A STRONGLY POLYNOMIAL ALGORITHM FOR APPROXIMATE CONVEX OPTIMIZATION WITH COMBINATORIAL CONSTRAINTS AND RESOURCE ALLOCATION

by

Eric J. Friedman

friedman@IEOR.Berkeley.EDU

Memorandum No. UCB/ERL/IGCT M92/6

15 January 1992
(Revised 12 October 1992)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A STRONGLY POLYNOMIAL ALGORITHM FOR APPROXIMATE CONVEX OPTIMIZATION WITH COMBINATORIAL CONSTRAINTS AND RESOURCE ALLOCATION

by

Eric J. Friedman

friedman@IEOR.Berkeley.EDU

Memorandum No. UCB/ERL/IGCT M92/6

15 January 1992
(Revised 12 October 1992)

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

We present a strongly polynomial algorithm for approximate convex programming with combinatorial constraints, which for any ν always finds a solution with relative error less than ν and is polynomial in $1/\nu$. Additionally, this algorithm allows us to calculate approximate solutions for convex objective functions over any convex feasible region for which the solution for linear objective functions can be solved in strongly polynomial time. This is then specialized to obtain an efficient algorithm for solving generalized multi-resource allocation problems with externalities.

KEYWORDS: Strongly Polynomial Algorithms, Convex Programming, Resource Allocation, Ellipsoid Algorithms.

This research has been funded by National Science Foundation Grant IRI-8902813.

1 Introduction

The complexity of algorithms for linear and combinatorial problems has been well studied. However, results for nonlinear problems have been sparse. In [12] Nemirovsky and Yudin develop a theory of complexity for nonlinear optimization and construct several polynomial algorithms for approximate convex optimization. Based on these ideas we develop algorithms for convex problems utilizing algorithms for solving the related linear problems. Specifically, in this paper we describe a strongly polynomial algorithm for convex programming with combinatorial constraints. This is related to the result of Tardos [14] that linear programming with combinatorial constraints is strongly polynomial. Since the exact optimizer of a convex program may have an infinite encoding length, we define a $\nu - accurate$ solution in a scale invariant manner, and show that our algorithm has a running time that depends linearly on $\log 1/\nu$.

This algorithm can be applied to any convex optimization problem in which the linear problem can be solved in strongly polynomial time. This shows that strong polynomiality of any linear problem over a convex set implies strong polynomiality of the related convex program. In general we can show the following. Assume that we are given a linear optimization problem $\min\{f(x) = c^t x \mid x \in G\}$, where G is a bounded convex set, that can be solved in time $T(G)$ independent of c. Then the extension to the case where $f(x)$ is an arbitrary convex function can be solved to accuracy ν in

$$\mathcal{O}(2nT(G) + n^2(n^2 + T_s(G))\log(n/\nu))$$

elementary operations, where $T_s(G)$ is the time required to find a separating hyperplane for G. (Note $T_s(G) \leq T(G)$.) Thus we see that convex optimization is 'not much harder' than linear optimization.

Finally, we specialize this to obtain an algorithm for allocating r resources with externalities and polymatroidal constraints which runs in

$$\mathcal{O}((rn)^4 \log(rn/\nu))$$

elementary operations.

2 Oracles

Since arbitrary convex functions are permitted we assume that there exists an oracle through which an algorithm can learn about the objective function. Given a point $x \in G$, the oracle allows us to compute both $U(x)$ and a supporting hyperplane to $U(x)$ at x in $\mathcal{O}(n)$ elementary operations. In the case where $U(x)$ is differentiable the problem of computing a supporting hyperplane is equivalent to that of finding $\nabla U(x)$. For the remainder of this paper we assume that $U(x)$ is differentiable. The extension to the nondifferentiable case is straightforward.

3 $\nu - Accuracy$

Consider the convex program

$$\min f(x) \;\; s.t. \;\; x \in G,$$

where f is a convex function and G is a bounded convex set in $\mathcal{R}^n$. As the optimal solution need not be rational (or even algebraic), it cannot be computed by any finite algorithm. Thus it is necessary to consider approximate solutions.

