemma for establishing the soundness is given below:

Lemma 3.4. *Let ϵ be the parameter of the PCP and let δ be such that the probability that the verifier of the PCP accepts is $(1 + \delta^t)/2$. Then there is a strategy for P_1 and P_2 in the two-prover protocol that makes the verifier of that protocol accept with probability at least $(4\epsilon\delta^2)^t$.*

Remark 3.5. As we know, by Lemma 3.2, the soundness of the two-prover protocol is bounded by c_1^{tu} and hence we can, by a suitable choice of u, get soundness $(1 + \delta^t)/2$ for any desired constant δ for the PCP.

Proof. Fix a choice of N and M. From the assumption in the lemma,

$$\mathop{\mathrm{E}}_{N,M,f,g,\mu}\left[\frac{1 + A(f)B(g)B(h)}{2}\right] = \frac{1 + \delta^t}{2}.$$

This implies that

$$\mathop{\mathrm{E}}_{N,M,f,g,\mu}[A(f)B(g)B(h)] = \delta^t.$$

We would now like to use this fact to extract a successful two-prover strategy. To this end, we evaluate the expression

$$\mathop{\mathrm{E}}_{f,g,\mu}[A(f)B(g)B(h)].$$

We need the following definition.

Definition 3.6. *For $\gamma = (\gamma_1, \ldots, \gamma_t)$, $\gamma_i \subseteq M_i$, define*

$$\pi_2(\gamma) = (\pi_2(\gamma_1), \ldots, \pi_2(\gamma_t))$$

where $\pi_2(\gamma_i)$ is the set of points in N_i with odd number of preimage points in γ_i under projection π.

Replacing each function by its Fourier expansion, we get

$$\mathop{\mathrm{E}}_{f,g,\mu}[A(f)B(g)B(h)] = \mathop{\mathrm{E}}_{f,g,\mu}\left[\sum_{\alpha,\beta^1\beta^2} \hat{A}_\alpha \chi_\alpha(f)\hat{B}_{\beta^1}\chi_{\beta^1}(g)\hat{B}_{\beta^2}\chi_{\beta^2}(h)\right]$$
$$= \sum_{\alpha,\beta^1\beta^2} \hat{A}_\alpha \hat{B}_{\beta^1}\hat{B}_{\beta^2} \mathop{\mathrm{E}}_{f,g,\mu}[\chi_\alpha(f)\chi_{\beta^1}(g)\chi_{\beta^2}(fg\mu)].$$

Since $\chi_{\beta^2}(fg\mu) = \chi_{\beta^2}(f)\chi_{\beta^2}(g)\chi_{\beta^2}(\mu)$, it is not difficult to see that the inner expectation equals 0 unless $\beta^1 = \beta^2 = \beta$ and $\alpha = \pi_2(\beta)$. Finally

$$\mathrm{E}[\chi_\beta(\mu)] = (1 - 2\epsilon)^{|\beta_1|+\cdots+|\beta_t|},$$

giving the total result

$$\sum_\beta \hat{A}_{\pi_2(\beta)}\hat{B}_\beta^2(1 - 2\epsilon)^{|\beta_1|+\cdots+|\beta_t|}. \tag{6}$$

Since B is folded over true, each nonzero term in this sum has $|\beta_i|$, and hence $|\pi_2(\beta_i)|$, odd for every i. In particular each β_i is nonempty. By Cauchy-Schwartz inequality, we have

$$\left(\sum_\beta \hat{A}_{\pi_2(\beta)}\hat{B}_\beta^2(1 - 2\epsilon)^{|\beta_1|+\cdots+|\beta_t|}\right)^2 \le \sum_\beta \hat{B}_\beta^2 \sum_\beta \hat{A}^2_{\pi_2(\beta)}\hat{B}_\beta^2(1 - 2\epsilon)^{2(|\beta_1|+\cdots+|\beta_t|)}$$

and since $\Sigma_\beta \hat{B}_\beta^2 = 1$, we can conclude that

$$\mathop{\mathrm{E}}_{N,M}\left[\sum_\beta \hat{A}^2_{\pi_2(\beta)}\hat{B}_\beta^2(1 - 2\epsilon)^{2(|\beta_1|+\cdots+|\beta_t|)}\right] \ge \mathop{\mathrm{E}}_{N,M}\left[\left(\sum_\beta \hat{A}_{\pi_2(\beta)}\hat{B}_\beta^2(1 - 2\epsilon)^{|\beta_1|+\cdots+|\beta_t|}\right)^2\right]$$
$$\ge \mathop{\mathrm{E}}_{N,M}\left[\left(\sum_\beta \hat{A}_{\pi_2(\beta)}\hat{B}_\beta^2(1 - 2\epsilon)^{|\beta_1|+\cdots+|\beta_t|}\right)\right]^2$$
$$\ge \delta^{2t}. \tag{7}$$

Let us now define a strategy for the two provers in the two-prover protocol as follows:

- P_1, upon receiving a set of clauses, finds the corresponding table B and selects a random β with probability $\hat{B}_\beta^2$. It returns $y_i \in \beta_i$, $1 \le i \le t$, chosen at random as the answer.
- P_2, upon receiving a set of variables, finds the corresponding table A and selects a random α with probability $\hat{A}_\alpha^2$. It returns $x_i \in \alpha_i$, $1 \le i \le t$, chosen at random.

The probability of the provers convincing the verifier is at least

$$\mathop{\mathrm{E}}_{N,M}\left[\sum_\beta \hat{A}^2_{\pi_2(\beta)}\hat{B}_\beta^2 \prod_{i=1}^t \frac{1}{|\beta_i|}\right].$$

For any $x > 0$, we have $x^{-1} > e^{-x}$. Applying this to $x = (4\epsilon|\beta_i|)$, we have

$$\frac{1}{|\beta_i|} \geq 4\epsilon e^{-4\epsilon|\beta_i|} \geq 4\epsilon(1 - 2\epsilon)^{2|\beta_i|}$$

and by now using the lower bound from (7), the lemma follows. ∎

Thus, we have shown that L has a PCP with completeness at least $(1 + (1 - 2\epsilon)^t)/2$ and soundness at most $(1 + \delta^t)/2$. To complete the proof of Theorem 1.2, we first broadly describe the relationship between PCPs and proving inapproximability results and then focus on the implications of the PCP designed above.

Let us define the following proof optimization problem based on the constructed PCP. For every possible location l of the proof we have a Boolean variable y_l. The goal is to find an assignment to the variables y_l such that the corresponding proof makes the verifier of the PCP accept x with the highest possible probability. If the PCP has completeness c and soundness s, then the value of the proof optimization problem, when x belongs to L, is at least c while it is at most s when x does not belong to L. This implies that if we can approximate the optimal value of the proof optimization problem within a factor better than c/s, then we can solve membership in L which is NP-hard in general.

Now consider the PCP that we have constructed. By doing some internal calculation, the verifier V finally reads three bits of the proof nonadaptively and checks that the product of these three bits has a given value. One might be tempted to think that this value is always 1 (in the $\{+1, -1\}$ notation) but it is not so due to the mechanism of folding that has the affect that sometimes the bit read in the proof should be negated before it is used.

Suppose that with probability p, V reads positions l_1, l_2, and l_3 and checks that the product of these three bits is -1. Let us write this as an equation

$$y_{l_1}y_{l_2}y_{l_3} = -1$$

and give it weight p. Continuing this way with all possible choices of positions for V, we get a system of weighted linear equations and the total weight of satisfied equations is exactly the probability that the verifier accepts a given proof. Thus, the proof optimization problem is exactly the problem of maximizing the total weight of satisfied equation and this is the problem for which we are trying to prove an inapproximability result.

Note that it is crucial that the verifier does not use too many random coins as the size of the resulting system of linear equations depends on the number of coin tosses. In fact, the size of the produced instance of equations is 2^r where r is the number of coins used. Hence, we conclude that it is hard to approximate Max-3Lin-2, in our performance measure, better than $(c - 1/2)/(s - 1/2)$ on instances of size 2^r. We will now fix the various parameters ϵ, δ, u, and t so as to get a strong separation.

Fix $\epsilon = \delta = 1/4$. Fix u to be a large enough constant so that $c_1^u < 4\delta^3$ where c_1 is the constant specified in Theorem 2.6. If m is the number of clauses in the 3-CNF formula given as input to the PCP, the number of equations produced by the transformation described above is $N = m^{O(t)}$ with total weight 1 (as we are dealing with probabilities). As seen by the analysis above

$$c - 1/2 \geq (1 - 2\epsilon)^t$$

and

$$s - 1/2 \leq \delta^t.$$

Therefore,

$$\frac{c - 1/2}{s - 1/2} \geq \frac{(1 - 2\epsilon)^t}{\delta^t} = 2^t.$$

Choose $l = \gamma^{-1}$ and set $t = (\log m)^l$. The number of equations is $N = m^{O(t)} = 2^{O((\log m)^{l+1})}$ *and the ratio* $\frac{c-1/2}{s-1/2} = 2^t = 2^{(\log m)^l}$. It follows that an approximation algorithm with a performance ratio of

$$2^{c(\log N)^{1-\gamma}}$$

for a suitable constant c would be sufficient to decide membership in L. As the size of the instance is only quasi-polynomial if an approximation algorithm with the given ratio existed and ran in quasipolynomial time this would imply $NP \subseteq DTIME[2^{(\log m)^{O(1)}}]$. This completes the proof of Theorem 1.2 for the case $k = 3$.

The extension to larger k is not difficult and can be done along the lines used for the similar extension in [14]. We omit the details.

3.3. Proof of Theorem 1.4

The proofs of Theorems 1.4 and 1.3 are quite similar. Since the proof of Theorem 1.4 is simpler, we present it first.

The basic idea is to start with any 3SAT formula ψ and produce a system of linear equations L_ψ such that the following holds: If ψ is satisfiable, then there is an assignment to the variables of L_ψ which achieves a high objective value. If ψ is not satisfiable, then every assignment to the variables of L_ψ only achieves a low objective value. We do this by taking t-wise products (sums in $\{0, 1\}$ notation) of sets of equations obtained from a 3SAT formula. We now give the details. In Håstad's paper [14], for any $\delta > 0$, a reduction is shown that takes 3SAT formulas with l clauses to Max-3-Lin-2 systems with $m = l^{O(1)}$ linear equations on n variables, $n < m$, such that

1. If ψ is satisfiable, then there is an assignment x^0 to the variables of the resulting system of linear equations L_ψ such that

$$W(L_\psi, x^0) \geq (1 - 2\delta)m.$$

2. If ψ is not satisfiable, then for every assignment x^0 to the variables to L_ψ,

$$W(L_\psi, x^0) \leq \delta m.$$

Let us introduce some notation.

Definition 3.7. *A system, L, with m equations is said to be δ-satisfiable if $W(L, x) \leq \delta m$ for every possible assignment x to its variables.*

Definition 3.8. *Let L denote a system of N linear equations. Its t-wise product, L^t, is a system of $(2N)^t$ linear equations defined as follows: Choose t equations from L. For every such choice of t equations, output 2^t linear equations obtained by taking products of every possible subset of these t equations.*

The following lemma says that taking t-wise products help in boosting the separation between the two cases: ψ is satisfiable or not.

Lemma 3.9. *The t-wise product of a δ-satisfiable system is a $(\frac{1+\delta}{2})^t$-satisfiable system.*

Proof. An assignment that satisfies all the t chosen equations satisfies all the 2^t constructed equations. An assignment that falsifies some equation satisfies exactly half of the equations constructed. It follows that a t-wise product is $(\frac{1+\delta}{2})^t$-satisfiable. ∎

The lemma implies that in the case when ψ is satisfiable then the system L_ψ^t is $(1 - \delta)^t$-satisfiable while when ψ is not satisfiable then L_ψ^t is at most $(\frac{1+\delta}{2})^t$-satisfiable.

We have one small detail to address. Taking the t-wise product produces equations of the form $1 = 1$ and $1 = -1$. These trivial equations have to be dropped as they are not allowed in our systems. This affects the number of equations as well as the obtained objective value. It turns out that if L has at least as many equations of the form $1 = 1$ as $1 = -1$ this process is only to our advantage and hence we like such systems.

Definition 3.10. *A system of linear equations L is said to be good if, in L, the number of equations of the form $1 = 1$ is greater than or equal to the number of equations of the form $1 = -1$.*

The t-wise products are in fact good.

Lemma 3.11. *A t-wise product, L^t, of any system of linear equations L is good.*

Proof. Fix a choice of t equations and consider the set S of 2^t equations produced by taking all possible subsets. Using the characteristic vector notation, view S as a vector space F^t over $F = \{0, 1\}$ under the usual operations. Let C be the subcollection of all subsets that result in an equation of the form $1 = 1$ and $1 = -1$. Then, C is a subspace of S since C is closed under symmetric difference. Choose a basis B for C. If every equation in B is of the form $1 = 1$, then every equation in C is of the form $1 = 1$. If there are equations in B of the form $1 = -1$, then there are equally many equations of the form $1 = 1$ and $1 = -1$ in C. ∎

Let us consider the effect on L_ψ^t of dropping these trivial equations. The objective value is, before dropping trivial equations, of the form $\delta' m'$ where m' is the number of equations and either $\delta' \geq (1 - \delta)^t$ or $\delta' \leq (\frac{1+\delta}{2})^t$. If the system is good the erasing decreases the objective value to $\delta' m' - C$ for some $C \geq 0$. We only need to bound the ratio between the objective values in the two cases and this is minimized when $C = 0$. It follows that

we can ignore these trivial equations as long as we make sure that the systems we study are good.

Though taking t-wise products helps us get an improved separation, the number of linear equations produced would be too many (for our choice of t) for a good inapproximability result. We decrease the number of equations using randomization.

For a 3SAT formula ψ, obtain L_ψ as described above. For a suitable choice of parameters s and t to be fixed later, choose $N = m^s$ random sets of equations from L_ψ each of size t. For every such choice R, construct 2^t equations as in the definition of L_ψ^t and choose one of these equations at random. Let the resulting system of linear equations be denoted by $L_{\psi,R}^t$. If $L_{\psi,R}^t$ is not good, repeat the construction. If after n such repetitions, a good system of equations is not produced, halt. Otherwise, erase all the trivial equations from $L_{\psi,R}^t$ and halt.

Since the probability that the system of linear equations produced in an iteration is not good is at most 1/2, the probability that the construction above fails to produce a good system of linear equations is upper bounded by $1/2^n$.

We would now like to compute the probability that a good system of linear equations has the separation property discussed above. We do it in two steps. First, we estimate this probability for a random system of N equations produced by an iteration of the algorithm above. Then, we show that this probability continues to be high conditioned on the fact that the system of linear equations is good.

Suppose ψ is satisfiable. Then, there is an assignment x^0 to the variables of L_ψ that satisfies $(1 - \delta)m$ of the equations in L_ψ. Over the various random choices,

$$\mathrm{E}[W(L_{\psi,R}^t, x^0)] \geq (1 - \delta)^t N.$$

We need the following result from Alon and Spencer [3].

Theorem 3.12 ([3], Theorem A.1.13). *Let X_i, $1 \leq i \leq k$, k arbitrary, be independent random variables such that*

$$X_i = \begin{cases} 1 & \text{with probability } \frac{1+p}{2}, \\ -1 & \text{otherwise}, \end{cases}$$

and let $X = \sum_{i=1}^k X_i$. Then,

$$\Pr[X < (1 - \epsilon)pk] < e^{-\frac{\epsilon^2}{8}pk}.$$

Let X_i denote the random variable that is 1 if the ith equation is satisfied by x^0 and -1 otherwise. Then, X_i is 1 with probability $(1 + p)/2$, where $p = (1 - \delta)^t$ and

$$\sum_{i=1}^{N} X_i = W(L_{\psi,R}^t, x^0).$$

Therefore, using Theorem 3.12, with $\epsilon = 1/2$, $p = (1 - \delta)^t$ and $k = N$, over the different choices of random equations,

$$\Pr[W(L^t_{\psi,R}, x^0) \geq (1 - \delta)^t N/2] \geq 1 - e^{-(1-\delta)^t N/32}. \tag{8}$$

Now suppose that ψ is not satisfiable. Then no assignment satisfies more than $(\frac{1+\delta}{2})m$ of the equations in L_ψ. We need the following result from Alon and Spencer [3].

