数理解析研究所講究録
第 872 巻 1994 年 49-64

Computational Geometry and Linear Programming

Hiroshi Imai*
Department of Information Science
University of Tokyo

Abstract

Computational geometry has developed many efficient algorithms for geometric problems in low dimensions by considering the problems from the unified viewpoint of geometric algorithms. It is often the case that such geometric problems may be regarded as special cases of mathematical programming problems in high dimensions. