LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-452

THE COMPLEXITY OF CONTINUOUS OPTIMIZATION

Phillip Rogaway

June 1991

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

The Complexity of Continuous Optimization

Phillip Rogaway

Abstract

Given a polynomial objective function $f(x_1, \ldots, x_n)$, we consider the problem of finding the maximum of this polynomial inside some convex set $D = \{x : Ax \leq B\}$. We show that, under a complexity assumption, this extremum cannot be *approximated* by *any* polynomial-time algorithm, *even exceedingly poorly*. This represents an unusual interplay of discrete and continuous mathematics: using a combinatorial argument to get a hardness result for a continuous optimization problem.

1 Introduction

An enormous literature exists on finding the maximum (or minimum) of an objective function $f(x)$ inside some "feasible region" $x \in D$.

Positive results have attracted the most attention. For certain special cases, polynomial-time algorithms are known. The most celebrated example of this is LINEAR PROGRAMMING, where the objective function is linear function, $f(x_1, \ldots, x_n) = \sum c_i x_i$ and the constraints specify a convex set $D = \{x : Ax \leq B\}$.

Negative results for continuous optimization are less well studied. One example is that of QUADRATIC PROGRAMMING, which is NP-hard [6]. This is useful information for the algorithm designer, suggesting that he not seek a polynomial-time algorithm for solving the general case of this problem. In cases such as this, a reasonable strategy is to seek provably good approximation algorithms.

Here we give strong evidence that this, too, can sometimes be too much to hope for. In particular, we show that, under a complexity assumption, it is impossible to approximate the maximum of a polynomial objective function inside some convex set. (Some complexity assumption would appear necessary, insofar as the conclusion implies P$\neq$NP.) Even finding a very poor approximation is impossible. The instance of the problem consists of a a polynomial $f(x_1, \ldots, x_n)$, described as a formal sum of terms, $\sum_{k=1}^{t} c_k \prod_{i \in A_k} x_i$, together with a matrix A and vector b specifying the region $D = \{x : Ax \leq b\}$.

In fact, the instance problem can be more constrained. It can be taken as a polynomial given by a formal sum of "terms," each "term" a product of some collection of x_i's and $(1 - x_j)$'s; that is, an instance looks like $f(x) = \sum_{k=1}^{t} \left[(\prod_{i \in A_k} x_i) \cdot (\prod_{j \in B_k} (1 - x_j)) \right]$, where the A_k's and B_k's are disjoint. The problem is to find the maximum of this polynomial inside the unit hypercube.

Our result represents an unusual interplay of discrete and continuous mathematics for achieving a negative result: in particular, we are using a *combinatorial* argument to say that we (probably) can *not* solve a class of optimization problems with objective functions that are well-behaved, *continuous* functions.

The proof of this theorem is to observe that there is an *approximation-preserving reduction* from our optimization problem to the problem of computing the size of a maximal cardinality clique in a graph. This reduction underlies a simple special case of a theorem of Ebenegger, Hammer, and

de Werra [1], which says that the maximal size of an independent set in a graph is the maximum of some multivariate polynomial associated to it. One concludes our theorem by using the recent result of Feige, Goldwasser, Lovász, Safra and Szegedy [2], which shows that, under a complexity assumption, the size of a maximal clique in a graph can not be estimated *even exceedingly poorly* (not within any constant, say).

2 Framework for Optimization and Approximation

In this section we modify somewhat earlier notions for approximation-preserving reductions put forth by Papadimitriou and Yannakakis [5], Panconesi and Ranjan [4], and others. We begin by defining the class of problems we will consider.

2.1 NP-Approximation Problems

We are interested only in *NP-optimization problems,* which we now define. Such a problem is given by a polynomial-time computable function $f : \{0,1\}^* \times \{0,1\}^* \to \mathbf{R}$, which associates to each *instance* x and *solution* y a real-valued *utility* $f(x,y)$. Saying f is polynomial-time computable means it can be computed in time polynomial in the length of its *first* argument.

As a matter of convenience, we consider maximization problems. For an optimization problem f, we define its *maximum* at x by $f^*(x) = \max_{y \in \Sigma^*} f(x,y)$. This maximum is a well-defined and Turing-computable real number by the assumption that f is polynomial-time. A solution y^* for which $f(x,y^*) = f^*(x)$ is called an *optimal solution* to f.

The *optimal solution problem* associated to an optimization problem f is the following: given instance x, find an optimal solution y^*.

We shall be interested in algorithms which fall short of this goal. To measure how close an algorithm comes to it, we speak of an approximation as being multiplicatively within some factor μ; addatively within some amount α; and, because we may wish μ and α to be functions of the input, we require a "norm" $\|\cdot\|$ measuring the size of input instances, and let μ and α depend on this. This is formalized below.

