CHAPTER 6

On the complexity of linear programming

Nimrod Megiddo

Abstract: This is a partial survey of results on the complexity of the linear programming problem since the ellipsoid method. The main topics are polynomial and strongly polynomial algorithms, probabilistic analysis of simplex algorithms, and recent interior point methods.

1 Introduction

Our purpose here is to survey theoretical developments in linear programming, starting from the ellipsoid method, mainly from the viewpoint of computational complexity.[1] The survey does not attempt to be complete and naturally reflects the author's perspective, which may differ from the viewpoints of others.

Linear programming is perhaps the most successful discipline of Operations Research. The standard form of the linear programming problem is to maximize a linear function $c^T x$ $(c, x \in R^n)$ over all vectors x such that $Ax = b$ and $x \geq 0$. We denote such a problem by (A, b, c). Currently, the main tool for solving the linear programming problem in practice is the class of simplex algorithms proposed and developed by Dantzig [43]. However, applications of nonlinear programming methods, inspired by Karmarkar's work [79], may also become practical tools for certain classes of linear programming problems. Complexity-based questions about linear programming and related parameters of polyhedra (see, e.g., [66]) have been raised since the 1950s, before the field of computational complexity

The author thanks Jeffrey Lagarias, David Shmoys, and Robert Wilson for helpful comments.

[1] The topic of linear programming has been interesting to economists not only due to its applicability to practical economic systems but also because of the many economic insights provided by the theory of linear programming. However, in this chapter we discuss linear programming from a point of view of theoretical computer science.

Reprinted from
Advances in Economic Theory, Fifth World Congress
Edited by Truman F. Bewley
© Cambridge University Press 1987
Printed in the United States of America

started to develop. The practical performance of the simplex algorithms has always seemed surprisingly good. In particular, the number of iterations seemed polynomial and even linear in the dimensions of problems being solved. Exponential examples were constructed only in the early 1970s, starting with the work of Klee and Minty [85].

The field of computational complexity developed rapidly during the 1970s. The question of the complexity of linear programming was formalized in a new and more precise sense. A specific question remained open for several years until finally solved by Khachiyan [83, 84] in 1979. He showed that linear programming, as a problem of recognizing a formal language, is in the class $\mathcal{P}$; that is, it can be solved in polynomial time relative to the length of the binary encoding of the input. Khachiyan's result was also applied in a very elegant way to problems of combinatorial optimization by Grotschel, Lovasz, and Schrijver [64].

Khachiyan's result was widely misinterpreted for a while, mainly because of popular articles claiming that a substitute had been found for the simplex algorithm. However, it was not long before it became clear that the ellipsoid algorithm (that was used by Khachiyan to prove his nice result) is not useful for solving linear programming problems in practice. This was quite a disappointment to those who believed complexity theory could be relied on in practice. It became clear that some exponential algorithms (namely, variants of the simplex method) were very efficient in practice, while a polynomial algorithm for the same problem (the ellipsoid method) was very inefficient. It was only natural that interest in the field started to increase in two directions: (i) analyzing the behavior of the simplex method from a different viewpoint, and (ii) searching for other methods.

A breakthrough in the analysis of the simplex method was independently made by Borgwardt [26, 27, 28] and Smale [144, 145]. In Section 4 we review further work in this field.

At least from a theoretical viewpoint, it is interesting to settle the computational complexity of linear programming under different models of computation. Khachiyan's result relies on the so-called logarithmic-cost model [9]. It is still an open question whether a system of linear inequalities can be solved in a number of arithmetic operations that is polynomially bounded by the dimensions of the system, independently of the magnitudes of the coefficients. In [107] such an algorithm is given for systems with at most two variables per inequality (whereas the general case can be reduced to at most three variables per inequality). Questions about the model are discussed in Section 2. So far, only partial results are known. Eva Tardos [147, 148] obtained a general linear programming algorithm whose number of elementary operations is independent of the magnitudes of coefficients in the objective-function and the right-hand-side vectors, but depends on the coefficients in the matrix A. This implies that many combinatorial optimization problems, including the minimum cost-flow problem, can be solved in "strongly" polynomial time. This work of Tardos is reviewed in Section 5.

Theoretical research on algorithms in recent years focused on the direction of estimating the asymptotic worst-case complexity of problems in $\mathcal{P}$. For instance, knowing that a certain problem can be solved in polynomial time, it is of interest to find exact (asymptotic) upper and lower bounds on the time it should take any algorithm to solve the problem in the worst case. There has been much research done in this direction in the related field of computational geometry [95]. For example, it is known that the complexity of computing the convex hull of a set of n points in the plane is $\theta(n \log n)$.[2] A surprising result was obtained in [109]; namely, for any fixed d, linear programming problems with d variables and n constraints can be solved in $O(n)$ time as n tends to infinity. This extended previous independent work ([48, 108]) that showed the same for $d \leq 3$. This area of research is reviewed in Section 6.

Linear programming was again in the news in the fall of 1984: Karmarker developed a new polynomial-time algorithm for linear programming that is in fact practical. It improves the upper bound on the complexity of linear programming, again under the logarithmic-cost model. Karmarkar has claimed very strongly [80] that his algorithm is superior by far to the simplex method. However, at the time of this writing there is no publicly available evidence to support clear superiority. This work inspired renewed interest in applying methods of nonlinear programming to the linear programming problem. The algorithm is reviewed in Section 7. Recent related work is reviewed in Section 8.

Regarding the worst-case complexity of the simplex method, it is only known that specific variants of the method require exponential time in the worst case. It is a major open question whether every variant[3] requires exponential time in the worst case. The complexity of the randomized simplex algorithm is not known. Results have been obtained about the worst-case complexity of certain variants of the simplex method when applied to special classes of linear programming problems. Of special interest are assignment problems and the more general minimum cost-flow problem. This topic is discussed in Section 9.

We conclude the paper with some discussion in Section 10 on theory versus practice.

[2] This means that there is an algorithm that finds the convex hull in $C_1 n \log n$ time and there is a constant C_2 such that any algorithm for the convex hull requires in the worst case at least $C_2 n \log n$ time.

[3] We have not defined what a variant is. Clearly, the effort per pivot step must be restricted because otherwise any algorithm can be stated as a variant of the simplex method.

2 On complexity and models of computation

Complexity of computations is a central area of research in theoretical computer science. One of the common measures of complexity of an algorithm is the *asymptotic worst-case running time.* Here we discuss this concept on a rather informal level. The interested reader may refer to [9] for exact definitions. Any measure of complexity must be defined relative to a specific model of computation. The meaning of a statement like "Problem P has complexity $O(f(n))$" is roughly as follows. First, P is understood to be a class of instances with a well-defined measure of length of an instance. Second, there is an "algorithm" and there is a constant C such that any instance of P of length n is solved by the algorithm within $Cf(n)$ time units. The notion of an algorithm, and the amount of time it takes to execute the basic operations in the algorithm, must be defined. It is customary among theoretical computer scientists to think of a problem as tractable if it has polynomial time complexity, and intractable otherwise. The advantage of this approach is that the property of polynomial time is robust in the sense that (to a certain extent) it does not depend on the model of computation.

Practitioners usually have different points of view. First, they are not interested so much in the worst-case performance of an algorithm. They usually like to have an idea about the *distribution* of running times, although they usually do not have a definite idea about the distribution of instances.[4] Also, practitioners are less interested in asymptotic complexity. The advantage of the asymptotic approach is that it is more amenable to mathematical analysis. When a theoretician says he has improved an algorithm he usually means he has designed an algorithm of lower asymptotic worst-case time complexity. Practitioners may then wonder at what size of an instance the asymptotically better algorithm becomes more favorable (yet in the sense of the worst case). Moreover, practitioners are interested in factors such as space (also of interest to theoreticians), program length and simplicity, numerical characteristics, and versatility.

In the context of linear programming, the size of an instance can be defined in different ways. Usually the complexity is expressed in terms of the numbers of rows and columns of the system. Assuming exact computation on real machines, the cost of performing arithmetic operations must depend also on the sizes of numbers involved. However, in practice most problems are being solved in floating-point arithmetic and the cost

[4] It remains a challenge to theoreticians to come up with models for predicting the efficiency of algorithms in practice. Results that depend on an exact probability distribution of instances are often not satisfactory. Because the distribution is not known, analysis should be done about *classes* of distributions characterized by weakest possible assumptions (see [7] for further discussion of this issue).

is uniform (if numbers do not get out of range). On the other hand, the known polynomial algorithms for linear programming require in the worst case a number of steps that depends on the sizes of the coefficients. To relate theoretical complexity results to the practical performance of the algorithm, one needs to specify the precision under which a solution must be obtained. Matters are then complicated even further by the need to specify the measure of approximation to be used.

Another point related to linear programming is the sparsity of the matrix. Most of the theoretical work on linear programming measures the complexity relative to the dimensions or the length of the binary representation of the problem. Of course, sparse problems have shorter representations even if each zero coefficient is given explicitly. However, sparse systems are being solved much more efficiently in practice, essentially because the necessary linear algebraic steps are carried out more efficiently. For polynomial worst-case complexity, sparsity helps only because of a shorter binary representation. Theorists pay little attention to the cost of performing single iterations, whereas practitioners have a goal of making the cost of a single step as low as possible. The initial attitude of theoreticians is to look at the general problem, first ignoring any structure. When any complexity measure is expressed in terms of several parameters rather than just a single input size, it becomes more difficult to compare algorithms, because the result of the comparison may depend on the particular combination of parameter values.

The problem of linear programming has more than one level of abstraction. The original problem is usually stated and solved over the field of the reals. Actually, for the traditional model of complexity, as well as for most practical computations, the field of the rationals is the appropriate one. There also exist combinatorial abstractions of linear programming in the context of "oriented matroids" (see, e.g., [22]). This area will not be discussed here.

From an algebraic point of view, the problem can be posed relative to any "ordered field." An ordered field is a field where the nonzero elements are classified as positive (when an element a is positive we write $a > 0$) or negative ($a < 0$) subject to the following axioms: (i) If a and b are positive then so are $a+b$ and ab. (ii) If a nonzero element a is not positive then $-a$ is. It is obvious that the simplex algorithm solves the linear programming problem over any ordered field. Interesting observations on resolving degeneracy in real problems via the use of larger ordered fields, as well as solutions to asymptotic problems, are given in [77] and the references thereof. A more recent paper on this subject is [51].

The term "ordered field" is justified by the fact that positivity induces a total order: An element a precedes an element b ($a < b$) if $b - a > 0$. The relations $\leq$ and $\geq$ are defined in the natural way. Obvious consequences are as follows: (i) $1>0$; (ii) $a+1>a$; (iii) $a, b<0$ implies $a+b<0$ and $ab>0$. It thus follows that an ordered field *cannot be finite*. Moreover, it must contain the field of rationals. This containment is in the strong sense of ordered fields; that is, for any ordered field F there is an isomorphism of ordered fields from the ordered field of the rationals, Q, into F. Let F be any ordered field. Since F contains the field of rationals Q, we can define a mapping ρ from F into the extended real line $R^* = R \cup \{-\infty, \infty\}$ as follows. For any $a \in F$ let $\rho(a) = \operatorname{Sup}\{r \in Q: r<a\}$. (We use r to denote both a rational number and its corresponding element in F.) By $\rho(a) = -\infty$ (respectively, $\rho(a)=\infty$) we mean $a<r$ ($a>r$) for all rational r. It is easy to check that $\rho(a)>\rho(b)$ implies $a>b$ while $a>b$ implies $\rho(a) \geq \rho(b)$. The mapping ρ is not necessarily one-to-one; for example, F may contain "infinitesimal" elements - that is, elements a such that $0<a<r$ for every positive rational r. However, ρ is a "homomorphism" from the subfield $F_R = \{a \in F: \rho(a) \in R\}$; that is,

$$\rho(0)=0, \quad \rho(1)=1, \quad \rho(\alpha+\beta)=\rho(\alpha)+\rho(b) \quad \text{and} \quad \rho(ab)=\rho(a)\rho(b).$$

It follows that many useful theorems on real systems of inequalities can be generalized to abstract ordered fields, even when the proof relies on properties of the real numbers. For example, vector spaces over ordered fields are not necessarily normed but some useful properties related to norms extend to such spaces. The following proposition is one such example, which turns out to be related to Karmarkar's algorithm.

Proposition 2.1. *Let $u \in F^n$ be an n-dimensional vector over an ordered field F and let $H \subseteq F^n$ be a linear subspace. Suppose $v^* \in H$ is such that $(u-v^*)^T w = 0$ for every $w \in H$. Under these conditions, v^* minimizes the "squared distance" function $f(v) = (u-v)^T(u-v)$ for $v \in H$.*

Proof: Let $v \in H$ be any vector and denote $w = v - v^*$. Then

$$(u-v)^T(u-v) = (u-v^*-w)^T(u-v^*-w) = (u-v^*)^T(u-v^*) + w^T w.$$

The claim follows from the fact that $w^T w \geq 0$. ■

At least from a theoretical viewpoint, the following (loosely stated) question is important for understanding linear programming: Can linear inequalities be decided in a polynomial number of field operations over any ordered field?[5] More precisely, is there a polynomial $p(m, n)$ and an

[5] The same question can be asked with respect to real closed fields. A field F is *real closed* if it satisfies: (i) $\sum \alpha_i^2 \neq 0$ unless $\alpha_i = 0$ for all i, and (ii) all the nontrivial algebraic extensions of F do not satisfy (i). It is known that any real closed field can be ordered uniquely.

algorithm[6] that decides any system $Ax \leq b$ of dimension $m \times n$ after performing no more than $p(m,n)$ field and other machine operations? The simplex method is valid over any ordered field but several variants of it are known to run in exponential time in the worst case.

There is still a problem with the formulation of the question in the preceding paragraph. It seems that multiplication and division are quite powerful operations. Complicated computational tasks can be performed by generating large numbers. For example, given a set S of n field elements $a_1, \ldots, a_n$, one can generate in $O(n)$ additions and multiplications an element A of the form $A = \sum_{j=1}^{2^n} \alpha_j M^j$, where $\alpha_1, \ldots, \alpha_{2^n}$ are all the sums over subsets of S, and $M > \sum a_i$ is another element. Operations on elements of the form of A can simulate simultaneous operations on all the subsets of S. This suggests that a polynomial number of multiplications involving large elements may suffice for solving real problems that require exponential time under a model of logarithmic cost. So it seems reasonable to include in a definition of a "strongly" polynomial algorithm [147, 148] also the requirement that when the input consists of rational numbers, the sizes of the numbers occurring in the computation are bounded by a polynomial in the input size. In other words, an algorithm is strongly polynomial if (first) it is polynomial and (second) the number of arithmetic operations is polynomial in the dimensions of the problem. Over an abstract field, because the size of an element is not well defined, we might require that the height[7] [110] of multiplications and divisions be bounded by $p(\log m, \log n)$ for some polynomial p. The simplex algorithm satisfies this requirement about heights, since each iteration can start from inputs and only has to solve a system of linear equations. Systems of linear equations can be solved over any ordered field in a polynomial number of field operations with small height of multiplications [53, 30, 134].

3 The ellipsoid method

The question that was settled by Khachiyan [83, 84] can be stated as follows. Given a system of linear inequalities $Ax \leq b$ ($A \in R^{m \times n}$, $b \in R^m$) with integral coefficients,[8] let L denote the total number of bits in a binary representation of the coefficients a_{ij} and b_i. The question is whether there

[6] We still assume here that an "oracle" can be used to perform the field operations – that is, the arithmetic operations and comparison.

[7] The height $h(a)$ of an input element a is 1. In general, for any elements generated by the algorithm, $h(a+b) = \max(h(a), h(b))$ and $h(ab) = h(a)h(b)$; subtraction and division are handled like multiplications and divisions, respectively.

[8] The extension to rational coefficients is trivial.

exists an algorithm for deciding feasibility (i.e., the existence of a solution to the system) that takes no more than $p(L)$ bit operations, where p is a polynomial. The answer is yes, and the proof relies on an algorithm by Yudin and Nemirovsky [161].

Before sketching the algorithm, we find it helpful to describe an abstract scheme of establishing a time bound for an algorithm that generates a sequence of objects O_i $(i=0,1,2,\dots)$ (e.g., points, intervals, ellipsoids). Suppose g is a criterion function from the set of objects into the positive reals. Assume the algorithm makes progress in terms of g, with a lower bound on the rate of improvement, so that there is a $\delta<1$ such that for every i, $g(O_{i+1})\leq\delta g(O_i)$. Furthermore, suppose the algorithm terminates when $g(O_i)\leq\epsilon$ for some prespecified $\epsilon>0$. Obviously, an upper bound on the number of steps of the algorithm can be stated in terms of $g(O_o)$, δ, and ϵ. A bound N on the number of steps is derived from the equation $g(O_o)\delta^N=\epsilon$, so that $N=(\log g(O_o)-\log\epsilon)/(-\log\delta)$. Thus, if $\log g(O_o)$, $-\log\epsilon$, and $-1/(\log\delta)$ are each bounded by a polynomial function of the input size then the algorithm terminates in polynomial time.

The objects generated by the "ellipsoid" algorithm are n-dimensional ellipsoids. A suitable criterion function maps n-dimensional ellipsoids to their n-dimensional volumes. It is easier to state the algorithm for the problem of deciding feasibility of a set of strict inequalities whose solution set is guaranteed to be bounded: $Ax<b$, $|x_j|<\lfloor 2^L/n\rfloor$. For proving a polynomial upper bound, this form is equivalent to the original one. The algorithm generates a sequence of ellipsoids with the following properties: (i) Each ellipsoid contains all the *basic* feasible solutions of the system $Ax\leq b$. (ii) The factor by which the volume of the current ellipsoid decreases during a single step satisfies $\delta<2^{-1/(2(n+1))}$. Interestingly, this factor depends only on the smaller dimension n and does not depend on the number of constraints m or the numerical values of the coefficients. This allows for many elegant applications of the method to combinatorial optimization problems with exponentially many constraints [64]. The efffect of L on the time bound is via the volumes of the first and last ellipsoids.

The first ellipsoid is a ball of radius $n2^L$ centered at the origin. Thus its volume is not greater than $(2n^2 2^{2L})^n$. This ball is guaranteed to contain all the basic feasible solutions of $Ax\leq b$ if there are any. A step of the algorithm starts by checking whether the center of the current ellipsoid solves the system $Ax<b$. If so, then the algorithm terminates with the center as solution; otherwise, a (volume-wise) smaller ellipsoid is constructed. It can be proved that if the volume of the current ellipsoid is less than $\epsilon=2^{-(n+1)L}$ then there can be no basic feasible solutions and hence

no feasible solutions at all. Polynomiality of the algorithm follows from the preceding discussion.

The original algorithm relies on the square-root operation. It is known that finite precision suffices [146, 155]. It follows that the ellipsoid algorithm can run over any ordered field; that is, the iterations can be performed. However, if infinitesimals are involved then the algorithm may not solve the problem. For example, the algorithm may never reach a point where infeasibility is evident. Similarly, if the volume of the initial ellipsoid is infinite then all the following ellipsoids have infinite volume. Thus, the ellipsoid algorithm does not solve the problem over general ordered fields. A good survey of the ellipsoid method is given in [24].

An algorithm by Levin [99] was shown by Yamnitsky and Levin [159] to run in polynomial time, using a similar analysis. Levin's algorithm works with simplices rather than ellipsoids. Hence, it can be run over any ordered field but does not solve the problem in general.

4 Probabilistic analyses of simplex algorithms

The number of pivot steps performed by simplex algorithms in practice is widely considered surprisingly small. The "surprise" is due to observations as follows. Every simplex algorithm visits bases of the underlying system of linear equations. It is known how to guarantee that no basis is visited more than once - that is, how to avoid "cycling." Because there are a finite number of bases, it follows that any noncycling simplex algorithm is finite. Moreover, this number is bounded (from above) by an exponential function of the dimensions of the system. It is not known whether there exists a simplex[9] algorithm whose (worst-case) number of steps is bounded by a polynomial function of the dimensions. Furthermore, for some simplex algorithms there have been constructed examples on which these algorithms perform an exponential number of steps. The first such example was designed by Klee and Minty [85]. Related examples were later designed for several simplex algorithms [78, 13, 119, 120, 63, 163]. It is customary to say that "the simplex algorithm is exponential," meaning that it requires in the worst case an exponential number of steps. However, this qualification is unfair because it has not been proven that for every variant of the method (in any precise sense) there exists an exponential example. There are in fact variants whose worst-case analysis seems veiy difficult. On the other hand, even for the (worst-case) exponential variants, practitioners report numbers of steps that are far smaller than

[9] We have stated earlier in this chapter that the concept of a simplex algorithm or variant is not well defined.

the numbers suggested by the worst-case complexity. The wide gap calls for mathematical analysis to explain it.

Many simplex algorithms can be described as generating simple paths in the one-dimensional skeleton of some polytope. In other words, in one step they move from a vertex[10] to an adjacent one, visiting no vertex more than once. This suggests that such variants would perform better on polytopes with fewer vertices. Thus, first attempts to explain the good performance of simplex algorithms in practice were based on estimating numbers of vertices of random polytopes. We will not review the results here. For analysis in linear programming it is interesting to consider random polytopes generated by picking half-spaces at random. Under most of the models investigated, it turned out that in a fixed dimension the expected number of vertices increased slowly with the number of half-spaces. However, the expected number of vertices was exponential in both the dimension and the number of half-spaces. Obviously, an approach based on estimating numbers of vertices of polytopes cannot demonstrate that a simplex algorithm is more efficient than an algorithm that enumerates all the vertices of a polytope. Such an approach ignores the particular way a specific algorithm chooses the next vertex in the path. Also, many simplex algorithms generate paths on which some linear function is monotone. The essence of the intuitive argument for the efficiency of monotone simplex algorithms is as follows. We think that if v^1 and v^2 are adjacent vertices then, on the average, there are many vertices w where the value of the objective function is between the values at v^1 and v^2. Thus, such vertices w are "skipped" when the algorithm moves from v^1 to v^2.

The works of Borgwardt [26, 27, 28] and Smale [144, 145] constituted a breakthrough in the probabilistic analysis of simplex algorithms. In these works, for the first time, the particular choice of the algorithm was taken into account. Both identified closely related variants of the simplex method, where it was possible to write closed-form formulas for the expected number of steps of the variant, in terms of the probability distribution over the set of problem instances. It is surprisingly easy to obtain formulas for these variants. The algorithms that can be analyzed in this way are based on parameterization in some form: The algorithm follows solutions to a parameterized family of problems. The number of steps corresponds to the number of certain cones met by a certain straight line. The hard part in this research is the analysis of the resulting formula.

[10] In the presence of degeneracy this statement is not accurate. The method actually visits *bases* of a linear system and may change the basis many times before moving to a new vertex. Furthermore, every linear programming problem can be reduced to a problem where all the algorithm has to do is move from the second-best vertex to the best one (but through many changes of the basis).

Borgwardt and Smale worked with closely related probabilistic models. However, their models differ in a very significant way. Under Borgwardt's model only feasible problems with known feasible solutions are generated. Under Smale's model most of the problems are either infeasible or unbounded, depending on whether we consider the primal or the dual problem. On the other hand, the questions that arise in the analyses of the formulas are very closely related. Borgwardt succeeded in proving that under his model the expected number of steps was bounded by a polynomial in both the dimension and the number of half-spaces. Smale proved that under his model if one of the parameters (i.e., either the dimension or the number of half-spaces excluding the nonnegativity constraints) was fixed then the expected number of steps was bounded by a polynomial in the logarithm of the other parameter. Megiddo [115] showed that in this case the expected number of steps was bounded by a function of the smaller parameter; that is, for any fixed m the poly-log function of n can be replaced by a constant. However, it is still an open question whether this constant depends polynomially on the smaller parameter. It is reasonable to conjecture that the ideas and methods used by Borgwardt would help in proving polynomiality here as well. Blair [20] obtained a result close to Smale's for a large set of algorithms. Blair's result is based on estimating the expected number of undominated columns. This is closely related to the expected number of vertices of random polyhedra.

Haimovich [67] and Adler [1] obtained a result related to the analysis of a similar simplex variant. They consider a model that was previously looked at in [31, 118, 2, 3, 104, 4]. In this model all the hyperplanes are fixed in a nondegenerate manner. A cell of the induced partition of space is picked at random. Suppose any two vectors c and c' are given. Consider the family of vectors of the form $c_t = tc + c'$, where t varies over the reals. For each cell of the partition consider a path of optimal solutions determined by minimizing the linear function $c_t^T x$ over the cell. Haimovich and Adler prove that the average length of such a path (i.e., the number of vertices on the path) is linear. However, this result applies neither directly to a specific algorithm for linear programming nor even to Phase II of an algorithm, since the "auxiliary" direction c' depends on the vertex of the cell that is produced in Phase I. Thus, it is hard to justify an assumption that the direction is independent of the cell. However, this result is surprisingly good and provides much insight into average lengths of parametric paths on random polyhedra.

Adler, Karp, and Shamir [5] show that for constraint-by-constraint algorithms (this does not include the self-dual method considered by Smale), the probabilistic model of "sign-invariance" (i.e., where the probability distribution is invariant under changes in the directions of inequalities) implies that the expected number of steps is bounded by a function of the smaller dimension. However, in that paper all the upper bounds are still exponential. The result in [115], of a bound depending only on the smaller dimension, is not covered by the result in [5].

Polynomial upper bounds depending only on the smaller dimension were obtained in [149, 7, 6, 8]. In these papers the analytic difficulties encountered in [144, 145, 115] are avoided. The mathematical analysis is considerably simplified if the starting point of the algorithm is changed from $(1, \ldots, 1)$ to $(\epsilon, \epsilon^2, \epsilon^3, \ldots)$. Furthermore, the assumptions of the probabilistic model can be relaxed. An upper bound of $O(\min(m^2, n^2))$ can be proved under assumptions of sign-invariance and nondegeneracy. A quadratic upper bound is proved in [7] for any assignment of powers of ϵ to rows and columns of the system. Moreover, [7] also proves a quadratic lower bound under a stronger model, which implies that even under the weaker model one cannot prove a subquadratic upper bound. It should also be noted that the algorithms in these papers are all special cases of the self-dual algorithm with various starting points (see [113]). It is not clear what is the best starting point for the average performance of the self-dual method for linear programming. The question of the starting point in the context of the general linear complementarity problem is tackled in [70, 140, 114, 149]. It is known that $(1, \ldots, 1)$ is worst for the average case of the general problem. However, the author conjectures that for the linear programming problem this point is best among all the nonnegative starting points. The model under which the $O(\min(m^2, n^2))$ result is obtained may be criticized for allowing unboundedness (or infeasibility) with increasing probability. However, at least for the case $m = n$ (i.e., for a system with n linear inequalities in n nonnegative variables) it implies that the conditional expectation of the number of steps, given that the problem is feasible and bounded, is only $O(m^{2.5})$, while the conditional expectation of the number of vertices is exponential.