It is important that this method of approximation be scale invariant; that is, the accuracy of a solution should not change under affine transformations of f or $\mathcal{R}^n$. For example, let x_{min} denote the optimal solution and x_{out} the solution found by our algorithm. Define a measure of approximation to be $|f(x_{out}) - f(x_{min})|$. Then an algorithm that guarantees any finite accuracy in a fixed number of steps necessarily guarantees arbitrary accuracy in the same number of steps, since we can rescale $f(x) \rightarrow af(x)$. If we solve $af(x)$ to accuracy ϵ then we have solved $f(x)$ to accuracy ϵ/a for a arbitrarily large. A similar argument can be made if we measure accuracy in terms of coordinates. Thus it is important to normalize the accuracy in some manner.

We consider two natural scale-invariant error measures. The first is the distance from the approximate solution to the actual solution in terms of coordinates, such as

$$e(x) = \frac{|x_{out} - x_{min}|}{\rho(G)},$$

where $\rho(G)$ is the radius of G. The second measures the distance in terms of the solution value such as

$$\nu(x) = \frac{|f(x_{out}) - f_{min}|}{|f_{min} - f_{min}|},$$

where f_{min} is the minimum and f_{max} the maximum of $f(x)$ over P. Note that in both cases the measure is invariant under affine transformations.

The first measure $e(x)$ is related to the one used in [7, 6], where polynomial results are obtained for convex separable functions. These results can be traced to the existence of a proximity result between exact solutions and solutions restricted to a lattice. However, these results are not true for nonseparable functions. In fact it is easy to show that no polynomial (or even exponential) algorithm can guarantee accuracy in coordinates for general nonseparable functions.

The second measure $\nu(x)$ is very natural when the goal of the optimization truly is to maximize some function. The goal of an algorithm in this case is to do as well as possible when measured in terms of the objective value. This measure also provides a good framework for studying the complexity issues in nonlinear optimization.

Thus, following Nemirovsky and Yudin [12], we define the error of a proposed solution x to be $\nu(x)$. An algorithm is defined to be $\nu - accurate$ if it finds a feasible solution x_{out} satisfying $\nu(x_{out}) \leq \nu$ for any problem. We therefore consider an algorithm to be polynomial if it is polynomial in $1/\nu$.

Finally, we note that if $f(x)$ is given as a polynomial of degree at most d then our algorithm is truly polynomial (but not strongly polynomial) in the encoding length of the problem when the accuracy is measured in terms of the absolute error $|f(x_{out}) - f_{min}|$, since in this case we can explicitly bound both the maximum and minimum of $f(x)$ over G in a polynomial of the encoding length.

4 Strong Polynomiality and Tardos' Algorithm

The concept of strong polynomiality is especially natural in the context of nonlinear optimization. Following [5] we say that an algorithm is strongly polynomial if it is:

- natural – it uses a polynomial number of elementary mathematical operations.
- polynomial – when applied to rational input, the algorithm is polynomial in the size of the input.

The first condition implies that the algorithm is polynomial in the number of real arithmetic operations, independent of the size of the numbers in the input. The second implies that when the input is rational the running time of the algorithm on a Turing machine is polynomial. When the first condition is satisfied the second condition reduces to showing that all intermediate results occurring during the calculation have size[1] that is polynomially bounded in the size of the input.

We use the term *natural* because it seems natural for an algorithm that solves a continuous optimization over $\mathcal{R}^n$ to be written in terms of elementary operations and not depend on the size of the input. Thus in this sense all known polynomial algorithms for general linear programming are *unnatural* because their running time depends on the size of the input. For example, Khachian's algorithm [10] uses the ellipsoid method to test for feasibility of a polytope. However, the number of iterations and the size of the numbers required depend on the encoding length of the polytope.

In [14], Tardos constructs a strongly polynomial algorithm for solving $\min\{c^t x \mid Ax \leq b\}$ assuming that A is combinatorial. A matrix A is combinatorial if its encoding length is polynomially bounded in its dimension. Examples include network matrices and any matrix for which all elements are bounded.

Since Tardos' algorithm will be used as a subroutine in our algorithm, we denote the time required to solve a combinatorial linear program $T(n, m, \Delta)$, where A is an $n \times m$ matrix whose largest subdeterminant is bounded by Δ.