Theorem 3.13 ([3], Theorem A.1.4). *Let X_i, $1 \leq i \leq k$, k arbitrary, be independent random variables such that*

$$X_i = \begin{cases} 1 & \text{with probability } \frac{1}{2} + \frac{p}{2}, \\ -1 & \text{otherwise}, \end{cases}$$

and let $X = \sum_{i=1}^k X_i$. Then, for any $a > 0$,

$$\Pr[X > kp + a] < e^{-\frac{a^2}{2k}}.$$

For a fixed assignment x^0, let X_i be the random variable that is 1 if the ith equation is satisfied by x^0 and -1 otherwise. Then, X_i is 1 with probability $(1 + p)/2$, where $p \leq ((1 + \delta)/2)^t$ and

$$\sum_{i=1}^{N} X_i = W(L^t_{\psi,R}, x^0).$$

Therefore, using Theorem 3.13, we get

$$\Pr\left[W(L^t_{\psi,R}, x^0) > N\left(\frac{1 + \delta}{2}\right)^t + 2\sqrt{nN} \right] < 2^{-2n},$$

and hence

$$\Pr\left[\max_{x^0} W(L^t_{\psi,R}, x^0) \leq N\left(\frac{1 + \delta}{2}\right)^t + 2\sqrt{nN} \right] \geq 1 - 2^{-n}. \tag{9}$$

Start with any $\epsilon < 1/4$. Fix $\delta = \epsilon/8$, $s = 2/\epsilon$, and $t = \log N$. Then, with probability at least $1 - 2^{1-n}$ over random choices in the construction above, both (8) and (9) hold simultaneously and hence the resulting system of linear equations has a separation property we are looking for. Since a random choice is good with probability at least 1/2, it follows that a good system of linear equations has the same separation property with probability at least $1 - 2^{2-n}$. Thus the probability that the system of linear equations produced by the construction above is good and has the separation property is at least $(1 - 2^{2-n})(1 - 2^{-n}) \geq 1 - 2^{3-n}$.

Also for the choice of δ, s and t above,

$$N\left(\frac{1+\delta}{2}\right)^t + 2\sqrt{nN} \leq 3\sqrt{nN}$$

$$(1-\delta)^t N/2 \geq N^{1-2\delta}.$$

Since 3SAT is NP-complete, any algorithm that gives an approximate value of the optimum within

$$\frac{N^{1-2\delta}}{3\sqrt{nN}} \geq N^{1/2-\epsilon}$$

can be used to solve an arbitrary problem in NP. This proves the theorem.

Since the random sampling step involves two-sided error, it seems like we need the assumption NP $\not\subseteq$ BPP. It is not difficult to see that using the self-reducibility of SAT, one can prove that if NP $\subseteq$ BPP, then in fact NP $\subseteq$ RP. Hence, the assumption NP $\not\subseteq$ RP is sufficient.

3.4. Proof of Theorem 1.3

We use a construction very similar to the one in Theorem 1.4. In the first stage, as in Theorem 1.4, we transform the 3SAT formula into a system of linear equations with the separation property. In the second stage, we replace random sampling by deterministic sampling obtained using walks on expanders. We now give the details of our construction.

Definition 3.14. *Let G be a d-regular graph on n vertices and let $d = \lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{n-1}$ be the eigenvalues of the adjacency matrix of G. G is said to be a Ramanujan graph if $\lambda_1, -\lambda_{n-1} \leq 2\sqrt{d-1}$.*

Explicit construction of such graphs were first shown by Lubotzky, Philips, and Sarnak [15].

Theorem 3.15 ([15]). *Pick a prime p congruent to 1 modulo 4. Then, for every prime q congruent to 1 modulo 4 different from p and such that the Legendre symbol $\left(\frac{p}{q}\right) = 1$, there is an explicitly constructible Ramanujan graph $G_{p,q}$ on $n = q(q^2-1)/2$ vertices which is $d = p+1$-regular.*

Theorem 3.16 ([3], p. 142). *Given any integer m and a prime p congruent to 1 modulo 4, using Theorem 3.15, we can construct explicitly a Ramanujan graph on $(1+o(1))m$ vertices that is $p+1$-regular.*

Thus, for m large enough, using the Theorem above, we can construct explicitly a 30-regular Ramanujan graph G on m' nodes for $m \leq m' \leq 2m$. Given a 3SAT formula ψ, we label vertex $i \leq m$ of G by equation i from L_ψ and every vertex $i > m$ by the trivial equation $1 = 1$ and call the new graph obtained as G_ψ.

For a 3SAT formula ψ, obtain L_ψ as described in Theorem 1.4. For a suitable choice of parameters d and t to be fixed later, choose $m'd^{t-1}$ sets of equations from L_ψ each of size t. These sets correspond to all paths of length t starting from every vertex in G_ψ. For every such choice, construct 2^t equations as in the definition of L^t_ψ and include all. Erase all the trivial equations.