5 Strongly polynomial algorithms

People have been interested in the computational complexity of linear programming since the development of the simplex method. The fundamental question concerned the dependence of the number of pivot steps on the dimensions m and n of the problem. The number of pivot steps is a natural measure of complexity for simplex algorithms, provided the effort per step is reasonable. The search for polynomial algorithms for linear programming was intensified in the 1970s because of two developments: (i) the discovery that several simplex algorithms required exponential numbers of pivot steps in the worst case, and (ii) the growing interest among computer scientists in polynomial-time algorithms. It seems that the concept of a polynomial algorithm was understood by the mathematical programming community differently. The mathematical programming community is a little more inclined toward practice. In practice, linear programming problems are usually solved in floating-point arithmetic and the numbers are limited to some finite range. The cost of performing arithmetic operations is constant provided the numbers stay in this range. The space occupied by a number is bounded provided it is in this range. Hence, for the mathematical programming community, the natural question was the existence of an algorithm that required only a polynomial number $p(m, n)$ of arithmetic operations. In theoretical computer science the common approach is that machines work with bits, and hence the size of the input has to be measured in bits and the running time in bit operations. This means that the size of the input for a linear programming problem depends not only on m and n but also on the coefficients themselves. The difference between the two approaches led to a surprise within the mathematical programming community when Khachiyan's result was announced. The presence of the parameter L (the number of bits in the binary representation of the input) in the polynomial upper bound was not expected. Of course, this parameter must appear in any bit-operations estimate of the running time, because it takes unbounded time to carry out arithmetic operations on unbounded numbers. However, it is not clear that the number of arithmetic operations should depend on the magnitudes of the numbers. The distinction between the two approaches can also be explained as a difference between models of computation. Theoretical computer scientists like to work with complexity classes that are robust against changes in the model of computation. The class P of problems solvable in polynomial time is a good example. However, the complexity of a problem is actually the complexity of a formal language that encodes it, and hence depends on the encoding. The question is: What is a reasonable model for discussing problems with numerical inputs? Suppose the input in a problem is a sequence of n numbers and the required output is another number. Is it reasonable to use the number n as the input size? The answer depends on the problem. Many computational number-theoretic problems have only one or two numbers as input (e.g., primality testing, greatest common divisor), and the difficulty of the problem depends on the magnitudes of the numbers. It is hard to imagine that such problems can be solved in a bounded number of arithmetic operations. On the other hand, it seems reasonable to measure the input to a problem of solving linear equations by the number of coefficients in the system. Moreover, the number of arithmetic operations needed for solving linear equations is bounded by a polynomial in the number of coefficients.

Naturally, there is interest in settling the complexity of linear programming under this alternative model. Results are known only for special classes of linear programming problems. It is known that systems of linear inequalities with at most two variables per inequality can be decided in strongly polynomial time [107]; that is, the number of arithmetic operations is bounded by a polynomial in m and n. In fact, a linear function with at most two nonzero coefficients can be optimized subject to such inequalities. The algorithm is also polynomial in the usual sense.

Desire for a strongly polynomial algorithm for the minimum cost-flow problem was expressed by Edmonds and Karp [54] in the same paper that proposed the first polynomial algorithm to the problem. A strongly polynomial algorithm for the minimum cost-flow problem was recently developed by Tardos [147], who also obtained [148] an algorithm for the general linear programming problem. In the rest of this section we review this nice result on linear programming in strongly polynomial time (with respect to the objective function and the right-hand-side vector) recently obtained by Tardos. Work related to Tardos's algorithm was done by Orlin [127] and Fujishige [58]. Another extension is the work by Frank and Tardos [56] mentioned at the end of this section.

The main result of Tardos [148] can roughly be described as follows. Linear programming problems (A, b, c) with rational coefficients can be solved in a number of arithmetic operations bounded by a polynomial in m, n and the number of bits in the binary representation of the matrix A (regardless of the magnitudes of coefficients in b and c). For any matrix of rational numbers, the length of the binary representation of the matrix is called the *size* of the matrix. Thus, the time-complexity of Tardos's algorithm depends only on the size of A and not on the sizes of b and c, whereas in the other polynomial-time algorithms [83, 84, 79] the complexity does depend on these sizes. However, the claim about polynomial dependence on the size of A is possible due to the existence of polynomial-time algorithms for linear programming, but Tardos's result should be appreciated independently of these results. It provides strongly polynomial algorithms for numerous problems of combinatorial optimization without relying on polynomial algorithms for the general linear programming problem.

The number of elementary operations used in Tardos's algorithm is independent of large numbers occurring in the objective function and the right-hand-side vectors.[11] It is linear programming duality that allows one to deal with these vectors in a symmetric way. We first consider the objective function vector. The essence of the method is as follows. Given

[11] If one assumes that the cost of an operation depends on the magnitudes of the operands then the performance is affected by the existence of large numbers anywhere.

any objective function vector, one can replace it by another with "moderate" coefficients. The optimal solution relative to the revised objective function identifies at least one constraint that is tight at the optimum relative to the original objective function. Thus, instead of solving a problem with arbitrarily large coefficients, one can solve a sequence of problems (each identifying at least one additional tight constraint) with moderate coefficients in the objective function. Each of these problems may still have arbitrarily large coefficients in the right-hand-side vector. However, with the help of duality, the same trick can be applied to the subproblems.

The main ideas of the algorithm can be explained in the case where one needs to maximize a linear function c^Tx over a nonempty polyhedral set P, given in the form $\{x: Ax=b, x\geq 0\}$, where the dimensions of the system are $m\times n$. The algorithm is stated for an integer matrix A. Obviously, any problem with rational A can be handled too. A modified direction $\bar{c}$ is computed as follows. First, any linear combination of rows of A can be added to c^T without changing the set of optimal solutions. Choosing an appropriate combination, we can replace c by a vector $c'=c-A^Ty$ such that $Ac'=0$. The case $c'=0$ is trivial. Otherwise, c' is replaced by $c''=(n^2\Delta/\|c'\|_\infty)c'$, where Δ is an upper bound on the absolute value of any minor of A. The size of Δ can be bounded by a polynomial in the size of A. Obviously, $\|c''\|_\infty=n^2\Delta$ and the set of c''-optimal solutions still equals the set of the c-optimal ones. Now, c'' is replaced by a vector $\bar{c}$ consisting of the integral values of the coordinates of c''. The set of $\bar{c}$-optimal solutions is no longer the same as the set of c-optimal solutions. However, important information can be obtained by solving the dual problem with $\bar{c}$. Suppose y is an optimal solution to this dual problem – that is, the problem of minimizing b^Ty subject to $A^Ty\geq\bar{c}$ and $y\geq 0$. It can be proved that, for any column A_j of A such that $y^TA_j\geq c_j''+n\Delta$, necessarily $x_j=0$ for any optimal solution x of the original primal problem (maximize c^Tx subject to $Ax=b$ and $x\geq 0$). The interesting fact is that there is at least one such column. Such a column can be dropped from the system and the same process repeated with a smaller system. After at most n steps we identify the set of all j's such that $x_j=0$ for any optimal solution x.

Each step in the above procedure requires the solution of the dual problem whose right-hand-side vector is $\bar{c}$ (consisting of moderate coefficients, i.e., integers of absolute values not greater than $n^2\Delta$), but whose objective function vector b may still consist of large numbers. The idea is to solve this problem with the same basic trick but note that feasibility is not guaranteed. Also, the description so far applies only to problems that are guaranteed to be feasible.

The feasibility of any system $Ax\leq b$ can be decided in polynomial time in the size of A as follows. An objective function c_0 is introduced: $c_0=\Sigma\,(\Delta+1)^ia_i$, where a_i is the ith row of A. Consider the problem of maximizing c_o^Tx subject to $Ax \leq b$. The dual problem of the latter is to minimize b^Ty subject to $A^Ty = c_o$ and $y \geq 0$. We first note that this dual problem is feasible (e.g., $y_i = (\Delta + 1)^i$ is feasible). Moreover, the size of its right-hand-side vector c_o is bounded by a polynomial in the size of the matrix A. The basic trick applies here (consider maximizing $-b^Ty$), and any polynomial algorithm can be used to solve the subproblems because the right-hand-side vector is of moderate size. If the dual problem is unbounded then the primal is infeasible; otherwise, the primal here is feasible since the dual is. In the latter case the basic algorithm identifies all the (dual) constraints that are tight at the (dual) optimum. Any polynomial linear programming algorithm can be used to select a dual optimal solution; it takes polynomial time in the size of A only, because the dual objective b is not required once the optimal face is known and the size of c_o is polynomially bounded by the size of A anyway. Once a dual optimal solution $\bar{y}$ is found, a primal feasible solution $\bar{x}$ is computed by solving the system $a_i\bar{x} = b_i$ for all i such that $\bar{y}_i > 0$. The validity of the last step follows from the particular choice of c_o and the complementary slackness theorem.

The complete algorithm for linear programming now works as follows. Given the problem of maximizing c^Tx subject to $Ax = b$ and $x \geq 0$, the algorithm first checks the feasibility of the system $\{Ax = b, x \geq 0\}$. If feasible then the feasibility of the dual system $A^Ty \geq c$ is checked. If the dual is also feasible then the basic algorithm is applied to the primal problem. The subproblems here are solved as follows. First, feasibility of the dual system $A^Ty \geq \bar{c}$ is checked (this may be done by any polynomial linear programming algorithm because $\bar{c}$ is of moderate size). If this dual is feasible then Tardos's algorithm can be called recursively to find an optimal solution to the problem of minimizing b^Ty subject to $A^Ty = \bar{c}$ and $y \geq 0$. Specific optimal primal and dual solutions can then be found by the procedure described above for detecting feasibility.

In the way stated in [148], Tardos's algorithm is not well-defined for a general matrix A of real numbers, even though the vectors c and b may be real. In fact, the coordinates of b and c can be elements of any ordered field. Normalization eliminates all the infinite elements and then infinitesimal elements are rounded down to zero. Recall that the basic trick of the algorithm is to replace c by a vector whose coordinates are integers with absolute values not greater than $n^2\Delta$, where Δ is determined by the matrix A. When A has integral entries, Δ has to be an upper bound on the absolute value of any minor of A. The natural generalization for real matrices is to choose Δ as an upper bound on the absolute value of the *ratio* of any two nonzero minors of A. The given problem is then reduced to a polynomial number (in m and n) of problems with modified objective functions. More precisely, the vector c is replaced by $\bar{c}$, where

$$\bar{c}_j = \lfloor (n^2\Delta/\|c'\|_\infty)c'_j \rfloor \quad \text{and} \quad c' = c - A^T y$$

so that $Ac' = 0$. This raises two questions. First, it is not known whether such an upper bound Δ can be found in a polynomial number (in m and n) of elementary operations on real numbers. It is easy to compute an upper bound on the absolute value of any minor: namely, $n!\Lambda^n$, where Λ is the maximum absolute value of entries of A. However, it is not known how to compute a positive lower bound for the absolute value of any nonzero minor of A in a polynomial number of real number operations. Even if an appropriate Δ could be found in polynomial time, it is not clear that the optimization problem with the modified objective function vector $\bar{c}$ is easier than the original one.

It is interesting to examine the existence of a Tardos-type result over general ordered fields. However, we have to be careful in phrasing the question. Consider the following question: Can we solve the problem (A, b, c) using a number of field operations that depends only on A? The answer is (trivially) yes, because the problem can be solved by enumerating all the bases. This procedure has an upper bound on the number of field operations that depends only on m and n. The more general interesting question is of course the existence of polynomial bounds (in m and n) but this is the fundamental question rather than what we would call a Tardos-type question. A more reasonable question can be asked as follows. Does there exist a function g that assigns to any field element a a positive integer $g(a)$, and does there exist an algorithm that solves the problem in $g(A)p(m, n)$ field operations, where $g(A)$ is the maximum of $g(a)$ over entries of A, and p is a polynomial?

Frank and Tardos [56] extend Tardos's algorithm to certain combinatorial optimization problems with exponentially many constraints. This extension applies the simultaneous approximation algorithm of [97] for computing an equivalent objective function vector with moderate coefficients. Most of the combinatorial optimization problems can be formulated as linear programming problems (possibly with exponentially many constraints), where the coefficients in the matrix are 0, 1, or -1. If the number of constraints is polynomial in the size of the original problem (e.g., the minimum cost-flow problem) then Tardos's method provides a strongly polynomial algorithm. Linear programming problems with an exponential number of constraints (given implicitly) can be solved in polynomial time by the ellipsoid algorithm, provided a polynomial-time algorithm is available for proving feasibility of a given point or else providing a violated constraint. The Frank and Tardos algorithm makes such polynomial algorithms strongly polynomial by modifying the objective function into an equivalent one whose size is polynomial. It replaces a given objective function vector c by a vector c^* of integers of polynomial size such that, for any vector $u \in \{-1,0,1\}^n$, $u^T c \geq 0$ if and only if $u^T c^* \geq 0$. The components of c may be elements of any ordered field. The computation of c^* relies on the simultaneous approximation algorithm.