5 The Algorithm

We consider the convex program

$$\min\{f(x) \mid Ax \leq b\},$$

where $f(x)$ is a differentiable convex function, A a combinatorial matrix, and b a rational vector. We define the feasible region $G = \{x \mid Ax \leq b\}$, and assume that G is bounded[2].

[1] We define the size of a number to be the number of bits required to represent it. For example if x is an integer then $size(x) = \lceil \log_2(x+1) \rceil + 1$. We will also use the floating point representation. Other representations can be useful as well. For example, the concept of *height* can be used to define linear programming over subrings of the reals as in [1].

[2] If G is unbounded then the solution of the convex program could be arbitrarily large or unbounded. If we assume that it is bounded and add the size of the solution to our measure of the problem size then we could still get results similar to those with a bounded feasible region.

If we assume that G is a body (i.e. has full dimension), then a direct application of the ellipsoid algorithm would give us a polynomial algorithm, but not a strongly polynomial one. However, in the case where G is not a body the ellipsoid algorithm would not work because the running time of the algorithm depends on the volume ratio between G and the initial bounding ellipsoid. This ratio is zero when G is not a body and depends on the encoding length of A and b when G is a body.

The main feature of our method is the construction of a bounding ellipsoid which has a volume ratio of

$$\mathcal{O}(n^{3n/2}),$$

independent of b, when projected onto the linear subspace containing G. The ellipsoid algorithm is then applied to the problem in the linear subspace to find the minimum.

By construction, our algorithm is polynomial in the number of exact arithmetic operations. However, it requires some care to show that it is polynomial when given rational input.

The algorithm is:

I. Construct an ellipsoid E and a linear subspace S such that $G \subseteq E \subset S$ and

$$n^{3n/2}|G|_S \geq |E|_S,$$

where $|\cdot|_S$ is the natural Euclidean volume in S.

A) Construct a linear subspace S and a parallelepiped P such that $G \subseteq P \subset S$ and

$$n^n|G|_S \geq |P|_S.$$

B) Construct an ellipsoid $E \subset S$ such that $P \subseteq E$ and

$$n^{n/2}|P|_S \geq |E|_S.$$

II. Run the ellipsoid method in the linear subspace S using E as the initial ellipsoid.

We will show that the first step requires $\mathcal{O}(nT(n,m,\Delta))$ operations, where $T(n,m,\Delta)$ is the running time of a strongly polynomial algorithm for LP. The second step requires $\mathcal{O}(n^2\log(n/\nu))$ iterations, where each iteration takes $\mathcal{O}(n^2+nm)$ operations[3].

Thus the total running time for the algorithm is:

$$\mathcal{O}(nT(n,m,\Delta)+n^3(n+m)\log(n/\nu)),$$

which is independent of $f(x)$ and b.

[3]Note that we cannot assume that $m \leq n$ as in linear programming.

5.1 Constructing a Bounding Ellipsoid

In this subsection we describe step I of our algorithm in detail. Let G be a convex body and

$$Q(c) = \text{argmin}\{c^\dagger x \mid x \in G\}.$$

Then step I of our algorithm computes a set of vectors which span the smallest linear subspace containing G and finds an ellipsoid E in this linear subspace satisfying $G \subseteq E$. This ellipsoid will have at most $n^{3n/2}$ times more volume than G in S. This applies to a convex body described by combinatorial linear constraints by using Tardos' algorithm. It also applies to any convex set for which the linear problem is solvable in strongly polynomial time and the encoding length of $Q(c)$ is independent of c. We will assume that such an algorithm exists for our problem.

The ellipsoid is computed in two steps. Step IA iteratively bounds the polytope between pairs of hyperplanes. Each pair defines an upper and lower bound for the polytope in a specific direction. The key idea used is that every new pair is chosen to be orthogonal to the currently known directions of the polytope. This procedure guarantees that null directions of the polytope will be found.