For a 3SAT formula ψ, let us denote by W_{x^0}, the set of vertices in G_ψ such that the corresponding equations are satisfied by an assignment x^0. Note that all the vertices corresponding to the trivial equations are included in this set. Given the size of such a subset W_{x^0}, we need to calculate the fraction of all the sets of equations of size t obtained by walks on expander that are completely contained in W_{x^0}.

The following proposition (slightly reworded) from [1] gives the required bounds.

Proposition 3.17 (Proposition 2.4 [1]). *Let H be regular graph of degree d on m vertices and let $d = \lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{m-1}$ be the eigenvalues of the adjacency matrix of H. Let W be a set of w vertices in H and put $\mu = w/m$. Let $P = P(W, t)$ be the total number of walks on t vertices that stay in W. Assume (for the lower bound only) that t is odd and that $\mu + \lambda_{m-1}(1 - \mu)/d \geq 0$. Then the following inequalities hold*:

$$wd^{t-1}(\mu + \lambda_{m-1}(1 - \mu)/d)^{t-1} \leq P \leq wd^{t-1}(\mu + \lambda_1(1 - \mu)/d)^{t-1}.$$

By Lemma 3.11, the resulting system is always good and hence we can ignore the step of erasing trivial equations. Let $L^t_{\psi,G}$ denote the new system of linear equations in the above construction before the erasure step. It has $N = m'd^{t-1} \cdot 2^t$ equations. We know that if ψ is satisfiable, then there is an assignment x^0 such that

$$|W_{x^0}| \geq (1 - \delta)m + (m' - m) \geq (1 - \delta)m'.$$

Using the proposition above,

$$\begin{aligned} W(L^t_{\psi,G}, x^0) &\geq (1 - \delta)[(1 - \delta) + \lambda_{m'-1}\delta/d]^{t-1}N \\ &\geq (1 - \delta)[(1 - \delta) - 2\sqrt{29}\delta/30]^{t-1}N \\ &\geq (1 - \delta)\left[1 - \frac{3\delta}{2}\right]^{t-1}N. \end{aligned}$$

On the other hand, if ψ is not satisfiable, then for every assignment x^0,

$$|W_{x^0}| \leq \left(\frac{1 + \delta}{2}\right)m + (m' - m) \leq \left(\frac{3 + \delta}{4}\right)m'.$$

From the proposition above, for every assignment x^0,

$$W(L^t_{\psi,G}, x^0) \le \left(\frac{3+\delta}{4}\right)\left[\left(\frac{3+\delta}{4}\right) + \lambda_1\left(\frac{1-\delta}{4}\right)\Big/ d\right]^{t-1} N$$
$$\le \left(\frac{3+\delta}{4}\right)\left[\left(\frac{3+\delta}{4}\right) + 2\sqrt{29}\left(\frac{1-\delta}{120}\right)\right]^{t-1} N$$
$$\le \left(\frac{3+\delta}{4}\right)\left[\frac{6}{7} + \frac{\delta}{4}\right]^{t-1} N.$$

Since 3SAT is NP-complete, any algorithm that gives an approximate value of the optimum within

$$\frac{(1-\delta)[1-\frac{3\delta}{2}]^{t-1}}{(\frac{3+\delta}{4})[\frac{6}{7}+\frac{\delta}{4}]^{t-1}}$$

can be used to solve an arbitrary problem in NP. Choosing $\delta = 1/20$ and $t = \log m$, the ratio is m^c for some constant $c > 0$ while the number of equations N is at most $2^{6t}m = m^7$. Thus the ratio between the two cases is N^γ, for some constant $\gamma > 0$. This proves the theorem.

3.5. Proof of Theorem 1.5

Recall the greedy algorithm giving performance ratio m described in Section 1.1. We now analyze what happens if we keep picking equations that we commit ourselves to satisfying. By using randomness we are able to improve the approximation ratio from m to $m/\log m$.

We are given a system of m linear equations L in n variables as in Definition 2.7. While describing the algorithm, we assume that equations in L are updated dynamically, and in particular, although w_i are the weights defining the system initially, the weights are updated after each iteration as described in Section 1.1. If the left-hand side of the two equations are equal, they are merged to one equation with the appropriate weight. If the right-hand sides are the same, the new weight is the sum of the original weights and if they are different, the new weight is the difference of the original weights and the right-hand side is that of the equation with higher weight. However, for simplicity, we use w_i to denote the weight of equation i at any intermediate stage.

Algorithm 2, basic run

Step 1: Pick equation i with probability proportional to its weight w_i. Suppose that i_0 is picked and j is an element occurring in α_{i_0}. For all $i \neq i_0$ such that α_i contains j, replace α_i by $\alpha_i \Delta \alpha_{i_0}$, and b_i by $b_i \cdot b_{i_0}$. Erase equation i_0.

Step 2: If several equations result with the same left-hand side, combine these by adjusting the weights as discussed above. If nontrivial equations remain, go to Step 1 and repeat.

Step 3: Output an assignment x^0 to the variables that satisfies all the equations picked in Step 1.

As each equation picked in Step 1 contains a variable that is eliminated from all the remaining equations, it is straightforward to find an assignment x^0 as required in Step 3. It is not difficult to see that the objective value of the obtained solution, $W(L, x^0)$, is exactly the sum of the weights of all equations picked in Step 1. Now consider the following algorithm that repeats the basic run a polynomial number of times and outputs the assignment with the highest objective value.

Algorithm 2: Fix constants $c > 0$ and $d > c + 2$. Repeat the basic run described above m^d times. Let x^j denote the assignment chosen during the jth iteration. Output the assignment x^l with the property that $W(L, x^l) = \max_j W(L, x^j)$, $1 \leq j \leq m^d$.

Our goal is now to show that Algorithm 2 achieves approximation ratio $\frac{4m}{c \log m}$. To do so, we start by defining the notion of a successful basic run. Let us say that a basic run is t-successful if the algorithm makes at least t repetitions of Step 1 during this run and if it picks an equation that is satisfied by the optimal assignment in each of the first t iterations.

Lemma 3.18. *A run is t-successful with probability at least 2^{-t} unless the optimal assignment is found within less than t iterations.*

Proof. Conditioning on the fact that the equations chosen up to a given iteration are satisfied by the optimal assignment, it is easy to see that the probability that the next chosen equation is satisfied by the optimal assignment is at least 1/2. This follows from the fact that the optimal assignment satisfies more of the remaining equations than the equations it falsifies. Hence, a run is t-successful with probability at least 2^{-t} unless the optimal assignment is already obtained in less than t iterations. ∎

By Lemma 3.18, unless the optimal assignment is found before $c \log m$ iterations, a single run is $c \log m$-successful with probability at least m^{-c}. Let R denote the random variable that counts the number of $c \log m$-successful runs in Algorithm 2. Then, the expected number of $c \log m$-successful runs in Algorithm 2, $E[R]$, is at least m^{d-c}. Using Chernoff bounds [5],

$$\Pr\left[R < \frac{m^{d-c}}{2}\right] < e^{-m^{d-c}/8}. \tag{10}$$

For the rest of the analysis, let us focus on the case when Algorithm 2 has at least $m^{d-c}/2$ $c \log m$-successful runs. In this situation, we are interested in comparing the objective value of the optimal assignment W' with the objective value of the assignment output by Algorithm 2. We start by first looking at the objective value of the output of a basic run of Algorithm 2 conditioned on the run being $c \log m$-successful.

Let us analyze the expected value of the weight of the equation picked in iteration i. We have two cases, either we have already picked equations of total weight $W'/2$ or we have at least weight $W'/2$ remaining among equations that can be picked at this iteration. In the

first case we already have a 2-approximation, and this is much better than what we are trying to prove, so that we only have to consider the second case.

Suppose that $m' \leq m$ equations remain and the total weight of equations remaining that are satisfied by the optimal assignment is $W'' \geq W'/2$. In addition, let the equations satisfied by the optimal assignment have weights $\{w_{i_j}\}_{j \in J}$ for some index set J. Then the expected weight of the equation picked in this iteration is

$$\sum_{j \in J} \frac{w_{ij}^2}{W''} \geq \frac{1}{mW''} \left(\sum_{j \in J} w_{i_j} \right)^2 = \frac{W''}{m} \geq \frac{W'}{2m},$$

where the first inequality is a consequence of the Cauchy-Schwartz inequality. We conclude that the expectation of the objective value obtained by a $c \log m$-successful run is at least

$$\frac{cW' \log m}{2m}.$$

Now the maximum that can ever be obtained is W', and hence, by a standard averaging argument, the probability that we get value at least

$$\frac{cW' \log m}{2m} - \frac{W'}{m} > \frac{cW' \log m}{4m} \tag{11}$$

is at least $\frac{1}{m}$. Let us denote by S, the event that Algorithm 2 outputs an assignment that satisfies (11). Then,

$$\Pr[S \text{ occurs} | R \geq m^{d-c}/2] \geq 1 - \left(1 - \frac{1}{m}\right)^{m^{d-c}/2}.$$

Since we chose d such that $d > c + 2$, then

$$\begin{aligned} \Pr[S \text{ occurs}] &\geq \Pr[S \text{ occurs} | R \geq m^{d-c}/2] * \Pr[R \geq m^{d-c}/2] \\ &\geq (1 - e^{-m/2})(1 - e^{-m^2/8}), \end{aligned}$$

using (10). Thus, except with exponentially small probability, Algorithm 2 outputs an assignment that satisfies (11). Since c was any arbitrary constant, we have obtained the randomized algorithm as required in Theorem 1.5.

To get a deterministic algorithm, we try all possible choices for the c first iterations of Algorithm 2 and backtrack. This can be done in time $O(m^c)$. Many of these runs are c-successful, and it is not difficult to see that one of them gives an approximation ratio of at most m/c.

It is not unlikely that we can get a deterministic approximation algorithm with approximation ratio $\Theta(\frac{m}{\log m})$ by derandomizing the randomized algorithm. We invite the reader to try to prove this.

3.6. Proof of Theorem 1.6

Recall that, in this result, we are interested in designing a randomized approximation algorithm that does well for the standard measure of approximation. Therefore, the objective value of a solution is given by the total weight of all equations satisfied by it. The algorithm is very similar to Algorithm 2. In the rest of the proof, we shall use constants d and c'. We will explain later how to fix these two constants in terms of c.

Algorithm 3: Repeat the basic run of Algorithm 2 m^d times. Output the assignment that maximizes the total weight of satisfied equations.

We would like to obtain a bound on the approximation ratio achieved by Algorithm 3. Recall that W denotes the sum of the weights of all the equations in the system. If the total weight of the equations satisfied by the optimal assignment is at most $W(1 - \sqrt{\frac{c' \log m}{m}})$, then any assignment for which the total weight of the satisfied equations is at least $W/2$ gives an approximation ratio $2 - 2\sqrt{\frac{c' \log m}{m}}$ and the output of any basic run of Algorithm 2 can be used to get this ratio. For an appropriate choice of c' in terms of c, this gives the desired approximation ratio.

