The Complexity of Approximating a Nonlinear Program

Mihir Bellare[*]   Phillip Rogaway[†]

October 1993

Abstract

We consider the problem of finding the maximum of a multivariate polynomial inside a convex polytope. We show that there is no polynomial time approximation algorithm for this problem, even one with a very poor guarantee, unless P = NP. We show that even when the polynomial is quadratic (i.e. quadratic programming) there is no polynomial time approximation unless NP is contained in quasi-polynomial time.

Our results rely on recent advances in the theory of interactive proof systems. They exemplify an interesting interplay of discrete and continuous mathematics—using a combinatorial argument to get a hardness result for a continuous optimization problem.

Key words: approximation, optimization, probabilistically checkable proofs, quadratic programming.

Abbreviated title: Approximate Nonlinear Programming

1 Introduction

Many nonlinear optimization problems are not known to admit polynomial time algorithms. In fact, most are NP-hard, so that finding a polynomial time solution is unlikely. Despite this, we often need to solve these "intractable" computational problems. As with NP-hard problems in combinatorial optimization, interest is turning to the development of (efficient) approximation algorithms—algorithms which run in polynomial time and find a solution not too far from an optimal one.

Will approximation succeed? While approximation algorithms for some nonlinear optimization problems do exist, we have no indication of the complexity of approximation in many important cases. Yet in this fledgling field, this seems an important thing to gauge. In particular, for those problems of significant practical interest, it is desirable to find "hardness of approximation" results

[*] High Performance Computing and Communications, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 USA. e-mail: `mihir@watson.ibm.com`.

[†] Department of Computer Science, University of California, Davis, Davis, CA 95616 USA. e-mail: `rogaway@cs.ucdavis.edu`.

which indicate when it is not worthwhile to seek an approximation algorithm—just as NP-hardness results indicate when it is not worthwhile to seek a polynomial time algorithm.

In general, obtaining "hardness of approximation" results has not been easy. In combinatorial optimization, many important problems defied such efforts for years. Recently, however, powerful techniques to indicate hardness of approximation have emerged; using interactive proofs, this exciting work has been able to settle the approximation complexity of a host of combinatorial optimization problems about which little was known before [15, 1, 2]. Typically, these results indicate hardness of approximation by showing that the existence of an approximation algorithm would imply an unbelievably efficient deterministic algorithm for NP.

Here we apply these techniques to nonlinear optimization. We will show that several important nonlinear optimization problems don't possess (efficient) approximation algorithms unless NP has efficient deterministic solutions. The results are strong in that the conclusions hold even for approximation algorithms with very poor guarantees. Yet our constructions and proofs are simple; the strength of our conclusions derives from the powerful results about interactive proofs that underlie this work. We will begin by looking at polynomial programming, and then turn to the special case of most interest: quadratic programming.

1.1 The Complexity of Polynomial Programming

POLYNOMIAL PROGRAMMING is the problem of finding the maximum of a multivariate polynomial $f(x_1, \ldots, x_n)$ inside $S = \{ x \in [0,1]^n : Ax \le b \}$, a "feasible region" specified by a set of linear constraints. This problem is known to be solvable in polynomial space [10] but is not known to be in NP. Denote by f^* and f_* the maximum and minimum of f inside S, respectively. Following [23, 3, 27, 26], we say an algorithm is a μ-approximation, for $\mu\colon \mathsf{N} \to [0,1]$, if it computes $\tilde{f}$ satisfying $|\tilde{f} - f^*| \le \mu(n)[f^* - f_*]$. It was pointed out by Vavasis [27, 26] that it is important, in the context of continuous optimization, to use this definition, as opposed to ones (more frequently used in combinatorial optimization) which measure the quality of an approximation compared only to f^*. The reasons for this are outside the scope of this paper, but Section 2 contains a brief discussion and the reader is referred to [27, 26] for more information. Notice that a 0-approximation is optimal, while the value of f at *any* feasible point is a 1-approximation. A 1-approximation is therefore easy to find. Our result says that efficiently computing an approximation which is even marginally better is as hard as deciding NP in polynomial time.

Theorem 1.1 *There is a constant $\delta > 0$ such that the following is true. Suppose* POLYNOMIAL PROGRAMMING *has a polynomial time, μ-approximation, where $\mu(n) = 1 - n^{-\delta}$. Then* P = NP.

Problem instances in our results include only integers for their numbers, and the results hold even when these integers are encoded in unary. Since the results are negative, this makes them stronger.

1.2 The Complexity of Quadratic Programming

QUADRATIC PROGRAMMING is the special case of POLYNOMIAL PROGRAMMING in which the polynomial f is of total degree 2; that is, maximize $f(x_1, \ldots, x_n) = \sum_{i \ge j} c_{ij} x_i x_j$ inside $S = \{ x \in [0,1]^n : Ax \le b \}$. It is probably the most important of the nonlinear optimization problems, with applications including economics, planning and genetics.

On the positive side, QUADRATIC PROGRAMMING is known to be in NP [25]. The convex case admits a polynomial time solution [19]. The concave and indefinite cases admit μ-approximation algorithms which, for any constant $\mu \in (0,1)$, are polynomial time under certain conditions on the objective function [26, 27]. The general case admits a weak polynomial time approximation algorithm; specifically, one which achieves a $(1-\Theta(n^{-2}))$-approximation [28].

On the negative side, QUADRATIC PROGRAMMING is NP-hard [24]. In fact, the existence of a polynomial time $.75n^{-1}$-approximation algorithm for this problem already implies P = NP [27]. In other words, finding an excellent approximation algorithm is unlikely. We improve this result to show that even finding a terrible approximation algorithm is unlikely. We say that a function of n is *quasi-polynomial* if it is bounded above by $n^{\log^c n}$ for some constant $c > 0$.

Theorem 1.2 *There is a constant $\delta > 0$ such that the following is true. Suppose* QUADRATIC PROGRAMMING *has a polynomial time, μ-approximation, where $\mu(n) = 1 - 2^{-\log^\delta n}$. Then any problem in* NP *can be solved in quasi-polynomial time.*

The conclusion can be strengthened to P = NP at the cost of raising the quality of approximation shown hard.

Theorem 1.3 *There is a constant $\mu \in (0,1)$ such that the following is true. Suppose* QUADRATIC PROGRAMMING *has a polynomial time μ-approximation. Then* P = NP.

The value of μ that can be achieved in the above depends on the "error probability" achievable by a two prover, one round proof for NP. Combining results of [2] and [14] with our proof, it is possible to achieve any constant $\mu < 1/3$.

1.3 Background, Techniques and Related Work

Our results rely on recent advances in the theory of interactive proof systems and the connection of these to approximation problems. We give a brief summary of relevant work in this area.

Interactive proofs were introduced by Goldwasser, Micali and Rackoff [18] and Babai [4]. Ben-Or, Goldwasser, Kilian and Wigderson [9] extended these ideas to define a notion of multi-prover interactive proofs. Applications of interactive proof based ideas to the derivation of hardness of approximation results emerged in the work of Condon [11] and Feige, Goldwasser, Lovász, Safra and Szegedy [15]. The latter showed that the size of a maximum independent set in a graph is hard to approximate. Their proof exploited a powerful result of Babai, Fortnow and Lund [5] which equates the class MIP of languages possessing multi-prover interactive proofs of membership with the class NEXP of languages recognizable in non-deterministic exponential time. Subsequent constructions of proof systems of lower complexity has lead to better results on the hardness of approximation [15, 1, 2].

To prove Theorem 1.1 we reduce the problem of computing the size of a maximal independent set in a graph to POLYNOMIAL PROGRAMMING in an approximation-preserving way and then apply the maximum independent set approximation hardness results of [15, 1, 2]. The reduction underlies a simple special case of a theorem of Ebenegger, Hammer and de Werra [12], who show that the maximal size of an independent set in a graph is the maximum of some multivariate polynomial associated to it.

Two prover, one round proofs are multi-prover proofs in which there are only two provers and the interaction is restricted to one round. Using techniques of Lapidot and Shamir [20], it was shown by Feige [13] that two provers and one round of interaction suffice to recognize any NEXP language with exponentially small error probability. This result, "scaled down" to NP (cf. Theorem 2.4) is the basis for our proof of Theorem 1.2. A different result about two prover, one round proofs (cf. Theorem 2.5) is the basis of the proof of Theorem 1.3.

The particular association of a quadratic program to a two-prover, one-round interactive proof that we use was independently discovered by Feige and Lovász [16]. Meanwhile the original work of Feige [13] on which some of our results were based has also been incorporated into this same joint paper with Lovász [16].

The present work and [16, 6] were the first to use two prover, one round proofs to show hardness of approximation results. Later these proofs systems were also used by [21].

QUARTIC PROGRAMMING is the special case of POLYNOMIAL PROGRAMMING in which the objective function is a polynomial of degree four. Since QUADRATIC PROGRAMMING is a special case of QUARTIC PROGRAMMING, the results of Theorems 1.2 and 1.3 apply. For the quartic case, however, a stronger result than Theorem 1.3 was recently obtained in [7]; the authors show that for *any* constant $\mu \in (0,1)$, if QUARTIC PROGRAMMING has a polynomial time, μ-approximation, then P = NP.

A preliminary version of this paper appeared as [8].

2 Preliminaries

The notation $|\cdot|$ will be used to denote the absolute value of a number, the length of a string, or the size of a set; the context will disambiguate. An optimization problem is specified by a pair (S,g). Here S is a map assigning to each *instance* w a set $S(w)$ called the *solution space* or *feasible region*, and $g(w,y)$ is the *utility* of a solution $y \in S(w)$. An instance w is degenerate if $S(w) = \emptyset$. The problem is to maximize the utility of a non-degenarate instance over the feasible region. (Minimization problems can be accommodated by modifying these definitions in the obvious ways.)

Let $\overline{\mathsf{R}} = \mathsf{R} \cup \{\infty, -\infty\}$. For non-degenerate w define $g^*(w), g_*(w) \in \overline{\mathsf{R}}$ by $g^*(w) = \sup_{y \in S(w)} g(w,y)$ and $g_*(x) = \inf_{y \in S(w)} g(w,y)$. A non-degenerate instance w is bounded if $g^*(w)$ and $g_*(w)$ are finite. Following [23, 3, 27, 26] we measure the quality of an approximation $\tilde{g}$ by seeing how much it differs from g^*, as measured in units of $|g^* - g_*|$. For simplicity we will only talk of approximation when the instance is (non-degenerate and) bounded, so that the unit of measurement $|g^* - g_*|$ is finite.

Definition 2.1 *Let (S,g) be an optimization problem, $\|\cdot\|$ a map from strings to N, and μ a map from N to $[0,1]$. A μ-*approximation *for (S,g) and norm $\|\cdot\|$ is a function $\tilde{g}$ which, on any non-degenerate, bounded instance w gives a number $\tilde{g}(w) \in \mathsf{R}$ for which*

$$|g^*(w) - \tilde{g}(w)| \;\leq\; \mu(\|w\|) \cdot |g^*(w) - g_*(w)| \;.$$

The norm is that aspect of the input in terms of which the quality of approximation μ is measured. For graph problems the norm is the number of nodes in the graph. For programming problems it is the number of variables in the program. An attribute of this definition which renders it preferable, in this context, to other definitions is the invariance under scaling: if $\tilde{g}$ is a μ-approximation to (S,g), then $a\tilde{g} + b$ is a μ-approximation to $(S, ag + b)$, for any constants a and b; this corresponds, for example, to the fact that measuring utility in different units should not affect the quality of an approximation. Another such attribute is the invariance under affine linear transformations of the feasible region and the objective function. For more information on the definition we refer the reader to [27, 26].

It is not required that an approximation algorithm “find” the point with the specified utility; it is not even required that there exist a point $\tilde{y} \in S(w)$ such that $\tilde{g} = g(w, \tilde{y})$. This is therefore a weak notion of approximation. Since our results are negative, this serves only to strengthen them.

We will be interested in polynomial time approximation algorithms. To avoid confusion, we emphasize that while the quality of the approximation is measured in terms of the norm, the running time of the approximation algorithm is measured, as usual, as a function of the length of the encoding of the instance. Approximation algorithms will return rationals, encoded as pairs of integers, each integer itself encoded as usual in binary.

A list of optimization problems we will consider follows. All numbers in problem instances are integers; this eliminate issues concerning computational complexity over the reals. Furthermore, the integers in problem instances are specified in *unary*; since our results are negative, this makes them stronger. In all the programming problems the feasible region is restricted to a subset of $[0,1]^n$ and the utility functions are continuous, so all (non-degenerate) instances are bounded.

INDEPENDENT SET

Instance: A graph $G = (V, E)$.
Solutions: $W \subseteq V$ is a solution if it is an independent set: for each $u, v \in W$, $\{u, v\} \notin E$.
Utility of Solutions: A solution W for the instance G has utility $|W|$.