Step IA:

1. Set $\hat{S} = \emptyset$.
2. For $i = 1$ to n
   (a) Compute a c_i such that $\forall j < i \quad c_i^\dagger r_j = 0$. (This can be done by Gaussian elimination and choosing the free components of c_i to be 1.)
   (b) Compute $x_i^\pm = Q(\pm c_i)$.
   (c) Set $\alpha_i^\pm = c_i^\dagger x_i^\pm$.
   (d) If $\alpha_i^- \neq \alpha_i^+$:
      i. then $r_i = x_i^+ - x_i^-$, and $\hat{S} = \hat{S} \cup \{r_i\}$,
      ii. otherwise let $r_i = sgn(c_i)$, where $sgn(x_1, x_2, \ldots, x_n) = (sgn(x_1), sgn(x_2), \ldots, sgn(x_n))$.
3. Reorder so that $\hat{S} = \{r_1, r_2, \ldots, r_k\}$. (Reorder the $\alpha_i^\pm$'s and the $x_i^\pm$'s the similarly.)
4. Let $P = \{x \mid \forall i,\ \alpha_i^- \leq c_i^\dagger x \leq \alpha_i^+\}$.
5. Set $S = Lin(r_1, r_2, \ldots, r_k)$, where $Lin(r_1, r_2, \ldots, r_k) = \{x \mid x = \sum_{j=1}^k a_j r_j,\ \ a_i \in \mathcal{R}\}$.

Note that step (2a) just computes a vector orthogonal to S in a manner that is obviously strongly polynomial. Also (2dii) computes a simple vector r_i such that $r_i^\dagger c_i > 0$.

Theorem 1 *The above algorithm (step IA) constructs a parallelepiped P and a subspace S such that*

$$n^n |G|_S \geq n! |G|_S \geq |P|_S.$$

Proof: Define $S_i = Lin(r_1, r_2, \ldots, r_i)$, $P_i = P \cap S_i$ and $G_i = G \cap S_i$. (These are the projections of S, P, and G onto the current S at step i.) We will prove by induction that

$$\frac{|G_j|_{S_j}}{|P_j|_{S_j}} \geq \frac{|G_{j-1}|_{S_{j-1}}}{j|P_{j-1}|_{S_{j-1}}}.$$

This will suffice as $|P_1| = |G_1|$. Note that

$$|P_j|_{S_j} = |P_{j-1}|_{S_{j-1}} \frac{\alpha_j^+ - \alpha_j^-}{\sqrt{c_j^\dagger c_j}},$$

which follows from the elementary formula volume(parallelepiped) = base * height.

Now let $G'_j = ConvexHull(G_{j-1}, x_i^\pm)$. By convexity $G'_i \subseteq G_i$. Since G'_i is a (generalized) cone we compute

$$|G'_i|_{S_i} = |P_{j-1}|_{S_{j-1}} \frac{\alpha_j^+ - \alpha_j^-}{j\sqrt{c_j^\dagger c_j}}$$

by using volume(cone in j dimensions) = base * height/j.

Thus we see that G'_i satisfies the inductive hypothesis, and noting $|G'_i|_{S_i} \leq |G_i|_{S_i}$ completes the proof. □

Now we transform into coordinates aligned with the r_i's, which makes P into a cube. We then enclose the cube within a sphere.

Step IB:

1. Let $C^\dagger = [c_1, c_2, \ldots, c_k]$,

$$\hat{C} = Diag(\frac{2}{\alpha_1^+ - \alpha_1^-}, \frac{2}{\alpha_2^+ - \alpha_2^-}, \ldots, \frac{2}{\alpha_k^+ - \alpha_k^-})C,$$

and

$$d = (\frac{\alpha_1^+ + \alpha_1^-}{2}, \frac{\alpha_2^+ + \alpha_2^-}{2}, \ldots, \frac{\alpha_k^+ + \alpha_k^-}{2}).$$

2. Define $y = \hat{C}x + d$, yielding $P_S = \{y | \forall i \leq k, \ -1 \leq y_i \leq 1\}$.

3. Define $M_E = Diag(1/n, 1/n, \ldots, 1/n)$ and $E = \{y | \ y^\dagger M_E y \leq 1\}$. Thus E is the sphere of radius $\sqrt{n}$ centered at the origin.

4. Now transform back into the original coordinates x. The volume ratio is preserved by this transformation so we have found the desired ellipsoid, since linear transformations of a sphere create an ellipsoid.