6 Linear programming in fixed dimension

Questions about the asymptotic worst-case time complexities of various problems became very popular during the 1970s. In particular, questions of computational geometry (see [95] for a survey) attracted much attention. It has been known that the complexity of finding the extreme points of the convex hull of a set of n points in the plane is $\theta(n \log n)$. This suggested that the complexity of the two-variable linear programming problem was the same. However, it was shown in [108, 48] that this problem could be solved in $O(n)$ time. The idea is quite simple. For any two constraints of the form $y \geq a_i x + b_i$ $(i=1,2)$, there is a value x' (namely, where $a_1 x' + b_1 = a_2 x' + b_2$, assuming a nondegenerate case) so that on each side of the line $\{x = x'\}$ one of the constraints dominates the other (in the weak sense). Thus, if at some point it becomes known that the search for a solution may be restricted (say) to the half-plane $\{x \geq x'\}$, then at that point one of the constraints may be eliminated. The algorithm is based on finding good values of x that allow for the elimination of "large" sets of constraints. Suppose, for simplicity, we have $n = 4k$ constraints of the form $y \geq ax + b$ and they are paired arbitrarily; that is, we have arranged them in $2k$ disjoint pairs. Let x_m denote the *median* of the intersection values $x'_1, \dots, x'_{2k}$ of these pairs (assuming a nondegenerate case). It is easy to decide in $O(n)$ time whether the search may be restricted to $\{x \leq x_m\}$ or to $\{x \geq x_m\}$. This decision then allows for the elimination of k constraints, namely, one dominated constraint from each pair of constraints that do not intersect on the side of the line $\{x = x_m\}$ to which the search may be restricted. It follows that in Cn time (where C is some constant) about a quarter of the set of constraints can be eliminated. This implies that by repeating the process about $\log_{4/3} n$ times the set of constraints is exhausted and eventually two critical constraints are identified. The total time is of the form $Cn(1 + (3/4) + (3/4)^2 + (3/4)^3 + \cdots)$ and is hence linear in n. The linear upper bound relies on the result that the median can be found in $O(n)$ time (see [9]) and this algorithm can be considered as an extension of the linear-time median-finding algorithm.

Three-variable linear programming problems can also be solved in $O(n)$ time [108, 48] but the algorithm is more involved and the resulting constant is larger. It turns out that only about one-sixteenth of the constraints can be eliminated during one iteration. The elimination is based on knowledge that the search may be restricted to a certain quadrant of the plane, determined in two median-finding steps. The generalization of this result to dimensions higher than three [109] is not trivial. The algorithm works according to the same principle of eliminating dominated constraints. In a fixed dimension (i.e., when the number of variables is fixed and the number of constraints is n), a fixed fraction of the set of remaining constraints is eliminated. However, the fraction tends to zero very fast with the number of variables. In the construction of [109] the fraction is doubly exponential, yielding an $O(2^{2^d}n)$ algorithm for d variables and n constraints. Improved constructions that provide a bound of $O(3^{d^2}n)$ were suggested in [50, 36].

Linear-time algorithms using similar methods also appear in [49, 112, 116, 164]. Although the basic algorithm for two variables is extremely fast (using "approximate" medians obtained by sampling, rather than exact medians), the algorithm in [109] is clearly not a serious tool for solving general linear programming problems in practice. However, considering asymptotic worst-case complexity, it is optimal for any fixed number of variables.

7 Karmarkar's algorithm and related work

Karmarkar [79] developed another polynomial-time algorithm for linear programming that generates a sequence of interior points converging to an optimal solution. Interior point methods are usually used in nonlinear programming [55, 61] and have been proposed for linear programming as well (see [60]), but no one before Karmarkar had identified a method that provably ran in polynomial time. To analyze the time-complexity of an algorithm, one usually needs a measure of the progress (a "merit function") that the algorithm makes toward the solution. However, it is usually not easy to identify good measures of the progress. The naive measures - like the amount of improvement in the objective function or the distance to the solution - usually do not suffice for proving nice bounds, simply because it is impossible to prove a good lower bound on the progress *per iteration*. Of course, efficient methods do not have to make much progress in every single step and thus, to prove their efficiency, one needs to understand how they make good progress in the long run.

A common technique in nonlinear programming is the *barrier-function* method. The goal is to solve constrained optimization problems with the tools of unconstrained optimization. A sequence of points is generated, starting in the interior of the feasible domain. The objective function is replaced by another function that is coherent with the original objective, and yet has a "barrier" ingredient that prevents the algorithm from getting out of the feasible domain. To specify a barrier-function method, one needs to construct such a function and also specify an unconstrained optimization algorithm.

Karmarkar's algorithm is easy to state with respect to the following form of the linear programming problem: *minimize* $c^T x$ *subject to* (i) $Ax = 0$, (ii) $\sum_k x_k = 1$, and (iii) $x \geq 0$, where A is of dimension $m \times n$; it is also assumed that the problem has an optimal solution and that the optimum of $c^T x$ equals zero. A vector x is called *interior* if it satisfies (i) and (ii) and if $x > 0$. It is assumed that an interior point x^0 is available in the beginning. Any linear programming problem can be reduced to such a form. This is accomplished if the program is formulated so that the primal and dual problems are combined. The algorithm can be stated with the following barrier function:

$$F(x) = \frac{c^T x}{(\prod_k x_k)^{1/n}},$$

which is well-defined for interior points x. Obviously, the value of $F(x)$ tends to infinity when x tends (from the interior) to a boundary point z such that $c^T z > 0$. Also, unless $c^T x$ is constant over the feasible region, $F(x) > 0$ for every interior point x. On the other hand, if x tends from the interior *along any straight line* to an optimal point x^* on the boundary then $F(x)$ tends to zero; this is true because $c^T x$ approaches zero faster than $(\prod_k x_k)^{1/n}$, given that $\sum_k x_k = 1$. It follows that while $F(x)$ is sought to be minimized over the interior of the feasible domain, a minimum of $c^T x$ over the feasible domain is approached. However, we have not yet explained how an optimum is actually reached in polynomial time.

The algorithm (to be described below) generates a sequence $x^0, x^1, x^2, \ldots$ of interior points along which the function $F(x)$ decreases monotonically. Moreover, the rate of decrease is provably good enough to imply the desired result. Notice that the infinite process of optimizing $F(x)$ does not produce a boundary point. One needs to apply a stopping rule, and then either leave the problem with the current point as an approximate solution or run a procedure that produces an exact solution from the current approximate one. In practice, one can sometimes guess the optimal basis and then verify that it is optimal. In theory, if $F(x)$ is sufficiently close to zero then an optimal solution can be easily computed from x. Thus, in theory, the stopping rule is based on a sufficiently small value of $F(x)$.

The upper bound on the running time of Karmarkar's algorithm is polynomial in terms of the length of the binary representation of all the numbers involved. Let L denote this input length. An argument that appears in the analysis of the ellipsoid algorithm is as follows. For some constant κ, if x^1 and x^2 are basic solutions such that $c^Tx^1 \neq c^Tx^2$ then $|c^Tx^1 - c^Tx^2| > 2^{-\kappa L}$. This implies that if x is any feasible solution such that $0 < c^Tx < 2^{-\kappa L}$, then – by modifying x into a basic feasible solution while improving the objective function value (a standard procedure sometimes called "purification") – we obtain an optimal solution. Thus a valid stopping rule is $c^Tx \leq 2^{-\kappa L}$. Note also that $F(x) > c^Tx$ for any interior x, so $F(x) < 2^{-\kappa L}$ is also a valid stopping rule. The barrier function $F(x)$ is also a suitable criterion function in terms of the discussion in Section 3, with the iterates x^i playing the roles of the objects. The initial interior point x^0 can be chosen so that $\log F(x^0)$ is bounded by a polynomial in L. The parameter ϵ here is equal to $2^{-\kappa L}$ so that $-\log \epsilon$ is also polynomial in L. It remains to show (see below) that the rate of improvement δ is also such that $-1/(\log \delta)$ is polynomial in L. From a theoretical viewpoint, this characteristic is perhaps the most significant contribution of [79].

Let x^i be an interior point. We would like to move from x^i to a point x^{i+1} such that $F(x^{i+1})/(F(x^i)$ is sufficiently small. Karmarkar's step is related to the fact that the barrier function is, in a certain sense, invariant under a *projective rescaling transformation P* defined as follows. Given a point $a = x^i$, let $x' = P(x)$ be defined for any feasible x. First, for convenience of notation let $D = \text{diag}(a_k)$ denote a diagonal matrix of order n, where the diagonal entries are the components of the vector a. Also, let e denote an n-vector of 1's. The vector x' is equal to $(D^{-1}x)/(e^TD^{-1}x)$. The sense of invariance can be explained as follows. First note that $x = (Dx')/(e^TDx')$. Thus,

$$F(x) = \frac{c^TDx'}{e^TDx'} \Bigg/ \frac{(\prod_k a_k x_k')^{1/n}}{e^TDx'} = \frac{c^TDx'}{(\prod_k a_k x_k')^{1/n}}.$$

Denote $c' = Dc$ and $F'(x') = (c')^Tx'/(\prod_k x_k')^{1/n}$. It follows that for any two points x, y, if $x' = P(x)$ and $y' = P(y)$ then

$$\frac{F(x)}{F(y)} = \frac{F'(x')}{F'(y')}.$$

This equality allows us to work in the space transformed under P so that a (relative) improvement in F', while taking a step from $a' = P(a)$, is the same as that of the corresponding step from a.

The natural question at this point is what is the benefit from the transformation P. The answer is that it reveals a good way to take a step from a', as we explain below. Note that in fact $a_k' = 1/n$ for every k. Also, the linear subspace $\{x: Ax = 0\}$ is transformed under P into a subspace $\{x: A'x = 0\}$ while the simplex $\{x: e^Tx = 1, x \geq 0\}$ is mapped onto itself. The situation is now as follows. There are two balls, B_1 and B_2, both centered at the current point a', with the following properties: (i) B_1 is contained in the (transformed) feasible domain while the latter is contained in B_2. (ii) The ratio of the radius of B_2 to that of B_1 is $n-1$. The balls are simply the intersections of $\{x: A'x=0\}$ with the largest inscribed and the smallest circumscribing balls of the simplex $\{x: \sum x_j = 1, x \geq 0\}$. This construction is reminiscent of Lenstra's construction [98] for a polynomial integer programming algorithm in a fixed dimension. The step of Karmarkar's algorithm in the transformed space is made in the direction of the projection of the vector c' on the subspace $\{x: A'x=0\}$; its length is equal to one-quarter of the radius of B_1. The minimum of the linear function $(c')^T y$ over the (transformed) feasible domain is zero. It follows from the ball inclusion relations that the value of this function is multiplied during this step by a factor not greater than $1-1/4n$. However, we are interested in the improvement of the function F' so we must consider the change in $(\prod_k x_k)^{1/n}$ during the step. This is the reason a step of only one-quarter rather than the full radius of B_1 is taken. It is shown that this guarantees that the denominator of the (transformed) barrier function does not decrease too much. The result is that the value F' (and hence also of F) is multiplied during a step by a factor not greater than $e^{-1/8n}$. The latter quantity plays the role of δ in the discussion of Section 3. Polynomiality follows from the fact that $-1/(\log \delta)$ here equals $8n$. Thus, as in the ellipsoid algorithm, the factor of improvement depends (polynomially) only on the dimension[12] and not on the numerical values of the coefficients. Blair [21] showed that a step of length $(n-1)/(2n-3)$ of the radius of B_1 was possible with a corresponding $\delta = (4/(3\sqrt{e}))^{1/n}$.

Karmarkar's algorithm provides the strongest upper bound known concerning the worst-case complexity of the linear programming problem under the logarithmic cost model. The number of arithmetic operations in the worst case is bounded by[13] $O(n^{3.5}L^*)$ if the matrix operations are carried out in a specific way described in [79].

Interestingly, the iterative step of Karmarkar's algorithm can be adapted to work over any ordered field. Moreover, the guaranteed rate of improvement prevails. It is easy to get rid of the logarithms and roots in the analysis of the algorithm. Thus, for example, one can easily obtain the following from Proposition 2.1.

Proposition 7.1. *Under the conditions of Proposition 2.1, if* $H = \{x \in F^n: Ax = 0\}$, *where* $A \in F^{m \times n}$ *is of rank* m, *then the vector* $v^* = [I - A^T(AA^T)^{-1}A]u$ *minimizes the function* f *over* H.

[12] Interestingly, in the ellipsoid algorithm the factor depends on the smaller dimension of the system, and this allows for the nice applications to combinatorial optimization problems with exponentially many constraints. This is not so with Karmarkar's method.

[13] The parameter L^* is usually smaller than L and reflects a better estimate of the distances between basic solutions using minors of the matrix A.

We also have the following.

Proposition 7.2. Let $u=(n^{-1},\dots,n^{-1})\in F^n$ *and let* $H\subseteq F^n$ *be determined by* $\sum_{i=1}^{n-1}x_i=1$ *and* $x_n=0$. *Under these conditions, the vector* $v^*=((n-1)^{-1},\dots,(n-1)^{-1},0)\in F^n$ *minimizes the function* $f(v)=(u-v)^T(u-v)$ *for* $v\in H$.

The following proposition extends [21, Lemma 5], and justifies the choice of the step length so that the barrier component deterioration is controlled.

Proposition 7.3. Let F be an ordered field, $n\geq 2$ *a natural number, and let* $\{x_1,\dots,x_n,\alpha\}\subseteq F$ *be such that*

$$\sum_{k=1}^{n}\left(x_k-\frac{1}{n}\right)^2\leq\frac{\alpha}{n(n-1)}\quad \text{and}\quad \sum_{k=1}^{n}x_k=1.$$

Under these conditions,

$$\prod_{k=1}^{n}x_k\geq\frac{1-\alpha}{n}\left(\frac{1+\alpha/(n-1)}{n}\right)^{n-1}.$$

Using a criterion function that avoids *n*th roots, $f(x)=(c^Tx)^n/\prod_k x_k$, one can see that during each step the value of this function is multiplied by a factor that is less than and bounded away from 1. If the problem is solved over the rationals then this implies a polynomial time bound. However, despite a guaranteed rate of improvement, the algorithm does not solve the problem over general ordered fields because it is not guaranteed to reach infinitesimally close to an optimum. It can be used as an approximate method though, where the approximation is in terms of the objective function value. Recall that the simplex method works over any ordered field.

The computation of the projection of c' dominates the effort involved in a single iteration of the algorithm. The projection problem of a vector c' on a subspace $\{x: A'x=0\}$ is equivalent to the minimization of $\|A'x-c'\|$. This is known as a *least-squares* problem for which there are several techniques available. Further development in the numerical solution least-squares problems should improve the performance of Karmarkar's algorithm.