POLYNOMIAL PROGRAMMING

Instance: Number n and t, and for each $k \in [1..t]$, an integer c_k and a subset A_k of $\{1, \dots, n\}$. Together this encodes a polynomial $f(x_1, \dots, x_n) = \sum_{k=1}^{t} c_k \left[(\prod_{i \in A_k} x_i)\right]$. Also an $m \times n$ integer matrix A and an integer m-vector b.
Solutions: A vector $x \in [0,1]^n$ is a solution if $Ax \leq b$.
Utility of Solutions: A solution x has utility $f(x)$.

POLYNOMIAL PROGRAMMING–RESTRICTED CASE

Instance: Numbers n and t, and for each $k \in [1..t]$, a pair A_k, B_k of disjoint subsets of $\{1, \dots, n\}$. Together, this encodes the polynomial $f(x_1, \dots, x_n) = \sum_{k=1}^{t} \left[(\prod_{i \in A_k} x_i) \cdot (\prod_{j \in B_k} (1 - x_j))\right]$.
Solutions: Any vector $x \in [0,1]^n$ is a solution.
Utility of Solutions: A solution x has utility $f(x)$.

QUADRATIC PROGRAMMING

Instance: A number n and, for each $i, j \in \{1, \dots, n\}$ with $i \leq j$, an integer c_{ij}. Together this encodes the quadratic polynomial $f(x) = \sum_{i \leq j} c_{ij} x_i x_j$. Also an $m \times n$ integer matrix A and an integer m-vector b.
Solutions: A vector $x \in [0,1]^n$ is a solution if $Ax \leq b$.
Utility of Solutions: A solution x has utility $f(x)$.

INDEPENDENT SET was shown hard to approximate by [15, 1, 2]. Stating the last of these results in terms of our definition we get the result we will use.

Theorem 2.2 *There is a constant $\delta > 0$ such that the following is true. Suppose* INDEPENDENT SET *has a polynomial time, μ-approximation, where $\mu(n) = 1 - n^{-\delta}$. Then* P = NP.

A two-prover, one-round interactive proof system involves a probabilistic, polynomial time *verifier*, V, and a pair of (computationally unbounded, deterministic) *provers,* A and B. Formally, a verifier is a pair of functions (π, ρ), each computable in time polynomial in the length of its first argument; π takes two string arguments and returns a string, and ρ takes five string arguments and returns a bit. A prover is a function which takes two string arguments and returns a string. Each prover can communicate with the verifier, but they can neither talk to one another once the protocol begins, nor can either prover see the communication between the verifier and the other prover. The parties share a common input w, and it is the provers (joint) goal to convince V to accept this string. To this end, the parties engage in a simple interaction, which is begun by the verifier. The latter applies π to the common input w and a string R (the verifier's random tape) to get a pair of "questions" p, q. He then sends p to A and q to B. The provers then provide answers, A sending the answer $a = A(w, p)$, and B sending $b = B(w, q)$. After the verifier receives his answers, he computes $\rho(w, p, q, a, b)$. If this value is 1 he is considered to "accept" else to "reject."

The number of coins flipped by the verifier and the size of answers sufficient to convince him are the attributes of the verifier which are important in our construction. We say that a verifier V has complexity $l\colon \mathsf{N} \to \mathsf{N}$ if, when the common input has length n, a random tape of length $l(n)$ suffices to produce the questions p, q, and $\rho(w, p, q, a, b) = 0$ if either a or b have length different from $l(n)$. It is convenient, although not necessary, to also assume that the lengths of the questions p, q are equal to $l(n)$. We denote by $\pi^i(w, R)$ the question to the i-th prover, $i = 1, 2$.

Definition 2.3 *Let $V = (\pi, \rho)$ be a verifier of complexity l. Let (A, B) be a pair of provers. For each w, let $\mathtt{ACC}_{V,(A,B)}(w)$ denote the probability that*

$$\rho(w, \pi^1(w, R), \pi^2(w, R), A(w, \pi^1(w, R)), B(w, \pi^2(w, R))) \ = \ 1$$

when R is chosen at random from $\{0, 1\}^{l(|w|)}$. The accepting probability of the verifier V at w *is the maximum of $\mathtt{ACC}_{V,(A,B)}(w)$ over all possible pairs (A, B) of provers. We denote it by $\mathtt{ACC}_V(w)$. If L is a language and ϵ a function of N to $[0, 1]$, we say that V has error probability ϵ with respect to L if the following two conditions hold: first, $w \in L$ implies $\mathtt{ACC}_V(w) = 1$; second, $w \notin L$ implies $\mathtt{ACC}_V(w) < \epsilon(|w|)$.*

We say that a language L has a two-prover, one-round proof with complexity l and error probability ϵ if there exists a verifier having complexity l and error probability ϵ with respect to L. Important to our results is the fact that NP-complete languages have two prover one round proofs of very low complexity. As usual, SAT denotes the decision problem for satisfiability of Boolean formulas.