Theorem 2 *The ellipsoid E constructed in step* IB *satisfies*

$$|E|_S \leq n^{n/2}|P|_S.$$

Proof: In y coordinates P_S is a box of volume 2^n. The sphere of radius $\sqrt{n}$ has volume less than $(2\sqrt{n})^n$ because it is contained in a box with sides of length $2\sqrt{n}$. □

Combining steps IA and IB gives us the desired bound.

Theorem 3 *The ellipsoid constructed in step* IB *satisfies*

$$|E|_S \leq n^{3n/2}|G|_S.$$

The number of operations in the above algorithm is polynomial in the number of elementary operations and the sizes of numbers that occur during the running of the algorithm are polynomially bounded in the size of the input, thus proving strong polynomiality. This is because the only operations we use involve solving for $Q(c)$, and Gaussian elimination. The size of the $x_i^{\pm}$'s is polynomially bounded in L and independent of c_i's. From Gaussian elimination on matrices containing r_i we obtain c_i's that are polynomially bounded.

5.2 The Ellipsoid Method

The final step (II) of our algorithm is the application of the ellipsoid method in the new (y) coordinates to our problem. The ellipsoid method was developed by Nemirovsky and Yudin [12] based on an idea of Levin [11]. It is most well known from its use by Khachian [10] to prove the polynomiality of linear programming. However, its original purpose was for convex programming. While it is useful theoretically for LP it does not seem to be of practical value; however, it may actually be useful for convex programming.[4]

In this section we describe the ellipsoid method for our problem. First we need some facts about ellipsoids. (See, e.g. [13]). An ellipsoid is the set

$$E(M,m) = \{x|\ (x-m)^t M^{-1}(x-m) \leq 1\}.$$

Given a vector v, define the set

$$E(M,m,v) = E(M,m) \cap \{x|\ v^t x \geq 0\} \subseteq E(M',m'),$$

where

$$|E(M',m')| \leq |E(M,m)|2^{-1/2(n+1)},$$

and

$$M' = \frac{n^2}{n^2-1}[M - \frac{2}{n+1}\frac{(Mc)(Mc)^t}{c^t Mc}]$$

$$m' = m - \frac{1}{n+1}\frac{Mc}{\sqrt{c^t Mc}}.$$

The ellipsoid algorithm runs as follows:

Step II— The Ellipsoid Algorithm:

[4]Ecker and Kupferschmid have done numerical studies [2, 3] showing that the ellipsoid method is very robust compared to other standard methods. It is also quite efficient for finding low accuracy solutions and is very simple to implement (i.e. 61 lines of FORTRAN code.)

1. Given $G \subseteq E(M_0, m_0)$. Let $N = \left\lceil 4n^2 \log \frac{n}{\nu} \right\rceil$, $v_0 = 0$, $v_{out} = \infty$, and $x_{out} = 0$.
2. For $i = 1$ to N
   (a) If $m_{i-1} \in G$
      i. then
         A. Let $c_i = \nabla f(m_{i-1})$.
         B. If $f(m_{i-1} \leq v_{out})$ then $v_{out} = f(m_{i-1})$ and $x_{out} = m_{i-1}$.
      ii. otherwise let c_{i-1} define a separating hyperplane to G. (i.e. let c_{i-1} be a row of A associated with one of the violated inequalities of $Am_{i-1} \leq b$.)
   (b) Calculate M_i, m_i such that $E(M_{i-1}, m_{i-1}, c'_{i-1}) \subseteq E(M_i, m_i)$ as above.
3. Output x_{out} and v_{out}.

Theorem 4 *The ellipsoid algorithm terminates with x_{out} such that $\nu(x_{out}) \leq \nu$ in $N = \left\lceil 4n^2 \log \frac{n}{\nu} \right\rceil$ iterations.*

The proof can be found in [12]. Briefly each iteration of the algorithm reduces the volume of the current ellipsoid by $2^{-1/2(n+1)}$. The current ellipsoid is guaranteed to contain all points better than the current best (v_{out}) by step (4). The algorithm continues until the current ellipsoid has ν^n less volume than the original feasible region. From this we can derive the number of iterations required. This volume reduction guarantees that there exist $z, y \in G$, not in the final ellipsoid, which satisfy $y = (1 - \nu)x_{min} + \nu z$. By the construction of the ellipsoid we know that $v_{min} \leq f(y)$, and by convexity we have $f(y) \leq (1-\nu)f(x_{min}) + \nu f(z)$. Noting that $f(z) \leq f_{max}$ we obtain the desired bound on accuracy.