It should be noted that the practical implementations of Karmarkar's algorithm may be quite different from the theoretical algorithm. Deviations from the theoretical algorithm can be in several ways: (i) Problems may be solved not in the combined form of the primal and the dual. (ii) Instead of a fixed step length, the proposed direction of movement may be searched for a best step. Moreover, even the direction of movement itself may be selected after a search of a low-dimensional subspace. (iii) The theoretical stopping rule is not practical so heuristic rules may be developed for stopping. (iv) The least-squares subproblems may be solved only approximately (especially if iterative methods are selected for them), and there may be a certain degree of heuristicism involved in the choice of the level of precision. (v) Some effort may be invested in frequent checking for optimality of a guessed basis. (vi) Variables and constraints may sometimes be eliminated on the basis of both theoretical and heuristic arguments. The significance of an algorithm running in polynomial time diminishes in practice if it is no longer clear that the particular implementation is guaranteed to run in polynomial time. On the other hand, it is easy to run any algorithm in "parallel" to a polynomial-time algorithm, so that at least one of the algorithms is guaranteed to terminate in polynomial time.

Since the first publication of Karmarkar's algorithm there have been several papers written on variants of the method [11, 19, 32, 33, 59, 101, 111, 103, 151, 156, 160]. It is not clear what qualifies an algorithm as a variant of Karmarkar's algorithm. Karmarkar's algorithm is classified in [60] as one member of a family of methods that have been experimented with in the past and also recently at Stanford. The claim is that Karmarkar's algorithm can be reproduced as a projected Newton search method applied to a certain barrier function with a certain rule for updating the barrier parameter. Also, resemblances between Karmarkar's barrier function, Frisch's barrier function [57], and Huard's method of centers [71] are mentioned in [151]. Resemblance to Lawson's algorithm [94, 136, 38, 39] was pointed out by Walter Murray. Here, an l_∞-approximation problem is solved by a sequence of weighted l_2-approximation problems with a rule for updating the weights. However, the formal equivalence shown in [60] does not degrade the significance of Karmarkar's result. It seems reasonable to conjecture that under suitable weak conditions *any* interior-point method can be formulated as a barrier-function method. The interesting problem is to identify good methods. Karmarkar provided the first polynomial one; his algorithm is based on sensible principles that do not hold for the barrier-function algorithms in general. On the other hand, it is still not clear whether his algorithm is in fact better in practice than other variants of the barrier-function idea.

In Section 8 we discuss recent applications of nonlinear programming methods to linear programming that were inspired by Karmarkar's work. Besides the references mentioned above, further work related to Karmarkar's algorithm can be found in [21, 25, 34, 45, 60, 75, 76, 81, 87, 91, 123, 130, 131, 137, 152, 156]. Todd and Burrell [151] and Anstreicher [11] show how to extract dual variables without having to run the problem in the combined primal-dual form. This resolves the difficulties with a "sliding" objective function mechanism that was described in an earlier version of Karmarkar's paper.

Other recent iterative methods, independent of Karmarkar's work, are [102, 117, 121, 122, 133]. It is expected that more papers will appear soon in this field. Unfortunately, it is not easy to assess the practical significance of newly proposed algorithms because of insufficient computational experience. Moreover, an acceptable format for testing software for mathematical programming has not been decided yet. Some related questions are discussed in Section 10.

8 Recent interior-point methods

Methods of nonlinear programming have been tried in the past for solving linear programming problems (see [60] for references). However, the simplex method has been accepted as the most practical general-purpose method for linear programming. Practitioners often try to linearize a nonlinear problem, but not to "unlinearize" a linear one. The ellipsoid algorithm is in fact a theoretical application of nonlinear programming to linear programming. Interest in nonlinear methods for linear programming was recently revived with the strong claims about the practicality of Karmarkar's algorithm. In this section we review some of the recently proposed algorithms.

We have explained the notion of a barrier-function method in Section 7. Given a valid barrier function for a linear programming problem, it is usually a good idea to use the Newton search method for finding the optimum of the function, especially if the latter is convex. The method is usually applied to unconstrained optimization problems but can easily be extended to handle linear equality constraints. A single iteration of the Newton search method amounts to searching a direction. The direction is obtained by optimizing a quadratic function that approximates the given function. Suppose we need to minimize a convex function $F(x)$ subject to $Ax=b$, and assume we have already computed a point a. Let ∇ and H denote the gradient and Hessian, respectively, of the function F at the point a. The approximate optimization problem at a is to minimize $Q(x)=F(a)+\nabla^T(x-a)+\frac{1}{2}(x-a)^TH(x-a)$ subject to $Ax=b$. The search direction v is obtained by minimizing $\frac{1}{2}v^THv+\nabla^Tv$ subject to $Av=0$. This direction can also be obtained by linear optimization over an ellipsoid as follows. The inequality $(x-a)^TH(x-a)\leq\rho^2$ $(\rho\neq 0)$ describes an ellipsoid centered at a. An approximate linear optimization problem at a is to minimize $\nabla^T(x-a)$ subject to $(x-a)^TH(x-a)\leq\rho^2$ and $Ax=b$. The direction to the optimum of the latter is independent of ρ and coincides with the direction obtained by Newton's method. There is yet another way to obtain the search direction by transforming the space. The gradient direction is not invariant under linear transformations of the space. Thus, this direction depends on the representation. It is possible to transform the space, choose the gradient direction in the transformed space, and return to the original space, so that the resulting direction is the same as the one supplied by Newton's method. If the Hessian is symmetric and positive-definite then $H^{-1}=WW^T$ for some real matrix W. In that case the transformation is $v'=W^{-1}v$. The approximate quadratic optimization problem at a is to minimize $\frac{1}{2}\|v'\|^2+\nabla^TWv'$ subject to $AWv'=0$. Equivalently, v' is determined by minimizing the linear function ∇^TWv' over a ball that can be described as the intersection of the full-dimensional ball $\{x':\|x'-a'\|\le\rho\}$ with the affine subspace $\{x':AWx'=b\}$. Obviously, the direction v' is obtained by projecting the transformed gradient $W^T\nabla$ on the linear subspace $\{v':AWv'=0\}$. The projection of a vector u on a subspace $\{z:Mz=0\}$ (assuming M has a full row rank) is given by $[I-M^T(MM^T)^{-1}M]u$ (see Proposition 7.1). Thus, $v=Wv'=W[I-W^TA^T(AWW^TA^T)^{-1}AW]W^T\nabla$. In the applications the matrix W usually has a special structure that allows for more efficient ways to compute the search direction.

It is interesting to note that Newton's method for nonlinear optimization is not invariant under monotone transformations of the objective function. Thus, one obtains different search directions if the objective function is replaced by its logarithm or its nth power. On the other hand, the search directions corresponding to all possible monotone transformations of the same objective function form a two-dimensional linear space. Thus the different algorithms are closely related.

Gill, Murray, Saunders, Tomlin, and Wright [60] propose an algorithm for linear programming where they apply the Newton search method to a traditional form of a barrier function. They obtain an unconstrained optimization problem (but still with equality constraints) with an objective function of the form $F_\mu(x)=c^Tx+\mu B(x)$. Here $B(x)$ is the barrier component that tends to infinity as x approaches the boundary of the feasible domain, and μ is a positive scalar that is driven by the algorithm to zero. For the linear programming problem of minimizing c^Tx subject to $Ax=b$ and $x\ge 0$, the proposed barrier function is $B(x)=-\sum_k \ln x_k$, so the unconstrained optimization problem with the parameter μ is to minimize $c^Tx-\mu\sum_k \ln x_k$ subject to $Ax=b$. Let us use the notation of Section 7. Thus, let a denote the current point, $D=\mathrm{diag}(a_k)$, and e is a vector of 1's. The gradient of this barrier function at a is $c-\mu D^{-1}e$ and the Hessian is μD^{-2}. For comparison, consider Karmarkar's potential function $F(x)=$

$n \ln c^T x - \sum_k \ln x_k$. Thus, in Karmarkar's algorithm, the gradient at the current point a is equal to $(n/c^T a)c - D^{-1}e$ and the Hessian is equal to $D^{-2} - n(c^T a)^{-2} cc^T$. Thus, if $\mu = (c^T a)/n$ then the directions of the gradients in both cases are formally the same. Note that if the optimal value of $c^T x$ is zero then by taking $\mu = (c^T a)/n$ we drive μ to zero. It is shown in [60] that there is a choice of μ that yields the same search direction. The Hessian in Karmarkar's function contains an extra term, attributable to the dependence of μ on x. Because the Hessian in the [60] algorithm is equal to μD^{-2}, it follows that the matrix W in that case equals $\sqrt{\mu} D$. The search direction is hence $D[I - \mu DA^T(AD^2A^T)^{-1}AD]D(c - \mu D^{-1}e)$. To compute the search direction, there is no need to invert the matrix AD^2A^T or even to generate it. What we have to do is essentially solve a system of equations of the form $(AD^2A^T)\xi = \beta$. Thus, we can expand the system in the form

$$\eta - DA^T\xi = 0 \qquad AD\eta = \beta,$$

so that sparsity of A can be exploited.

Iri and Imai [76] propose an algorithm that can also be described as a Newton search direction method applied to a barrier function. They work on the problem of minimizing $c^T x$ subject to inequality constraints $Ax \geq b$, where the optimal value of the objective function is zero and an initial interior point is known. The barrier function is

$$F(x) = (c^T x)^{m+1} \Big/ \prod_k (A_k^T x - b_k),$$

where m is the number of rows in the matrix A and A_k^T denotes the kth row of A. This function is based on the ideas behind the potential function used by Karmarkar for the analysis of his algorithm. The function $F(x)$ is convex. The algorithm iterates as follows. Given an interior point x^i, it finds the minimum x' of the quadratic approximation to $F(x)$ based on the Hessian at x^i. It then searches the feasible segment of the line determined by x^i and x' for a minimum x^{i+1} of $F(x)$. It is interesting to consider an analogous algorithm for the problem in the form we have used in this chapter, namely: minimize $c^T x$ subject to $Ax = b$ and $x \geq 0$. The barrier function of Iri and Imai would be $f(x) = (c^T x)^{n+1}/(\prod_k x_k)$, where $x \in R_+^n$. The gradient ∇ of F at a point a satisfies, for every j, $\nabla_j = f(a)[(n+1)(c^T a)^{-1} c_j - a_j^{-1}]$. It follows that the Hessian H of f at a satisfies, for every i and j,

$$H_{ij} = \nabla_i \nabla_j - f(a)[(n+1)(c^T a)^{-2} c_i c_j - \delta_{ij} a_i^{-2}],$$

where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise. It follows that

$$\frac{H_{ij}}{f(a)}=\left((n+1)\frac{c_i}{c^Ta}-\frac{1}{a_i}\right)\left((n+1)\frac{c_j}{c^Ta}-\frac{1}{a_j}\right)-(n+1)\frac{c_ic_j}{(c^Ta)^2}+\frac{\delta_{ij}}{a_i^2}$$
$$=n(n+1)\left(\frac{c_i}{c^Ta}-\frac{1}{na_i}\right)\left(\frac{c_j}{c^Ta}-\frac{1}{na_j}\right)+\frac{\delta_{ij}}{a_i^2}-\frac{1}{na_ia_j}.$$

In matrix notation,

$$\frac{1}{f(a)}H=n(n+1)uu^T+D^{-1}\left(I-\frac{1}{n}ee^T\right)D^{-1},$$

where $u_k=(c^Ta)^{-1}c_k-(na_k)^{-1}$ and $D=\operatorname{diag}(a_k)$. It is now clear that H is positive semidefinite at any $a\in R_+^n$ so F is convex over R_+^n. Interestingly, Karmarkar's potential function $(c^Tx)^n/(\prod_k x_k)$ is not convex. For example, let $n=2$, $c=(1,0)^T$, and $a=(\frac{1}{2},\frac{1}{2})^T$. Then the Hessian of this function at a is proportional to the 2×2 matrix $I-ee^T$, which is indefinite. On the other hand, Karmarkar's algorithm works within the subspace $\{x:\sum_k x_k=1\}$. The restriction of Karmarkar's function to the latter subspace *is* convex. The proof of this claim follows by eliminating x_n. Consider the function $G(x)(c^Tx)^n/((1-\sum_k x_k)\prod_k x_k)$, where $x\in R_+^{n-1}$. The Hessian of the latter can be represented as the sum of the Hessian of $F(x)=(c^Tx)^n/\prod_k x_k$ (also with $x\in R_+^{n-1}$) and a positive multiple of ee^T, so the Hessian of G is positive definite.

The difference between the [60] and the [76] algorithms is that the former has a parameter that is updated by the algorithm, so that the Newton step is computed with respect to a different function in each iteration; the latter applies Newton's method to a fixed function. Because the rule for updating the parameter is incorporated into the barrier function in [76], the Hessian matrix is naturally a little more complicated. On the other hand, it is still amenable to sparse matrix computation. The search direction is obtained by solving (for v) an optimization problem of the form: minimize $\frac{1}{2}v^THv+g^Tv$ subject to $Av=0$. With a vector λ of Lagrange multipliers, this amounts to the following system of equations:

$$Hv-A^T\lambda=-g \qquad Av=0.$$

In [60], H is diagonal so the sparsity of the latter system is essentially determined by the sparsity of the matrix A. In the algorithm analogous to [76] the matrix H is dense but has a nice structure: $H=D^{-2}-ww^T$ for a certain vector w. Thus, the system of equations (in v and λ) is

$$D^{-2}v-A^T\lambda=(w^Tv)w-g \qquad Av=0.$$

The latter can be solved by taking a linear combination of solutions for two systems with a diagonal matrix in the upper left-hand corner:

$$D^{-2}v^1 - A^T\lambda^1 = w \qquad Av^1 = 0$$

and

$$D^{-2}v^2 - A^T\lambda^2 = -g \qquad Av^2 = 0.$$

The solution is $(v, \lambda) = t(v^1, \lambda^1) + (v^2, \lambda^2)$, where t is determined by the equation $w^T(tv^1 + v^2) = t$.