Theorem 2.4 [13, 20] *There is a constant $c > 0$ such that* SAT *has a two prover, one round proof with complexity $O(\log^c n)$ and error probability $1/n$.*

If constant error probability suffices, the complexity can be reduced to logarithmic. The following result is derived by applying a transformation of [17] to the main result of [2].

Theorem 2.5 *There is a constant $\epsilon < 1$ such that* SAT *has a two prover, one round proof with complexity $O(\log n)$ and error probability ϵ.*

The current best value for the constant c in the first theorem is $c = 3$ [13, 20]. For the second it is possible to achieve any constant $\epsilon > 1/2$ (cf. [2, 14]).

3 The Complexity of Polynomial Programming

We prove Theorem 1.1. Let $G = (V, E)$ be an instance of INDEPENDENT SET. We will construct from G an instance f of POLYNOMIAL PROGRAMMING–RESTRICTED CASE where $f^* = G^*$ and $\|f\|$ is polynomially bounded in $\|G\|$. We will then explain why our reduction is enough to establish the theorem.

Without loss of generality, assume G has no isolated nodes. The program f is constructed as follows. Introduce a formal variable x_e for each edge $e \in E$. To each edge $e = \{u, v\}$, arbitrarily order its endpoints (u, v) and associate the polynomial x_e with endpoint u and the polynomial $1 - x_e$ with endpoint v, defining $x_{uv} = x_e$ and $x_{vu} = 1 - x_e$. The polynomial f is defined as the sum, over all vertices, of the product of the polynomials associated to that vertex: $f(x) = \sum_{u \in V} \prod_{v \in N(u)} x_{uv}$, where $N(u)$ is the set of all vertices adjacent to u. This is a degree $\Delta = \max_u deg(u)$ polynomial in $m = |E|$ variables. An algorithm which reduces INDEPENDENT SET to POLYNOMIAL PROGRAMMING–RESTRICTED CASE constructs from graph G the polynomial f described above, obtains an estimate of its maximum in $[0, 1]^m$, and then returns this as its own estimate for the size of the maximal independent set in G. Note that f is easily constructed from G in polynomial time, and f has norm (number of variables) which is at most the square of the norm of G (the number of nodes).

Let $f^* = \max_{0 \le x_e \le 1} f(x)$ denote f's maximum inside the m-dimensional unit hypercube, and let G^* denote the size of a maximum independent set of G. We show that $G^* = f^*$.

First, we claim that $f^* \ge G^*$. For given an independent set W of cardinality ω, one constructs an assignment $x = \{x_e\}$ of utility at least ω by setting $x_e = 1$ if $u \in W$ and e is ordered (u, v); by setting $x_e = 0$ if $u \in W$ and e is ordered (v, u); and by setting x_e arbitrarily otherwise. The assignment is well defined by W being an independent set. We have $f^* \ge G^*$ because $f(x) \ge \sum_{u \in W} \prod_{v \in N(u)} x_{uv} = \omega$.

Conversely, $G^* \ge f^*$. For given an assignment $x = \{x_e\}$, construct an independent set W of cardinality at least $\lceil f(x) \rceil$ as follows: Choose an edge $e = \{u, v\}$ and set $\pi_u = \prod_{r \in N(u) - \{v\}} x_{ur}$, and $\pi_v = \prod_{r \in N(v) - \{u\}} x_{vr}$. Now if $\pi_u \le \pi_v$, then adjust the assignment by "pushing" all x_{uv} units from u to v and obtain an assignment x' of at least as great a value as that of x; that is, letting $x'_e = x_e$ apart from setting $x_e = 0$ if $e = (u, v)$ and $x_e = 1$ if $e = (v, u)$, we have $g(x') \ge g(x)$, as $g(x') - g(x) = x_{uv}(\pi_v - \pi_u) \ge 0$. If, instead, $\pi_u > \pi_v$, then let $x' = x$ except $x_e = 1$ if $e = (u, v)$ and $x_e = 0$ if $e = (v, u)$; this again ensures that $g(x') \ge g(x)$. Repeating this process for each edge of G gives an assignment x'' with $g(x'') \ge g(x)$ and each $x''_e \in \{0, 1\}$. Consider the set of vertices $W = \{u \in V : x''_{uv} = 1 \text{ for all } v \in N(u)\}$. Then W is an independent set of vertices and $|W| = g(x'') \ge g(x)$.