Definition 2.1 *A* (μ,α)-approximation *for the* NP*-optimization problem* f *and the norm* $\|\cdot\|$*, where* μ, α*, and* $\|\cdot\| : \{0,1\}^* \to \mathbf{R}$*, is a function* A *which, on any instance* x*, gives a solution* $\tilde{y} = A(x)$ *for which*

$$\frac{f^*(x)}{\mu(n)} - \alpha(n) \;\le\; f(x,\tilde{y}) \;\le\; \mu(n)\cdot f^*(x) + \alpha(n),$$

where $n = \|x\|$*.* *A* μ-approximation *is a* (μ,α)*-approximation for some constant* α*.* *A* constant-factor approximation *is a* (μ,α)*-approximation, for some constants* μ *and* α*.*

This definition holds equally well for maximization and minimization problems, though, in each case, obviously only one of the two inequalities above is relevant.

We may wish to admit probabilistic approximation algorithms A. In this case, A's input is the pair $(x, 1^k)$ and, for any instance x, with probability at least $1 - 2^{-k}$, algorithm A must find a $\tilde{y}$ meeting the constraints above.

2.2 Approximation-Preserving Reductions

Rather than mimicking the usual "Cook reductions" we prefer to define approximation-preserving "Turing reducibility." (A terminological history appears in [3].) Though this added generality is not needed here, we expect it to be significant in other contexts.

Consider algorithms A with access to an oracle $\tilde{g}$ computing approximate solutions for some optimization problem g. Such an algorithm will be denoted $A^{\tilde{g}}$. An oracle call by A is counted as taking one unit of time.

Definition 2.2 *Let f and g be optimization problems. We say that f (μ, α)-*reduces to *g with respect to norms $\|\cdot\|_f$ and $\|\cdot\|_g$, written $f \leq_{\mu,\alpha} g$, if there is a polynomial-time Turing machine A such that if $\tilde{g}$ is any (μ, α)-approximation to g (with respect to $\|\cdot\|_g$), then $A^{\tilde{g}}$ is a (μ, α)-approximation algorithm for f (with respect to $\|\cdot\|_f$). We say that f* approximation-preserving reduces to *g (with respect to norms $\|\cdot\|_f$ and $\|\cdot\|_g$), written $f \leq_A g$, if there is a polynomial-time Turing machine A such that, for any constants μ and α, if $\tilde{g}$ is a (μ, α)-approximation to g with respect to $\|\cdot\|_g$ then $A^{\tilde{g}}$ is a (μ, α)-approximation algorithm for f with respect to $\|\cdot\|_g$. If $f \leq_A g$ and $g \leq_A f$, then the problems are said to be* equivalent *with respect to approximation, written $f \equiv_A g$.*

3 Preliminaries

3.1 Some Continuous and Discrete Optimization Problems

We define the following optimization problems. Solutions which do not encode the specified combinatorial structure have utility 0.

CLIQUE
Instance: A (simple, finite, undirected) graph $G = (V, E)$.
Utility of solutions: A solution W for the instance G has utility $|W|$ if $W \subseteq V$ is a clique: for each $v, w \in W \Rightarrow \{v, w\} \in E$.

INDEPENDENT SET
Instance: A (simple, finite, undirected) graph $G = (V, E)$.
Utility of solutions: A solution W for the instance G has utility $|W|$ if $W \subseteq V$ is an independent set: for each $v, w \in W \Rightarrow \{v, w\} \notin E$.

POLYNOMIAL PROGRAMMING
Instance: A polynomial in some number of variables, encoded as a formal sum $f(x_1, \ldots, x_n) = \sum_{k=1}^{t} c_k \left[\left(\prod_{i \in A_k} x_i\right)\right]$, plus an $n \times m$ matrix A and an m-vector b.
Utility of solutions: A n-vector x satisfying $Ax \leq b$ has utility $f(x)$.

POLYNOMIAL PROGRAMMING – RESTRICTED CASE
Instance: A number n and some number t of disjoint sets $A_k, B_k \in \{1, \ldots, n\}$, encoding the polynomial $f(x_1, \ldots, x_n) = \sum_{k=1}^{t} \left[\left(\prod_{i \in A_k} x_i\right) \cdot \left(\prod_{j \in B_k} (1 - x_j)\right)\right]$.
Utility of solutions: A solution $x \in [0, 1]^n$ has utility $f(x)$.

3.2 Properties of Approximation-Preserving Reduction

The following proposition can be verified immediately from the definition:

Proposition 3.1 *Approximation-preserving reductions are transitive: if $f \leq_A g$ and $g \leq_A h$, then $f \leq_A h$.*

The straightforward map between INDEPENDENT SET and CLIQUE instances establishes the following (where $\|G\| = |V(G)|$ is the cardinality of G's vertex set):

Proposition 3.2 CLIQUE $\equiv_A$ INDEPENDENT SET.

The result of [2] and the notion of reducibility given immediately gives the following, where "slightly superpolynomial time" means "time complexity of $n^{O(\log\log n)}$." (The result of [2] gives other tradeoffs for complexity assumptions and the quality of approximations as well.)

Proposition 3.3 *Assume* NP *does not have slightly superpolynomial time algorithms. Then if* CLIQUE $\leq_A f$, f *does not have any polynomial-time constant-factor approximation algorithm.*

4 The Complexity of Polynomial Optimization

Theorem. Assume NP does not have slightly superpolynomial algorithms. Then there is no polynomial-time constant-factor approximation algorithm for POLYNOMIAL PROGRAMMING.

Proof: We show that the conclusion holds even for POLYNOMIAL PROGRAMMING – RESTRICTED CASE. In view of results given so far, it is enough to show that INDEPENDENT SET approximation-preserving reduces to this. (In fact, the problems are approximation-equivalent.) Let $G = (V, E)$ be an instance of the INDEPENDENT SET problem. Without loss of generality, assume G has no isolated nodes. We construct from G a polynomial f (an instance of POLYNOMIAL PROGRAMMING) as follows. Introduce a formal variable x_e for each edge $e \in E$. To each edge $e = \{v, w\}$, arbitrarily order its endpoints (v, w) and associate the polynomial x_e with endpoint v and associate the polynomial $1 - x_e$ with the other endpoint, w. Let $x_{vw} = x_e$ and let $x_{wv} = 1 - x_e$. The polynomial f is the sum, over all vertices, of the product of the polynomials associated to that vertex, $f(x) = \sum_{v \in V} \prod_{w \in N(v)} x_{vw}$, where $N(v)$ is the set of all vertices adjacent to v. This is a degree $\Delta = \max_v deg(v)$ polynomial in n variables.

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

To see this, we first observe that $f^* \geq w^*$. For given an independent set W of cardinality w, one constructs an assignment $x = \{x_e\}$ of utility at least w by setting $x_e = 1$ if e is ordered (v, w) and $v \in W$; by setting $x_e = 0$ if e is ordered (w, v) and $v \in W$; and by setting x_e arbitrarily otherwise. The conclusion follows because $f(x) \geq \sum_{v \in W} \Pi_{w=\in N(v)} x_{vw} = w$.

Conversely, $w^* \geq f^*$. 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 $e = \{v, w\}$ and let Let $\pi_v = \prod_{r \in N(v) - \{w\}} x_{vr}$ and let $\pi_w = \prod_{r \in N(w) - \{v\}} x_{wr}$. Now if $\pi_v \leq \pi_w$, then adjust x by "pushing" all x_{vw} units from v to w 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 $e = (v, w)$ and $x_e = 1$ if $e = (w, v)$, we then have $f(x') \geq f(x)$, as $f(x') - f(x) = x_{vw}(\pi_w - \pi_v) \geq 0$ If, instead, $\pi_v > \pi_w$, then let $x' = x$ except $x_e = 1$ if $e = (v, w)$ and $x_e = 0$ if $e = (w, v)$; this again ensures that $f(x') \geq f(x)$. Repeating this process for each edge of G gives an assignment x'' with $f(x'') \geq f(x)$ and each $x''_e \in \{0, 1\}$. Consider the set of vertices $W = \{v \in V : x''_{vw} = 1 \text{ for all } w \in N(v)\}$. Then W is an independent set of vertices and $|W| = f(x'') \geq f(x)$.

Now, we were given a graph G and mapped it to a polynomial f. Suppose we have a (μ, α)-approximate solution $\tilde{x}$ to an optimal solution x^* of f. Then, since $f^* = W^*$, we know $f(\tilde{x}) \geq |W^*|/\mu - \alpha$, where $|W^*$ is the size of a maximal independent set in G. Mapping $\tilde{x}$ to an independent set according to the construction above gives an independent set $\tilde{W}$ of cardinality at least $f(\tilde{x})$. For this set $\tilde{W}$, we have that $|\tilde{W}| \geq |W^*|/\mu - \alpha$, as required. ■

Acknowledgments

The author gratefully acknowledges Peter Hammer for his visit to Dartmouth College, in which he described the result in [1] used here. Thanks also to Mihir Bellare, Shafi Goldwasser, Silvio Micali

and Steven Vavasis (via Mihir), for sharing their knowledge and insights with me.

References

[1] Ch. 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, vol. 19, 83–97, 1984. (North Holland Mathematics Studies, 95.)

[2] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. Manuscript, 1991.

[3] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

[4] A. Panconesi and D. Ranjan, Quantifiers and approximation. *Proceedings of the Twenty Second Annual ACM Symposium on the Theory of Computing*, 446–456, 1990.

[5] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Proceedings of the Twentieth Annual ACM Symposium on the Theory of Computing*, 229–234, 1988.

[6] S. Sahni. Computationally related problems. *SIAM J. of Computing*, Vol. 3, No. 4, 262–279, 1974.