Vanderbei, Meketon, and Freedman [156] propose an algorithm that, in a certain sense, can be derived by dropping the projective ingredient from Karmarkar's algorithm. This algorithm (which we refer to as the VMF algorithm) can be described as follows. Suppose the problem is to minimize c^Tx subject to $Ax = b$ and $x \geq 0$. Let a denote the current interior point. Thus, $Aa = b$ and $a > 0$. The direction that the VMF algorithm takes is computed as follows. It is convenient to describe this direction using an affine scaling transformation. Given a, any point x is mapped into x' where $x'_k = x_k/a_k$. In particular, the current point a is mapped into the n-vector e consisting of n 1's. Let D denote the same diagonal matrix used above; that is, $D = \operatorname{diag}(a_k)$. Thus, the problem is transformed into an equivalent one: minimize c^TDx' subject to $ADx' = b$ and $x' \geq 0$.

The equivalence holds in general regardless of the vector b and the optimal value of the objective function. For comparison, consider Karmarkar's transformation. The latter is applied when the problem is to minimize c^Tx subject to $Ax = 0$, $e^Tx = 1$, and $x \geq 0$. Given an interior point a, the projective rescaling transformation takes x into $x' = (D^{-1}x)/(e^TD^{-1}x)$ so the problem is transformed into the following: minimize c^TDx'/e^TDx' subject to $ADx'/e^TDx' = 0$, $e^Tx'_j = 1$, and $x' \geq 0$. However, because $b = 0$ and the optimal value of the objective function is also equal to 0, it follows that the latter is equivalent to minimizing c^Tx' subject to $ADx' = 0$, $e^Tx'_j = 1$, and $x' \geq 0$.

In the transformed space the direction v' is obtained by projecting the direction Dc on the space $\{v' : ADv' = 0\}$. Thus, in this space the VMF algorithm moves from e to a point of the form $e + \alpha v'$, where α is chosen so that the new point is still interior. This step can be interpreted in two ways: (i) It is a *rescaled* steepest descent method; the algorithm moves (in the transformed space) in the direction opposite to the projected gradient. (ii) The algorithm considers (in the transformed space) a ball, contained in the feasible region and centered at the current point. It takes a direction that passes through the point where the objective function is minimized over that ball.

The VMF direction is computed as follows. First, the projection of any vector u on a subspace $\{z : Mz = 0\}$ is given by $(I - M^T(MM^T)^{-1}M)u$. Thus the projection of the vector Dc on the subspace $\{v' : ADv' = 0\}$ is given by $(I-(AD)^T(AD^2A^T)^{-1}AD)Dc$. The inverse image of the latter (i.e., the VMF direction in the original space) is obtained by multiplying by D. Thus, this direction is $D^2(I-A^T(AD^2A^T)^{-1}AD^2)c$.

Interestingly, the VMF direction can be derived from the [60] algorithm as the limit of the search direction when the parameter μ tends to zero. It is instructive to consider analogous algorithms for the problem in inequality constraints form, since it seems that in this form projections on subspaces may not be needed and insight into duality may be gained. Consider the problem of minimizing c^Tx subject to $Ax \geq b$ (where $A \in R^{m \times n}$ is of rank n). An adequate barrier function for the problem in this form would be $F(x) = c^Tx - \mu \sum_k (A_k^T x - b_k)$, where A_k^T denotes the kth row of the matrix A. Suppose that a is an interior feasible point; that is, $A_k^T a > b_k$ $(k = 1, \ldots, m)$. Let $D = \mathrm{diag}(A_k^T a - b_k) \in R^{m \times m}$. The gradient of F at a is equal to $c - \mu A^T D^{-1} e$ (where e is an m-vector of 1's). The Hessian of F at a is proportional to $A^T D^{-2} A$ so the Newton search direction at a with parameter μ is equal to $(A^T D^{-2} A)^{-1}(c - \mu A^T D^{-1} e)$. When μ tends to zero this direction tends to $v = (A^T D^{-2} A)^{-1} c$. Note that, as in the other algorithms, v is convenient for sparse matrix computation if A is sparse. It can be computed as the solution of $(A^T D^{-2} A) v = c$, which can be solved as an expanded sparse system in the unknowns u and v:

$$u - D^{-1}Av = 0 \qquad A^T D^{-1} u = c.$$

The latter arises in the solution of a familiar least-squares problem: minimize $\|u\|$ subject to $A^T D^{-1} u = c$, where the nearest point to the origin in the affine subspace $\{u : A^T D^{-1} u = c\}$ is sought. The components of v serve as Lagrange multipliers in this least-squares problem. Thus, the effort per iteration here is also dominated by a least-squares problem of the same size and structure. Interestingly, the dual linear programming problem can be represented in the form: maximize $b^T D^{-1} u$ subject to $A^T D^{-1} u = c$ and $u \geq 0$. This representation reflects a scaling transformation on the dual problem which is similar but not identical to the VMF transformation. The dual problem in the original form is to minimize $b^T y$ subject to $A^T y = c$ and $y \geq 0$. The VMF transformation would be to divide each y_k by its current value, whereas here we multiply each y_i by the corresponding current value of $A_k^T x - b_k$.

The algorithm proposed by Barnes [19] turns out to produce the same direction of movement as the VMF algorithm, even though it is stated differently. Given a, Barnes considers a certain ellipsoid E centered at a and contained in the feasible domain. He chooses as the direction of movement the one where the minimum of the linear function c^Tx over E is found. The ellipsoid E can be described as the intersection of the full-dimensional ellipsoid $\bar{E} = \{x : \|D^{-1}x - e\| \leq 1\}$ (where $D = \mathrm{diag}(a_k)$, as above) with the affine space $\{x: Ax=b\}$. It is easy to verify that $\bar{E}$ is contained in the nonnegative orthant, hence E is contained in the feasible domain. The equivalence to the VMF direction is by substituting $x=Dx'$. The Barnes direction in the transformed space is obtained by minimizing the function c^TDx' over the ball E' defined as the intersection of the full-dimensional ball $\bar{E}'=\{x': \|x'-e\|\le 1\}$ with the subspace $\{x': ADx'=b\}$. Obviously, this direction is independent of the radius of the ball. In other words, the same direction is obtained if the ellipsoid $\bar{E}$ is replaced by an ellipsoid $\bar{E}(t)=\{x: \|D^{-1}x-e\|\le t\}$ for any positive t. Thus, the property that the ellipsoid E is contained in the feasible domain is irrelevant.

A variation on Karmarkar's algorithm, proposed in [111], can be explained as follows. Karmarkar's algorithm is based on a projective transformation that maps the linear objective function into a fractional one. If the optimal value is zero then the function may be replaced by its numerator. However, the minima of the fractional function and its numerator over the inner ball usually do not coincide. In the original space, the inverse image of the inner ball is an ellipsoid (assuming the feasible domain is bounded, as in Karmarkar's formulation). If that ellipsoid is indeed a good approximation to the feasible domain then a good direction should be obtained by minimizing the objective function c^Tx over it. Thus, the problem of minimizing c^Tx subject to $Ax=b$ and $x\ge 0$ is approximated by the problem of minimizing c^Tx subject to $Ax=b$ and $x^TD^{-1}[I-(n-1)^{-1}ee^T]D^{-1}x\le 0$. With a vector λ of Lagrange multipliers for the linear equations and a scalar multiplier η attributed to the boundary of the ellipsoid, we obtain the following system of equations:

$$(D^{-2}-ww^T)x-A^T\lambda=\eta c \qquad Ax=b$$
$$x^T(D^{-2}-ww^T)x=0,$$

where $w=(n-1)^{-1/2}D^{-1}e$. This system can be solved using ideas already described in this section for the analogue of the [76] algorithm.

9 Simplex methods for network problems

The simplex algorithm has been used extensively and very successfully to solve network problems in practice. We will not review practical experience here. On the theoretical side, it was shown in [162] that exponential cases exist within the class of network problems. However, in recent years there have been identified simplex algorithms that perform only a polynomial number of pivot steps when applied to certain network problems. This direction is presumed to lead to better understanding of the simplex method. It may help finding a polynomial simplex variant or a proof that no such one exists.

Interesting work on network simplex algorithms was done by Cunningham [41, 42] (who, according to [73, 17], has done further unpublished work on polynomial simplex network algorithms). Roohy-Laleh [138] uses a primal simplex method for the assignment problem that runs in $O(n^3)$ pivot steps. Hung [72] obtains a number of pivots that is polynomial in the size of the problem (i.e., taking the numerical values of the costs into account). Balinski [17] develops a "signature" method that solves - in $O((n-1)(n-2)/2)$ pivot steps - the assignment problem as a dual simplex algorithm. Additional works on this problem are [10, 62, 154]. Ikura and Nemhauser [74] derive a dual simplex algorithm for the more general (but yet the uncapacitated) transportation problem. Their algorithm solves a sequence of scaled problems (using ideas from [54]). It is not strongly polynomial because the upper bound on the number of pivots depends on the numerical values. A polynomial dual simplex algorithm for the minimum cost-flow problem was presented in [128]. So far, it is not known whether a strongly polynomial simplex algorithm for the transportation problem exists. However, Orlin [126] claims that the original Edmonds and Karp scaling algorithm [54] can be made strongly polynomial with minor modifications. It should be noted that Tardos's algorithm for the transportation problem [147] is strongly polynomial, although it is not clear whether a strongly polynomial simplex algorithm can be derived from it.

10 Theory versus practice

Complexity theory in 1985 does not provide enough insight into the practical efficiency of algorithms for linear programming. Even if the issue of the distribution of inputs is resolved, the efficiency of an algorithm even on a single instance is determined by too many factors. The word "algorithm" usually refers to the underlying method and not to the detailed implementation. Thus, for example, the particular method of solving linear equations during a single iteration is not part of the algorithm. However, the efficiency of an algorithm in practice depends very heavily on the implementation. Obviously, different implementation strategies may be efficient for some inputs while others may be efficient for other inputs. Thus, when comparing different algorithms, one has to specify what part of the implementation strategy is to be considered a part of the algorithm. An interesting question at the present is whether interior-point methods will prove better for linear programming than the simplex algorithms. First, the concepts are not at all well defined. However, the answer will probably depend on the development of methods for analyzing the density structure of the matrices involved. It became apparent during the 12th International Symposium on Mathematical Programming (1985) that various interior-point methods converge in a small number of iterations. The numbers of pivot steps of simplex algorithms are also predictable to a certain extent. The potential for improvement in interior-point methods is in the reduction of the effort per iteration. Regarding the simplex method, the question of the effort per iteration seems to be better understood than the question of reducing the number of steps.

Reported computational experience is used by practitioners as an indicator for the efficiency of an algorithm. This approach suffers from many known disadvantages. Here we mention briefly only a few of them.

It is often said that there is a need for a standard set of test problems. The idea is that whenever a new algorithm is proposed for a problem, people would be able to compare it with previous methods by running it (or letting the proposer run it for them) on the test problems. However, one should be very careful to separate the research and development of the algorithm from the test problems. If there is a fixed small set of test problems, there is always the danger that we will improve the implementation by choosing the values of parameters to fit the test problems. Software packages typically have a certain degree of heuristicism involved. There are simple parameters - such as step sizes, accuracy in performing single iterations, and the like - that require some arbitrary choice of value by the designer. The choice should not depend on the test problems. Whenever sparse matrix computations are involved there is always the question of factorization and preconditioning; the performance of an implementation depends very much on the success of these operations, success that may be attributable to heuristic techniques having nothing to do with the theory of linear programming. It is natural that the person who models a real-life problem for solution as a linear programming problem understands the structure of the problem better than any heuristic computer program that must analyze this structure.

During the development of an algorithm, the designer sometimes likes to see his algorithm performing well and therefore enjoys running successful instances rather than unsuccesful ones. This distortion of proportion should not be projected to the phase of testing the final algorithm on independent test problems.

Obviously, a test of a final algorithm should be performed like any other controlled experiment. However, even this is not enough. There is a fundamental difficulty with testing algorithms that makes this field less exact than the classical exact sciences. When results of a computational experiment are reported, we usually cannot tell whether there have been prior experiments that produced contradicting results. Thus, the reported results may be easily reproducible yet not particularly instructive if the experiment itself was selected from a larger concealed set of experiments.

An even more problematic issue is proprietary material. We may refer to [40] for recommendations:

> Occasionally, the solution of a proprietary problem may shed light on some aspect of the algorithm which could not be seen otherwise. Nonetheless, we believe that these problems should be referred to in the report only under special circumstances and with adequate justification. . . . Experiments involving the use of proprietary programs should only be published because of the presentation of a new strategy or of new theoretical developments. Authors should be willing to reproduce their experiments for the referees. Where necessary, referees should exercise this right. (pp. 199, 201)

References

[1] Adler, I. 1983. "The Expected Number of Pivots Needed to Solve Parametric Linear Programs and the Efficiency of the Self-Dual Simplex Method." Technical Report, Department of Industrial Engineering and Operations Research, University of California-Berkeley.

[2] Adler, I., and S. E. Berenguer. 1981. "Random Linear Programs." Technical Report ORC 81-4, Operations Research Center, University of California-Berkeley.

[3] Adler, I., and S. E. Berenguer. 1981. "Duality Theory and the Random Generation of Linear Programs." Technical Report, Department of Industrial Engineering and Operations Research, University of California-Berkeley.

[4] Adler, I., and S. E. Berenguer. "Generating Random Linear Programs." Technical Report, Department of Industrial Engineering and Operations Research, University of California-Berkeley.