We have shown how to efficiently map G to f and $\tilde{f}$ to $\tilde{G}$. Now suppose we had a $(1 - \|f\|^{-2\delta})$-approximation for POLYNOMIAL PROGRAMMING–RESTRICTED CASE. Since $f^* = G^*$, $f_* = G_*$ (both are 0), and $\|f\| \le \|G\|^2$, we immediately get a $(1 - n^{-4\delta})$-approximation for INDEPENDENT SET. Likewise, the straightforward reduction from POLYNOMIAL PROGRAMMING–RESTRICTED CASE to POLYNOMIAL PROGRAMMING, in which each expression $1 - x_j$ is replaced by a formal variable x'_j and constraints are added to enforce that $x'_j = 1 - x_j$, also preserves the optimal value, the worst value, is efficient, and at most doubles the norm. Thus a $(1 - n^{-\delta})$-approximation for POLYNOMIAL PROGRAMMING easily gives a $(1 - n^{-2\delta})$-approximation for POLYNOMIAL PROGRAMMING–RESTRICTED CASE. Putting this all together and using Theorem 2.2, with the constant δ of the present theorem being 4 times the constant of Theorem 2.2, we conclude our result.

4 The Complexity of Quadratic Programming

We prove Theorem 1.2. Let $V = (\pi, \rho)$ be the verifier and c the constant of the two prover one round proof of Theorem 2.4, and let l be the complexity of this verifier as specified by the theorem. Let $\tilde{g}$ be a quasi-polynomial time, μ-approximation algorithm for QUADRATIC PROGRAMMING. To prove the theorem, we specify a quasi-polynomial time decision procedure M for SAT. We let w denote the input to M, with N denoting its length. We write l for $l(N)$. If $\rho(w, p, q, a, b) = 1$ then let $\hat{\rho}(p, q, a, b)$ be the number of strings $R \in \{0,1\}^l$ satisfying $\pi(w, R) = (p, q)$. Otherwise, let $\hat{\rho}(p, q, a, b) = 0$. For each $p, a \in \{0,1\}^l$ we introduce a variable $x_{p,a}$, and for each $q, b \in \{0,1\}^l$ we introduce a variable $y_{q,b}$. On input w, algorithm M constructs the $n = 2^{2l+1}$ variable quadratic program

$$\textit{maximize:} \quad f(xy) \;=\; \textstyle\sum_{p,a,q,b \in \{0,1\}^l} \;\hat{\rho}(p,q,a,b) \cdot x_{p,a} \cdot y_{q,b}$$

$$\textit{subject to:} \begin{cases} \sum_{a \in \{0,1\}^l} x_{p,a} = 1 & \text{for each } p \in \{0,1\}^l, \\ \sum_{b \in \{0,1\}^l} y_{q,b} = 1 & \text{for each } q \in \{0,1\}^l, \text{and} \\ 0 \le x_{p,a}, y_{q,b} \le 1 & \text{for all } p,q,a,b \in \{0,1\}^l. \end{cases}$$

We denote by (f, A, b) this QUADRATIC PROGRAMMING instance, where A and b are defined so that $A(xy) \le b$ captures the above set of constraints. M now applies the approximation algorithm $\tilde{g}$ to this program, and lets $\tilde{f}$ denote the output. If $\tilde{f} \ge 2^l(1 - \mu(n))$ then M accepts; otherwise, it rejects. To see that M runs in quasi-polynomial time, first note that the length of an encoding of (f, A, b) is $2^{\lg^{O(1)} N}$, even with the encoding in unary as we assume. And this program can easily be computed in $2^{\lg^{O(1)} N}$ time. M will run the quasi-polynomial time algorithm $\tilde{g}$ on an input of quasi-polynomial size in N, so the total running time is quasi-polynomial in N. To argue the correctness of M, let $f^* = \max_{xy: A(xy) \le b} f(xy)$ denote the maximum value of f over its feasible region. A boolean point is one all of whose coordinates are 0 or 1. Standard arguments show that the maximum of f occurs at a boolean point:

Lemma 4.1 *There exists a boolean point x^*y^* in the feasible region of (f, A, b) such that $f^* = f(x^*y^*)$.*

This enable us to see that the maximum value of f equals, apart from a scaling factor, the value of the interaction defined by V.

Lemma 4.2 $\mathtt{ACC}_V(w) = 2^{-l} \cdot f^*$.

Proof: Let (A, B) be a pair of provers. Set $x_{p,a} = 1$ if $a = A(w, p)$ and 0 otherwise, and $y_{q,b} = 1$ if $b = B(w, q)$ and 0 otherwise. Then xy is a (boolean) point in the feasible region, and $2^{-l} \cdot f(xy) = \mathtt{ACC}_{V,(A,B)}(w)$. Since (A, B) was arbitrary it follows that $2^{-l} \cdot f^* \ge \mathtt{ACC}_V(w)$.

Conversely, by Lemma 4.1 we know that there is a boolean point xy in the feasible region such that $f^* = f(xy)$. Since xy is both boolean and feasible it must be that for each p there is a unique $\bar{a}$ such that $x_{p,a} = 1$ if $a = \bar{a}$ and $x_{p,a} = 0$ otherwise. Set $A(w, p) = \bar{a}$. Construct B correspondingly from y. Then note that $\mathtt{ACC}_{V,(A,B)}(w) = 2^{-l} \cdot f(xy)$. So $\mathtt{ACC}_V(w) \ge 2^{-l} \cdot f^*$. ■

We note that the minimum f_* of f over the feasible region is nonnegative. The approximation thus guarantees $|f^* - \tilde{f}| \le \mu(n) f^*$. Now if $w \in$ SAT, then the program constructed has maximum

$f^* = 2^l$, while if $w \notin$ SAT, then it has maximum $f^* < \epsilon 2^l$ where $\epsilon = 1/N$ is the error probability of V with respect to SAT. (cf. Theorem 2.4 and Lemma 4.2). So

(1) if $w \in$ SAT then $\tilde{f} \geq 2^l(1 - \mu(n))$, and

(2) if $w \notin$ SAT then $\tilde{f} < \epsilon 2^l(1 + \mu(n))$.

Thus our decision procedure for SAT is correct as long as $\epsilon 2^l(1 + \mu(n)) \leq 2^l(1 - \mu(n))$. Simplifying this expression yields that $\mu(n)$ must be at most $(1-\epsilon)/(1+\epsilon) = (N-1)/(N+1)$. Since $l = \lg^c N$ and $n = 2^{\Theta(l)}$ we may certainly find a positive δ so that defining μ as in the theorem statement does indeed guarantee $\mu(n) \leq (N-1)/(N+1)$.

An analogous proof with Theorem 2.5 substituted for Theorem 2.4 yields Theorem 1.3.

We believe these results could be improved to show that there is a constant $\delta > 0$ such that the following is true: if QUADRATIC PROGRAMMING has a polynomial time, μ-approximation, where $\mu(n) = 1 - n^{-\delta}$, then P = NP. One way to do this would be to construct two prover, one round proof systems for SAT which have appropriate complexity and error probability. Specifically, it suffices that the verifier use logarithmic randomness and answer sizes to achieve error probability $1/n$. (The length of the questions p, q is not important; our construction is easily modified so that the size of the quadratic program associated to the verifier and a string w depends only on the randomness and answer sizes.)

Acknowledgments

We are grateful to Peter Hammer, who, during a visit to Dartmouth College, described the work in [12] which inspired our initial results. We thank Stephen Vavasis for much helpful information on the subject of nonlinear optimization, and especially for explaining to us the importance of using the right definition of a μ-approximation. We thank Rajeev Motwani for drawing our attention to [3]. We thank an anonymous referee for comments.

References

[1] S. Arora and S. Safra, "Probabilistic checking of proofs; a new characterization of NP," *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science* (IEEE Computer Society Press, 1992) 2–13.

[2] S. Arora, C. Lund, M. Motwani, M. Sudan and M. Szegedy, "Proof verification and hardness of approximation problems," *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science* (IEEE Computer Society Press, 1992) 14–23.

[3] G. Ausiello, A. D'Atri and M. Protasi, "Structure preserving reductions among convex optimization problems," *Journal of Computer and System Sciences* **21** (1980) 136–153.

[4] L. Babai, "Trading group theory for randomness," *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing* (ACM Press, 1985).