Hence, we now consider the case when the total weight of the equations satisfied by the optimal assignment is at least $W(1 - \sqrt{\frac{c' \log m}{m}})$.

Lemma 3.19. *A run is $\sqrt{c'm \log m}$-successful with probability at least $m^{-O(c')}$ unless it finds an assignment that satisfies equations of total weight $3W/4$.*

Proof. Let us analyze the probability that the next equation picked during a run is satisfied by the optimal assignment conditioned on the run being successful until this iteration and that the total weight of all equations picked so far in Step 1 is at most $W/2$.

Consider the remaining system of linear equations. Among the equations satisfied by the optimal assignment, at most $\sqrt{\frac{c' \log m}{m}}$ by weight could have been eliminated till now due to the updating done in Step 2 of the basic run. Therefore, in the remaining system of equations, we have equations of total weight at least $W(\frac{1}{2} - 2\sqrt{\frac{c' \log m}{m}})$ that are satisfied by the optimal assignment and equations of total weight at most $W\sqrt{\frac{c' \log m}{m}}$ that are falsified by the optimal assignment. The probability of picking an equation that is satisfied by the optimal assignment is hence at least

$$1 - 3\sqrt{\frac{c' \log m}{m}},$$

and we can conclude that we have an $m^{-O(c')}$ probability of having a $\sqrt{c'm \log m}$-successful run. ∎

Now the total weight of the equations picked in Step 1 of such a run can be analyzed in a way similar to what was done in Section 3.5. We do not repeat the calculations in this section. It can be shown that with probability at least $1/m$, the total weight of equations picked in Step 1 of a $\sqrt{c'm \log m}$-successful run is at least

$$\Omega\left(W\sqrt{\frac{c'\log m}{m}}\right).$$

Now using the fact that Algorithm 3 chooses the best assignment output by m^d repetitions of a basic run, it can be shown that, for an appropriate choice of d in terms of c', the objective value of the assignment output by Algorithm 3 is at least

$$W\left(\frac{1}{2}+\Omega\left(\sqrt{\frac{c'\log m}{m}}\right)\right)$$

with high probability. Finally we can choose c' in terms of c so that Theorem 1.6 follows. We omit the details.

3.7. Other Constraint Satisfaction Problems

Some of our results and methods extend in a straightforward way to other constraint satisfaction problems using reductions.

Our framework applies to those problems for which it is known that they cannot be approximated much better than by picking an assignment at random and a number of such problems can be found in [14], the most central problem among them being satisfiability. Let Max-Ek-Sat be the problem of satisfying the maximum number of clauses of a CNF-formula where each clause contains exactly k literals.

Since each clause is satisfied with probability $1-2^{-k}$, the relevant measure is

$$\max_L \frac{SAT[OPT(L)]-W(1-2^{-k})}{SAT[ALG(L)]-W(1-2^{-k})}. \tag{12}$$

Theorem 3.20. *Consider Max-Ek-Sat. There exists a fixed constant $c>1$ such that the following holds: for any $k\in O(\log n)$, there is a randomized polynomial time algorithm that, with probability at least 1/2, outputs an assignment that gives an approximation ratio at most $c^k\sqrt{m}$.*

Proof. Given a clause with k literals, we can write a system of 2^k-1 linear equations in the same variables with the property that if the clause is satisfied, 2^{k-1} of the linear equations are satisfied while if it is not, none of the equations are satisfied. For example, if the clause is $(x_1\vee x_2\vee x_3)$, then the seven equations are $x_1=-1$, $x_2=-1$, $x_3=-1$, $x_1x_2=-1$, $x_1x_3=-1$, $x_2x_3=-1$, and $x_1x_2x_3=-1$.

Thus if we start with m clauses of total weight W we get, replacing each clause by 2^k-1 linear constraints of equal weight, a system of total weight $(2^k-1)W=W'$ and $(2^k-1)m$ equations.

Now an assignment that satisfies clauses of weight $W(1-2^{-k})+X$ satisfies linear equations of weight

$$2^{k-1}(W(1-2^{-k})+X)=W'/2+2^{k-1}X,$$

and thus ratios of the advantage over a random assignment is preserved by this reduction.

Therefore, using the approximation algorithm for Max-k-Lin-2 described in Theorem 1.1, the theorem follows. ∎

The above theorem can be generalized to obtain a similar result on the new approximation ratio for any constraint satisfaction problem where each constraint only depends on at most k variables. Let us argue this informally.

Take any constraint on k variables $C(x) = C(x_1, x_2, \ldots, x_k)$ and think of this as function mapping $\{-1, 1\}^k$ to the real numbers. We can expand it using the Fourier transform

$$C(x) = \sum_{\alpha} \hat{C}_\alpha \chi_\alpha(x).$$

But this implies that the single constraint $C(x)$ can be replaced by constraints

$$\chi_\alpha(x) = \text{sign}(\hat{C}_\alpha)$$

each with weight $|\hat{C}_\alpha|$. As we noted in Lemma 2.8 this is just a system of linear equations and we can apply our approximation algorithm.

As a special case we note that in the above discussion on Max-Ek-Sat there was no need to assume that each clause was of length exactly k, and we could have treated the general case by introducing some more notation.