[5] Adler, I., R. M. Karp, and R. Shamir. 1983. "A Family of Simplex Variants Solving an $m \times d$ Linear Program in Expected Number of Pivots Depending on d Only." Report UCB CSD 83/157, Computer Science Division, University of California-Berkeley.

[6] Adler, I., R. M. Karp, and R. Shamir. 1983. "A Simplex Variant Solving an $m \times d$ Linear Program in $O(\min(m^2, d^2))$ Expected Number of Pivot Steps." Report UCB CSD 83/158, Computer Science Division, University of California-Berkeley.

[7] Adler, I., and N. Megiddo. 1985. "A Simplex Algorithm Whose Average Number of Steps Is Bounded between Two Quadratic Functions of the Smaller Dimension." *Journal of the Association for Computing Machinery* 32: 871-95.

[8] Adler, I., N. Megiddo, and M. J. Todd. 1984. "New Results on the Behavior of Simplex Algorithms." *Bulletin of the American Mathematical Society* 11: 378-82.

[9] Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithm.* Reading, Mass.: Addison-Wesley.

[10] Akgul, M. 1985. "A Genuinely Polynomial Primal Simplex Algorithm for the Assignment Problem." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[11] Anstreicher, K. M. 1985. "Analysis of a Modified Karmarkar Algorithm for Linear Programming." Working Paper Series B#84, School of Organization and Management, Yale University.

[12] Aspvall, B., and Y. Shiloach. 1980. "A Polynomial Algorithm for Solving Systems of Linear Inequalities with Two Variables per Inequality." *SIAM Journal on Computing* 9: 827-45.

[13] Avis, D., and V. Chvatal. 1978. "Notes on Bland's Pivoting Rule." *Mathematical Programming Study* 8: 24-34.

[14] Baathe, O., and P. O. Lindberg. 1985. "Studies on the Efficiency of the Stochastic Simplex Method." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[15] Balinski, M. L. 1984. "The Hirsch Conjecture for Dual Transportation Polyhedra." *Mathematics of Operations Research* 9: 629-33.

[16] Balinski, M. L. 1984. "A Good (Dual) Simplex Method for the Assignment Problem." Report AD 275.07.84, C.N.R.S., Laboratoire d'Econometrie, de l'Ecole Polytechnique, Paris.

[17] Balinski, M. L. 1985. "Signature Methods for the Assignment Problem." *Operations Research* 33: 527-36.

[18] Balinski, M. L., Th. M. Leibling, and A.-E. Nobs. 1985. "On the Average Length of Lexicographic Paths." RO 850415, Department de Mathematiques, Ecole Polytechnique Federale de Lausanne, CH-1015 Lausanne-Ecublens, Switzerland.

[19] Barnes, E. R. 1985. "A Variation on Karmarkar's Algorithm for Solving Linear Programming Problems." Research Report No. RC 11136, IBM T. S. Watson Research Center, Yorktown Heights, New York.

[20] Blair, C. E. 1983. "Random Linear Programs with Many Variables and Few Constraints." Faculty Working Paper No. 946, College of Commerce and Business Administration, University of Illinois-Champaign-Urbana.

[21] Blair, C. E. 1985. "The Iterative Step in the Linear Programming Algorithm of N. Karmarkar." Unpublished manuscript, College of Commerce and Business Administration, University of Illinois-Champaign-Urbana.

[22] Bland, R. G. 1977. "A Combinatorial Abstraction of Linear Programming." *Journal of Combinatorial Theory* 23 (B): 33-57.

[23] Bland, R. G. 1978. "New Finite Pivoting Rules." *Mathematics of Operations Research* 3: 103-7.

[24] Bland, R. G., D. Goldfarb, and M. J. Todd. 1981. "The Ellipsoid Method: A Survey." *Operations Research* 29: 1039-91.

[25] Blum, L. G. 1985. "Towards an Asymptotic Analysis of Karmarkar's Algorithm. Extended abstract.

[26] Borgwardt, K.-H. 1977. "Untersuchungen zur Asymptotik der mittleren Schriftzahl von Simplexverfahren in der Linearen Optimierung." Disserta-

tion, Universität Kaiserlautern.

[27] Borgwardt, K.-H. 1982. "Some Distribution-Independent Results about the Asymptotic Order of the Average Number of Pivot Steps of the Simplex Method." *Mathematics of Operations Research* 7: 441–62.

[28] Borgwardt, K.-H. 1982. "The Average Number of Steps Required by the Simplex Method Is Polynomial." *Zeitschrift für Operations Research* 26: 157–77.

[29] Borgwardt, K.-H. 1985. "Average Behavior of the Simplex Algorithm: Some Improvements in the Analysis of the Rotation-Symmetry-Model." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[30] Borodin, A., J. von zur Gathen, and J. E. Hopcroft. 1982. "Fast Parallel Matrix and GCD Computations." *Information and Control* 52: 241–56.

[31] Buck, R. C. 1943. "Partition of Space." *American Mathematical Monthly* 50: 541–4.

[32] Cavalier, T. M., and A. L. Soyster. 1985. "Some Computational Experience and a Modification of the Karmarkar Algorithm." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[33] Chandru, V., and B. P. Kochar. 1985. "A Class of Algorithms for Linear Programming." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[34] Charnes, A., T. Song, and M. Wolfe. 1984. "An Explicit Solution Sequence and Convergence of Karmarkar's Algorithm." Unpublished manuscript, University of Texas–Austin.

[35] Chvatal, V. 1983. *Linear Programming.* New York: W. H. Freeman and Co.

[36] Clarkson, K. 1984. "Linear Programming in $O(n(5/2)^d 3^{d^2})$ Time." Unpublished manuscript, Department of Computer Science, Stanford University.

[37] Clausen, J. "Recent Results on the Complexity of the Simplex Algorithm." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[38] Cline, A. K. 1970. "Uniform Approximation as a Limit of L_2 Approximations." Ph.D. thesis, University of Michigan.

[39] Cline, A. K. 1972. "Rate of Convergence of Lawson's Algorithm." *Mathematics of Computation* 26: 167–76.

[40] Crowder, H., R. S. Dembo, and J. M. Mulvey. 1979. "On Reporting Computational Experience with Mathematical Software." *ACM Transactions on Mathematical Software* 5: 193–203.

[41] Cunningham, W. H. 1976. "A Network Simplex Method." *Mathematical Programming* 11: 105–16.

[42] Cunningham, W. H. 1979. "Theoretical Properties of the Network Simplex Method." *Mathematics of Operations Research* 4: 196–298.

[43] Dantzig, G. B. 1963. *Linear Programming and Extensions.* Princeton, N.J.: Princeton University Press.

[44] Dantzig, G. B. 1980. "Expected Number of Steps for a Linear Program with a Convexity Constraint." Technical Report SOL 80-3, Systems Optimization Laboratory, Department of Operations Research, Stanford University.

[45] de Ghellinck, G., and J.-Ph. Vial. 1985. "An Extension of Karmarkar's Algorithm for Solving a System of Linear Homogenous Equations on the Simplex." Discussion Paper No. 8538, C.O.R.E., Catholic University of Louvain, Belgium.

[46] Dobkin, D. P., and S. P. Reiss. 1980. "The Complexity of Linear Programming." *Theoretical Computer Science* 11: 1–18.

[47] Dunham, R., D. G. Kelly, and J. W. Tolle. 1977. "Some Experimental Results Concerning the Expected Number of Pivots for Solving Randomly Generated Linear Programs." Report TR 77-16, Operations Research and Systems Analysis Department, University of North Carolina–Chapel Hill.

[48] Dyer, M. E. 1984. "Linear Time Algorithms for Two- and Three-Variable Linear Programs." *SIAM Journal on Computing* 13: 31–45.

[49] Dyer, M. E. 1984. "An $O(n)$ Algorithm for Multiple-Choice Knapsack Linear Program." *Mathematical Programming* 29: 57–63.

[50] Dyer, M. E. 1984. "On a Multidimensional Search Technique and Its Application to the Euclidean One-Center Problem." Department of Mathematics and Statistics, Teesside Polytechnic, Middlesbrough, Cleveland TS1 3BA, United Kingdom.

[51] Eaves, B. C., and U. G. Rothblum. 1985. "A Theory of Extending Algorithms for Parametric Problems." Technical Report RE5685, Department of Operations Research, Stanford University.

[52] Eaves, B. C., and H. Scarf. 1976. "The Solution of Systems of Piecewise Linear Equations." *Mathematics of Operations Research* 1: 1–27.

[53] Edmonds, J. 1967. "Systems of Distinct Representatives and Linear Algebra." *Journal of Research of the National Bureau of Standards* 71B: 241–5.

[54] Edmonds, J., and R. M. Karp. 1972. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." *Journal of the Association for Computing Machinery* 19: 248–64.

[55] Fiacco, A. V., and G. P. McCormick. 1968. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* New York: J. Wiley and Sons.

[56] Frank, A., and E. Tardos. 1985. "An Application of Simultaneous Approximation in Combinatorial Optimization." In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (1985),* pp. 459–63. Los Angeles: IEEE Computer Society Press.

[57] Frisch, K. R. 1955. "The Logarithmic Potential Method of Convex Programming." Unpublished manuscript, University Institute of Economics, Oslo, Norway.

[58] Fujishige, S. 1985. "An $O(m^3 \log m)$ Capacity-Rounding Algorithm for the Minimum-Cost Circulation Problem: A Dual Framework of the Tardos Algorithm." Technical Report No. 254 (85-3), Institute of Socio-Economic Planning, University of Tsukuba, Sakura, Ibaraki 305, Japan.

[59] Gay, D. M. 1985. "A Variant of Karmarkar's Linear Programming Algorithm for Problems in Standard Form." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[60] Gill, P. E., W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. 1985. "On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method." Technical Report SOL 85-11, Systems Optimization Laboratory, Department of Operations Research, Stanford University.

[61] Gill, P. E., W. Murray, and M. H. Wright. 1981. *Practical Optimization*. New York: Academic Press.

[62] Goldfarb, D. 1985. "Dual Simplex Algorithms for the Assignment Problem." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[63] Goldfarb, D., and W. Sit. 1979. "Worst Case Behavior of the Steepest Edge Simplex Method." *Discrete Applied Mathematics* 1: 277-85.

[64] Grotschel, M., L. Lovasz, and A. Schrijver. 1981. "The Ellipsoid Method and Its Consequences in Combinatorial Optimization." *Combinatorica* 1: 169-97.

[65] Grotschel, M., L. Lovasz, and A. Schrijver. 1984. "Corrigendum to Our Paper 'The Ellipsoid Method and Its Consequences in Combinatorial Optimization'." *Combinatorica* 4: 291-5.

[66] Grunbaum, B. 1967. *Convex Polytopes*. New York: Wiley.

[67] Haimovich, M. 1983. "The Simplex Algorithm Is Very Good! - On the Expected Number of Pivot Steps and Related Properties of Random Linear Programs." Technical Report, Columbia University.

[68] Haverly, C. A. 1985. "Behavior of the Simplex and Karmarkar Algorithms." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[69] Hoffman, A. J., M. Mannos, D. Sokolowsky, and N. Wiegmann. 1953. "Computational Experience in Solving Linear Programs." *J. Soc. Indust. Applied. Math.* 1: 17-33.

[70] Howe, R. 1983. "Linear Complementarity and the Average Volume of Simplicial Cones." Cowles Foundation Discussion Paper No. 670, Yale University.

[71] Huard, P. 1967. "Resolution of Mathematical Programming with Nonlinear Constraints by the Method of Centers." In J. Abadie (ed.), *Nonlinear Programming*, pp. 207-19. Amsterdam: North-Holland.

[72] Hung, M. S. 1983. "A Polynomial Simplex Method for the Assignment Problem." *Operations Research* 31: 595-600.

[73] Ikura, Y., and G. L. Nemhauser. 1982. "An Efficient Primal Simplex Algorithm for Maximum Weighted Vertex Packing on Bipartite Graphs." *Annals of Discrete Mathematics* 16: 149-68.

[74] Ikura, Y., and G. L. Nemhauser. 1983. "A Polynomial-Time Dual Simplex Algorithm for the Transportation Problem." Technical Report No. 602, School of Operations Research and Industrial Engineering, Cornell University.

[75] Iri, M. 1985. "Another 'Simple and Fast' Algorithm for Linear Programming." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[76] Iri, M., and H. Imai. 1985. "A Multiplicative Penalty Function Method for Linear Programming - Another 'New and Fast' Algorithm." Research Memorandum RMI 85-04, Department of Mathematical Engineering and Instrumentation Physics, University of Tokyo, Tokyo, Japan.

[77] Jeroslow, R. G. 1973. "Asymptotic Linear Programming." *Operations Research* 21: 1128–41.

[78] Jeroslow, R. G. 1973. "The Simplex Algorithm with the Pivot Rule of Maximizing Criterion Improvement." *Discrete Mathematics* 4: 367–77.

[79] Karmarkar, N. 1984. "A New Polynomial-Time Algorithm for Linear Programming." In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (1984)*, pp. 302–11. New York: ACM. Revised version in *Combinatorica* 4: 373–95.

[80] Karmarkar, N. K. 1985. "Further Developments in the New Polynomial Time Algorithm for Linear Programming." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[81] Karmarkar, N. K., and L. P. Sinha. 1985. "Application of Karmarkar's Algorithm to Overseas Telecommunications Facilities Planning." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[82] Karp, R. M., and C. H. Papadimitriou. 1980. "On Linear Characterizations of Combinatorial Optimization Problems." In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (1980)*, pp. 1–9. Los Angeles: IEEE Computer Society Press.

[83] Khachiyan, L. G. 1979. "A Polynomial Algorithm in Linear Programming." *Soviet Math. Dokl.* 20: 191–4.