[5] L. Babai, L. Fortnow and C. Lund, "Non-deterministic exponential time has two-prover interactive protocols," *Computational Complexity* 1 (1991) 3–40.

[6] M. Bellare, "Interactive proofs and approximation," *IBM Research Report* RC 17969 (New York, 1992). Also *Proceedings of the Second Israel Symposium on Theory of Computing and Systems* (1993).

[7] M. Bellare, S. Goldwasser, C. Lund and A. Russell, "Efficient probabilistically checkable proofs and applications to approximation," *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing* (ACM Press, 1993) 294–304.

[8] M. Bellare and P. Rogaway, "The complexity of approximating a nonlinear program," *IBM Research Report* RC 17831 (New York, 1992).

[9] M. Ben-Or, S. Goldwasser, J. Kilian and A. Wigderson, "Multi-prover interactive proofs: how to remove intractability assumptions," *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing* (ACM Press, 1988) 113–131.

[10] J. Canny, "Some algebraic and geometric computations in PSPACE," *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing* (ACM Press, 1988).

[11] A. Condon "The complexity of the max word problem and the power of one-way interactive proof systems," *Computational Complexity* 3 (1993) 295–305.

[12] C. Ebenegger, P. Hammer and D. de Werra, "Pseudo-boolean functions and stability of graphs," in *Algebraic and Combinatorial Methods in Operations Research,* Annals of Discrete Mathematics 19 (North Holland Mathematics Studies, 95, 1984) 83–97.

[13] U. Feige, "NEXPTIME has two-provers one-round proof systems with exponentially small error probability" (1991). Manuscript.

[14] U. Feige, "On the success probability of the two provers in one round proof systems," *Proceedings of the Sixth Annual Structure in Complexity Theory Conference* (IEEE Computer Society Press, 1991) 116–123.

[15] U. Feige, S. Goldwasser, L. Lovász, S. Safra and M. Szegedy, "Approximating clique is almost NP-complete," *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science* (IEEE Computer Society Press, 1991) 2–12.

[16] U. Feige and L. Lovász, L. (1992), "Two-prover one round proof systems: their power and their problems," *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing* (ACM Press, 1992) 733–744.

[17] L. Fortnow, L., J. Rompel and M. Sipser "On the power of multiprover interactive protocols," *Proceedings of the Third Annual Structure in Complexity Theory Conference* (IEEE Computer Society Press, 1988).

[18] S. Goldwasser, S. Micali and C. Rackoff, (1989), "The knowledge complexity of interactive proofs," *SIAM J. Computing* 18 (1989) 186–208.

[19] M. Kozlov, S. Tarasov and L. Hačijan, "Polynomial solvability of convex quadratic programming," *Dokl. Akad. Nauk SSSR* 248 (1979) 1049–1051. [Translated as *Soviet Math Dokl.* 20, 1108–1111.]

[20] D. Lapidot and A. Shamir, "Fully parallelized multi-prover protocols for NEXP-time," *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science* (IEEE Computer Society Press, 1991) 13–18.

[21] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems," *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing* (ACM Press, 1993) 286–293.

[22] T. Motzkin and E. Straus, "Maxima for graphs and a new proof of a theorem by Tuán," *Notices of the American Mathematical Society* 11 (1964) 533–540.

[23] A. Nemirovsky, and D. Yudin, *Slozhnost' Zadach i Effektivnost' Metodov Optimizatsii* (1979). [Translated by E. Dawson as *Problem Complexity and Method Efficiency in Optimization*, (John Wiley and Sons, 1983).]

[24] S. Sahni, (1974), "Computationally related problems," *SIAM J. of Computing* 3 (1974) 262–279.

[25] S. Vavasis, "Quadratic programming is in NP," *Information Processing Letters* 36 (1990) 73–77.

[26] S. Vavasis, "Approximation algorithms for indefinite quadratic programming," TR 91-1228, Department of Computer Science, Cornell University (New York, 1991). To appear in *Mathematical Programming*.

[27] S. Vavasis, "On approximation algorithms for concave programming," in: C.A. Floudas and P.M. Pardalos, ed., *Recent Advances in Global Optimization* (Princeton University Press, 1992) 3–18.

[28] S. Vavasis, "Polynomial time weak approximation algorithms for quadratic programming," in: P. Pardalos, ed., *Complexity in Numerical Optimization* (World Scientific, 1992) 490–500.