Let us now turn to inapproximability results and also in this case we can get results for Max-Ek-Sat.

Theorem 3.21. *Unless NP* $\subseteq$ *DTIME*$[2^{(\log m)^{O(1)}}]$, *for all* $k \geq 3$ *and* $\epsilon > 0$, *there is no algorithm that approximates Max-Ek-Sat within* $2^{(\log m)^{1-\epsilon}}$ *and runs in time* $2^{(\log m)^{O(1)}}$.

Proof. The proof is again using a gadget. We use the observation that Theorem 1.2 holds even for systems of linear equations in which each equation contains exactly k variables. We can replace each such equation by 2^{k-1} clauses such that if the linear equation is satisfied, then all clauses are satisfied while if it is falsified, one of the clauses is falsified. For example, the equation $x_1x_2x_3 = 1$ can be replaced by the clauses $(\bar{x}_1 \vee x_2 \vee x_3)$, $(x_1 \vee \bar{x}_2 \vee x_3)$, $(x_1 \vee x_2 \vee \bar{x}_3)$ and $(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$. If the system of linear equations had m equations of total weight W, we get $2^{k-1}m$ clauses of total weight $2^{k-1}W = W'$. A solution that satisfied equations of weight $W/2 + X$ satisfies clauses of weight $W'(1 - 2^{-k}) + X$ and thus our measure is preserved by the reduction. ∎

It is not clear how to generalize the inapproximability results. In particular, to get any strong inapproximability result in our measure it must be the case that the random assignment algorithm gives the optimal approximation constant in the old, standard, approximation measure. There is no good characterization of which predicates have this property, and hence we cannot hope for a general reduction preserving our new measure.

4. CONCLUSIONS

We present our results for the new approximation measure in tabular form below. The number δ has the meaning "a fixed positive constant" while ϵ can be replaced by "any positive constant."

Problem	*Upper Bound*		*Lower Bound*
Max-Lin-2	$\frac{\epsilon m}{\log m}$	$m^{\frac{1}{2}-\epsilon}$	Assuming NP $\not\subseteq$ BPP
		m^{δ}	Assuming P $\neq$ NP
Max-k-Lin-2	$(\delta)^{-k}\sqrt{m}$	$2^{(\log m)^{1-\epsilon}}$	Assuming NP $\not\subseteq$ DTIME($2^{(\log m)^{O(1)}}$)
		$(\log m)^{\delta}$	Assuming NP $\not\subseteq$ DTIME($2^{O(\log m \log\log m)}$)
		$1/\epsilon$	Assuming P $\neq$ NP

It is an open question to check if Algorithm 1 achieves an approximation ratio of $O(\sqrt{m})$ on every instance. It is easy to construct instances where the found assignment has objective value $O(1)$ but we have been unable to construct such examples where it is also the case that the optimum is large.

We also note that all upper bounds obtained are larger than $\sqrt{m}$ while all lower bounds are smaller than $\sqrt{m}$ and thus a tempting but not very well supported guess might be that $\sqrt{m}$ is the correct answer for many of the problems.

ACKNOWLEDGMENTS

We would like to thank Avi Wigderson for many helpful discussions. We are also grateful to two anonymous referees whose comments helped to improve the presentation of the paper.

REFERENCES

[1] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman, Derandomized graph products, Comput Complexity 5 (1995), 60–76.

[2] N. Alon, G. Gutin, and M. Krivelevich, Algorithms with large domination ratio, J Algorithms 50 (2004), 118–131.

[3] N. Alon and J. Spencer, The probabilistic method, Wiley-Interscience, New York, 2000.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, Proof verification and intractability of approximation problems, J ACM 45(3) (1998), 501–555.

[5] S. Arora and S. Safra, Probabilistic checking of proofs: A new characterization of NP, J ACM 45(1) (1998), 70–122.

[6] W. Beckner, Inequalities in Fourier analysis, Ann of Math 102 (1975), 159–182.

[7] M. Bellare, O. Goldreich, and M. Sudan, Free bits, PCPs and non-approximability—towards tight results, SIAM J Comput 27 (1998), 804–915.

[8] M. Bellare and P. Rogaway, The complexity of approximating a nonlinear program, J Math Program B 69(3) (1995), 429–441.

[9] A. Bonami, Étude des coefficient Fourier des fonctiones de $L^p(G)$, Ann Inst Fourier (Grenoble), 20(2) (1970), 335–402.

[10] M. Blum, A. Kalai, and H. Wasserman, Noise-tolerant learning, the parity problem and the statistical query model, Proc 32nd Annu ACM Symp Theory of Computation, 2000, pp. 435–440.

[11] L. Engebretsen and J. Holmerin, Towards optimal lower bounds for clique and chromatic number, Theoret Comput Sci 299(1–3) (2003), 537–584.

[12] S. Khot, Hardness results for approximate hypergraph coloring, Proc 34th Annu ACM Symp Theory of Computing, 2002, pp. 351–359.

[13] S. Khot, Inapproximability results for max-clique, chromatic number and approximate graph coloring, Proc 42nd IEEE Conf Foundations of Computer Science, 2001, pp. 600–609.

[14] J. Håstad, Some optimal inapproximability results, J ACM 48 (2001), 798–859.

[15] A. Lubotzky, R. Philips, and P. Sarnak, Ramanujan graphs, Combinatorica 8(3) (1988), 261–277.

[16] R. Raz, A parallel repetition theorem, SIAM J Comput 27(3) (1998), 763–803.