Note that the square root in the algorithm could cause the encoding length of the numbers to become very large or even infinite. We can avoid this problem by using a standard argument as in [13]. Represent a number as a P-bit binary integer multiplied by an integral power of 2. Thus if x is a real number and $\hat{x}$ our representation then $|\hat{x} - x| \leq 2^{-P}$. Now choose a slightly larger ellipsoid at each iteration. This will slow our convergence slightly, doubling the number of iterations. However by choosing P sufficiently large we can guarantee the required accuracy. Thus all gradients can be computed in floating point arithmetic.

At first it may seem odd that ellipsoid algorithm for our problem is strongly polynomial while it is only polynomial when applied to linear programming. This can be reconciled by noting that the ellipsoid algorithm for linear programming must reduce the volume of the initial ellipsoid by $\mathcal{O}(2^{-2nL})$. In our case we chose our initial ellipsoid so that the algorithm only needs to reduce it by $\mathcal{O}((\nu/n)^n)$ which is independent of L.

6 Multi-Resource Allocation with Externalities

The allocation of resources to consumers or processors is an important problem in economics [8] and distributed computing [4]. In the case of a single resource without externalities there has been much progress in the development of efficient algorithms [6, 9]. In [6] Hochbaum designs an algorithm that solves the single resource allocation problem efficiently, where the

accuracy is defined in terms of coordinates. However, this measure of accuracy is not useful for extending her results to the multi-resource allocation problem, as we noted earlier. Thus, there have been no complexity results, that we know of, for the multi-resource allocation problem with externalities.

Here we show how these problems can be solved in strongly polynomial time. Then we show how to solve the generalized problem with polymatroidal constraints.

The multi-resource allocation problem with externalities can be written as:

$$\max \sum_{i=1}^{n} U_i(x_i^1, x_i^2, \ldots, x_i^r, \lambda^1, \lambda^2, \ldots, \lambda^r)$$

s.t.

$$\forall i \in \{1,2,\ldots,n\},\ j \in \{1,2,\ldots,r\}\ 0 \leq x_i^j,\ 0 \leq \lambda^j \leq b_j$$

$$\forall j \in \{1,2,\ldots,r\}\ \sum_{i=1}^{n} x_i^j \leq \lambda^j$$

where $U_i(x_i^1, x_i^2, \ldots, x_i^r, \lambda^1, \lambda^2, \ldots, \lambda^r)$ is concave, non-decreasing in x_i^j, and non-increasing in λ^j.

We can transform this into a standard form by defining $y_i^j = x_i^j/b_j$ and setting $y_0^j = 1-\lambda^j$. This gives a problem that is easier to analyze:

$$\max \sum_{i=1}^{n} U_i(y_i^1, y_i^2, \ldots, y_i^r, 1-y_0^1, 1-y_0^2, \ldots, 1-y_0^r)$$

s.t.

$$\forall i \in \{0,1,\ldots,n\},\ j \in \{1,2,\ldots,r\}\ 0 \leq y_i^j$$

$$\forall j \in \{1,2,\ldots,r\}\ \sum_{i=0}^{n} y_i^j \leq 1.$$

This is just a special case of convex programming with combinatorial constraints, where we can explicitly calculate an initial ellipsoid, thus removing the need to use Tardos' algorithm and speeding up the procedure.

Note that the feasible region is a body G with $|G| = 2^{-rn}$. It is contained in the unit square centered at $x_c = \{1/rn, 1/rn, \ldots, 1/rn\}$. Now the initial ellipsoid is the sphere of radius $\sqrt{rn}$ centered at x_c. This sphere S satisfies

$$2^{2rn}(rn)^{rn/2}|G| \geq |S|.$$

Applying the ellipsoid method to this leads to an algorithm that takes

$$\mathcal{O}((rn)^4 \log \frac{rn}{\nu})$$

elementary operations which is strongly polynomial.

The generalized multi-resource allocation problem can also be solved within this framework. The generalized multi-allocation problem in standard form is:

$$\max \sum_{i=1}^{n} U_i(y_i^1, y_i^2, \ldots, y_i^r, y_0^1, y_0^2, \ldots, y_0^r)$$

s.t.

$$\forall i \in \{0,1,\ldots,n\},\ j \in \{1,2,\ldots,r\}\ l_i^j \leq y_i^j$$

$$\forall j \in \{1,2,\ldots,r\}\ \sum_{i=0}^{n} y_i^j \leq 1$$

$$\forall j \in \{1,2,\ldots,r\}\ A \in H\ \sum_{i \in E} y_i^j \leq \phi_j(A),$$

where l_i^j's are given lower bounds, and $\phi_j(\cdot)$ is a submodular function. Let $H \subset 2^{\{1,2,\ldots,n\}}$ where $H = \{A_1, A_2, \ldots, A_m\}$. We allow H to be one of three possibilities. Either H is a partition of $\{1,2,\ldots,n\}$, or it consists of nested subsets $S_1 \subset S_2 \ldots \subset S_m \subset \{1,2,\ldots,n\}$, or it is tree constrained. Tree constrained implies that the sets S_i can be placed in a tree where a set is a child of a node only if it is a subset of the set at that node. These classes are discussed in detail in [6].

The importance of polymatroidal constraints is that they allow the separable problem to be solved by the greedy algorithm. (However, the nonseparable problem is *not* solvable by the greedy algorithm in general.)

Since the problem with linear constraints is separable we can apply the greedy algorithm very efficiently. We can solve the linear problem in $\mathcal{O}((rn)^2)$ strongly polynomial time. In this case $T(G) = \mathcal{O}((rn)^2)$, so the running time of the algorithm is

$$\mathcal{O}((rn)^4 \log \frac{rn}{\nu}),$$

as in the simple case.

Note that the form of the objective function does not affect the running time of the algorithm. Thus this algorithm will work for *any* convex function with any of the three types of polymatroidal constraints.

7 Conclusions

We have shown that $\nu - accurate$ convex programming over linear constraints is not much harder than linear programming. This can be seen as a generalization of Hochbaum and Shanthikumar's result that "Convex Separable Optimization is Not Much Harder than Linear Optimization"[7].

We believe that the application of ideas from computational complexity should apply generally to convex optimization, and that the generalization of more results from linear programming to convex programming with linear (and perhaps convex) constraints would be invaluable in better understanding the roots of computational complexity.

8 Acknowledgements

I would like to thank Ross Baldick, Dorit Hochbaum, and Adam Landsberg, Henrik Lenstra, and Shmuel Oren for useful conversations.

References

[1] I. Adler and P. A. Beling. Polynomial algorithms for linear programming over the algebraic numbers. Preliminary Draft, 1991.

[2] J.G. Ecker and J. Kupfershmid. The ellipsoid algorithm for nonlinear programming. *Mathematical Programming*, 27:83–106, 1983.

[3] J.G. Ecker and J. Kupfershmid. A computational comparison of the ellipsoid algortihm with several nonlinear programming algorithms. *SIAM J. Control and Optimization*, 23:657–74, 1985.

[4] D. W. Ferguson, C. Nikolau, , and Y. Yemini. Microeconomic algorithms for load balancing in distributed computer systems. *Proceedings of the 8th International Conference on Distributed Computing Systems*, 1988.

[5] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin, 1988.

[6] D. S. Hochbaum. Lower and upper bounds for the allocation problem and other nonlinear optimization problems. Manuscript, School of Business Administration and Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720. September 1989, Revised December 1990.

[7] D. S. Hochbaum and J. G. Shanthikumar. Convex separable optimization is not much harder than linear optimization. *Journal of the ACM*, 37(4):843–862, October 1990.

[8] L. Hurwicz. The design of mechanisms for resource allocation. *The American Economic Review*, pages 1–30, May 1973.

[9] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches.* MIT Press, 1988.

[10] L.G. Khachian. A polynomial algorithm for linear programming. *Soviet Math. Doklady*, 20:191–4, 1979.

[11] A. Yu Levin. On an algorithm for minimizing convex functions. *Soviet Maths*, 1:286–90, 1965.

[12] A.S. Nemirovsky and D.B. Yudin. *Problem Complexity and Method Efficiency in Optimization.* Wiley, New York, 1983.

[13] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[14] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, March–April 1986.