[84] Khachiyan, L. G. 1980. "Polynomial Algorithms in Linear Programming." *USSR Computational Mathematics and Mathematical Physics* 20: 53–72.

[85] Klee, V., and G. J. Minty. 1972. "How Good Is the Simplex Algorithm?" In O. Shisha (ed.), *Inequalities III*, pp. 159–75. New York: Academic Press.

[86] Klee, V., and D. W. Walkup. 1967. "The d-Step Conjecture for Polyhedra of Dimension $d<6$." *Acta Math.* 117: 53–78.

[87] Kojima, M. 1985. "Determining Basic Variables of Optimum Solutions in Karmarkar's New LP Algorithm." Research Report No. B-164, Department of Information Sciences, Tokyo Institute of Technology, O-Okayama, Meguro-ku, Tokyo, Japan.

[88] Kolata, G. 1979. "Mathematicians Amazed by Russian's Discovery." *Science* (2 November): 545–6.

[89] Kolata, G. 1982. "Mathematician Solves Simplex Problem." *Science* (2 July): 39.

[90] Kolata, G. 1984. "A Fast Way to Solve Hard Problems." *Science* (21 September): 1379–80.

[91] Kortanek, K. O., D. N. Lee, and M. Shi. 1985. "An Application of a Hybrid Algorithm for Semi-Infinite Programming." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[92] Kozlov, M. K., S. P. Tarasov, and L. G. Khachiyan. 1979. "Polynomial Solvability of Convex Quadratic Programming." *Dokl. Akad. Nauk SSSR* 5: 1051–3.

[93] Kuhn, H., and R. E. Quandt. 1963. "An Experimental Study of the Simplex Method." American Mathematical Society, *Proc. Symp. Appl. Math.* 15: 107–24.

[94] Lawson, C. L. 1961. "Contributions to the Theory of Linear Least Maximum Approximation." Ph.D. thesis, University of California–Los Angeles.

[95] Lee, D. T., and F. F. Preparata. 1984. "Computational Geometry - A Survey." *IEEE Transactions on Computers* C-33: 1072–1101.

[96] Lemke, C. E. 1965. "Bimatrix Equilibrium Points and Mathematical Programming." *Management Science* 11: 681–9.

[97] Lenstra, A. K., H. W. Lenstra, Jr., and L. Lovasz. 1982. "Factoring Polynomials with Rational Coefficients." *Math. Ann.* 26: 515–34.

[98] Lenstra, H. W., Jr. 1983. "Integer Programming with a Fixed Number of Variables." *Mathematics of Operations Research* 8: 538–48.

[99] Levin, A. Yu. 1965. "On an Algorithm for Convex Minimization." *Soviet Mathematics Doklady* 160.

[100] Liebling, Th. M. 1973. "On the Number of Iterations of the Simplex Method." In R. Henn, H. Kunzi, and H. Schubert (eds.), *Methods of Operation Research* 17: 284–64.

[101] Lustig, I. 1985. "A Practical Approach to Karmarkar's Algorithm." Technical Report SOL 85-5, Systems Optimization Laboratory, Department of Operations Research, Stanford University.

[102] Mangasarian, O. 1984. "Normal Solutions of Linear Programs." *Mathematical Programming Study* 22: 206–16.

[103] Marsten, R. E., and D. F. Shanno. 1985. "On Implementing Karmarkar's Algorithm." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[104] May, J., and R. Smith. 1982. "Random Polytopes: Their Definition, Generation, and Aggregate Properties." *Mathematical Programming* 24: 39–54.

[105] Megiddo, N. 1982. "Is Binary Encoding Appropriate for the Problem-Language Relationship?" *Theoretical Computer Science* 19: 337–41.

[106] Megiddo, N. 1982. "Polylog Algorithms for LP with Application to Exploding Flying Objects." Unpublished manuscript, Carnegie-Mellon University.

[107] Megiddo, N. 1983. "Towards a Genuinely Polynomial Algorithm for Linear Programming." *SIAM Journal on Computing* 12: 347–53.

[108] Megiddo, N. 1983. "Linear-Time Algorithms for Linear Programming in R^3 and Related Problems." *SIAM Journal on Computing* 12: 759–76.

[109] Megiddo, N. 1984. "Linear Programming in Linear Time When the Dimension Is Fixed." *Journal of the Association for Computing Machinery* 31: 114–27.

[110] Megiddo, N. 1984. "Dynamic Location Problems." To appear in *Proceedings of the Third International Symposium on Locational Decisions.* Thompson's Island, Mass.

[111] Megiddo, N. 1984. "A Variation on Karmarkar's Algorithm." Unpublished manuscript.

[112] Megiddo,.N. 1985. "Partitioning with Two Lines in the Plane." *Journal of Algorithms* 6: 430–3.

[113] Megiddo, N. 1985. "A Note on the Generality of the Self-Dual Simplex Algorithm with Various Starting Points." *Methods of Operations Research* 49: 271–5.

[114] Megiddo, N. 1986. "On the Expected Number of Linear Complementarity Cones Intersected by Random and Semi-Random Rays." *Mathematical Programming* 35: 225–35.

[115] Megiddo, N. 1986. "Improved Asymptotic Analysis of the Average Number of Steps Performed by the Self-Dual Simplex Algorithm." *Mathematical Programming* 35: 140–72.

[116] Megiddo, N., and T. Ichimori. 1985. "A Two-Resource Allocation Problem Solvable in Linear-Time." *Mathematics of Operations Research* 10: 7–16.

[117] Mitra, G., M. Tamiz, J. Yadegar, and K. Darby-Dowman. 1985. "Experimental Investigation of an Interior Search Algorithm for Linear Programming." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[118] Motzkin, T. S. 1955. "The Probability of Solvability of Linear Equalities." In H. A. Antosiewicz (ed.), *Proceedings of the Second Symposium in Linear Programming,* pp. 607–11. USAF: National Bureau of Standards and Directorate of Management Analysis.

[119] Murty, K. G. 1978. "Computational Complexity of Complementary Pivot Methods." *Mathematical Programming Study* 7: 61–73.

[120] Murty, K. G. 1980. "Computational Complexity of Parametric Linear Programming." *Mathematical Programming* 19: 213–19.

[121] Murty, K. G. 1985. "A New Interior Variant of the Gradient Projection Method for Linear Programming." Technical Paper 85-18, Department of Industrial and Operations Engineering, University of Michigan.

[122] Murty, K. G., and Y. Fathi. 1984. "A Feasible Direction Method for Linear Programming." *Operations Research Letters* 3: 121–7.

[123] Nazareth, J. L. "Karmarkar's Method and Homotopies with Restarts." Manuscript, CDSS, P.O. Box 4908, Berkeley, Calif. 94704.

[124] Orden, A. 1976. "Computational Investigation and Analysis of Probabilistic Parameters of Convergence of a Simplex Algorithm." In *Progress in Operations Research II,* pp. 705–15. Amsterdam: North-Holland.

[125] Orden, A. 1980. "A Step Towards Probabilistic Analysis of Simplex Convergence." *Mathematical Programming* 19: 3–13.

[126] Orlin, J. B. 1984. "Genuinely Polynomial Simplex and Nonsimplex Algorithms for the Min-Cost Flow Problem." Working Paper 1615-84, Sloan School of Management, Massachusetts Institute of Technology.

[127] Orlin, J. B. 1985. "A Dual Version of Tardos' Algorithm for Linear Programming." Working Paper 1686-85, Sloan School of Management, Massachusetts Institute of Technology.

[128] Orlin, J. B. 1985. "A Polynomial Time Dual Simplex Algorithm for the Minimum Cost Flow Problem." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[129] Orlin, J. B. "On the Simplex Algorithm for Networks and Generalized Networks." *Mathematical Programming,* to appear.

[130] Padberg, M. W. 1985. "A Different Convergence Proof of the Projective Method for Linear Programming." Unpublished manuscript, New York University.

[131] Padberg, M. W. 1985. "Solution of a Nonlinear Programming Problem Arising in the Projective Method for Linear Programming." Unpublished manuscript, New York University.

[132] Padberg, M. W., and M. R. Rao. 1980. "The Russian Method and Integer Programming." GBA Working Paper, New York University.

[133] Pan, V. Y. 1983. "On the Computational Complexity of Solving a System of Linear Inequalities." Department of Computer Science, State University of New York.

[134] Pan, V., and J. Reif. 1985. "Efficient Parallel Solution of Linear Systems." In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (1985),* pp. 143–52. New York: ACM.

[135] Pickel, P. F. 1985. "Implementing the Karmarkar Algorithm Using Simplex Techniques." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[136] Rice, J. R., and K. H. Usow. 1968. "The Lawson Algorithm and Extensions." *Mathematics of Computation* 22: 118–27.

[137] Rinaldi, G. 1985. "The Projective Method for Linear Programming with Box-Type Constraints." Report R.119, Instituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, 00815 Roma.

[138] Roohy-Laleh, E. 1981. "Improvements to the Theoretical Efficiency of the Network Method." Ph.D. thesis, Carleton University.

[139] Ross, S. M. 1981. "A Simple Heuristic Approach to Simplex Efficiency." Department of Industrial Engineering and Operations Research, University of California-Berkeley.

[140] Saigal, R. 1983. "On Some Average Results for Random Linear Complementarity Problems." Department of Industrial Engineering, Northwestern University.

[141] Saigal, R. 1983. "An Analysis for the Simplex Method." Preliminary Report, Department of Industrial Engineering, Northwestern University.

[142] Shamir, R. 1984. "The Efficiency of the Simplex Method: A Survey." Department of Industrial Engineering and Operations Research, University of California-Berkeley.

[143] Shostak, R. 1981. "Deciding Linear Inequalities by Computing Loop Residues." *Journal of the Association for Computing Machinery* 28: 769–79.

[144] Smale, S. 1983. "On the Average Number of Steps of the Simplex Method of Linear Programming." *Mathematical Programming* 27: 241–62.

[145] Smale, S. 1983. "The Problem of the Average Speed of the Simplex Method." In A. Bachem, M. Grotschel, and B. Korte (eds.), *Mathematical Programming: The State of the Art,* pp. 530–9. Berlin: Springer-Verlag.

[146] Stone, R. E. 1980. "Khachiyan's Algorithm with Finite Precision." Working Paper SOL 80-1, Department of Operations Research, Stanford University.

[147] Tardos, E. 1984. "A Strongly Polynomial Minimum Cost Circulation Algorithm." Report No. 84356-OR, Institut fur Oconometrie und Operations Research, University of Bonn. Forthcoming in *Combinatorica.*

[148] Tardos, E. 1985. "A Strongly Polynomial Algorithm to Solve Combinatorial Linear Problems." Report No. 84360-OR, Institute for Econometrics and Operations Research, University of Bonn.

[149] Todd, M. J. 1983. "Polynomial Expected Behavior of a Pivoting Algorithm for Linear Complementarity and Linear Programming Problems." Technical Report No. 595, School of Operations Research and Industrial Engineering, Cornell University.

[150] Todd, M. J., and B. P. Burrell. 1985. "The Ellipsoid Algorithm Generates Dual Variables." *Mathematics of Operations Research* 10: 688–700.

[151] Todd, M. J., and B. P. Burrell. 1985. "An Extension of Karmarkar's Algorithm for Linear Programming Using Dual Variables." Technical Report No. 648, School of Operations Research and Industrial Engineering, Cornell University.

[152] Tomlin, J. A. 1985. "An Experimental Approach to Karmarkar's Projective Method for Linear Programming." Report, Ketron, Inc., Mountain View, Calif. 94040.

[153] Traub, J. F., and H. Wozniakowski. 1982. "Complexity of Linear Programming." *Operations Research Letters* 1: 59–62.

[154] Tufecki, S. 1985. "A Polynomial Dual Simplex Algorithm for Assignment and Transportation Problems." Manuscript, University of Florida.

[155] Ursic, S. 1982. "The Ellipsoid Algorithm for Linear Programming in Exact Arithmetic." In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (1982),* pp. 321–6. Los Angeles: IEEE Computer Society Press.

[156] Vanderbei, R. J., M. J. Meketon, and B. A. Freedman. 1985. "A Modification of Karmarkar's Linear Programming Algorithm." Report, AT&T Bell Laboratories, Holmdel, N.J. 07733.

[157] Vershik, A. M., and P. V. Sporyshev. 1983. "An Estimate of the Average Number of Steps in the Simplex Method, and Problems in Asymptotic Integral Geometry." *Soviet. Math. Dokl.* 28: 195–9.

[158] Wan, Y.-H. 1983. "On the Average Speed of the Lemke's Algorithm for Quadratic Programming." Department of Mathematics, State University of New York-Buffalo.

[159] Yamnitsky, B., and L. A. Levin. 1982. "An Old Linear Programming Algorithm Runs in Polynomial Time." In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (1982),* pp. 327–8. Los Angeles: IEEE Computer Society Press.

[160] Ye, Y. 1985. "*K*-Projection and the Cutting-Objective Methods for Linear Programming." Presented at the 12th Symposium on Mathematical Programming, Cambridge, Mass.

[161] Yudin, D. B., and A. S. Nemirovsky. 1976. "Informational Complexity and Effective Methods for Solving Convex Extremum Problems." *Economica i Mat. Metody* 12.

[162] Zadeh, N. 1973. "A Bad Network Problem for the Simplex Method and Other Minimim Cost Flow Algorithms." *Mathematical Programming* 5: 255–66.
[163] Zadeh, N. 1980. "What Is the Worst Case Behavior of the Simplex Algorithm?" Technical Report No. 37, Department of Operations Research, Stanford University.
[164] Zemel, E. 1984. "An $O(n)$ Algorithm for Multiple Choice Knapsack and Related Problems." *Information Processing Letters* 18: 123–8.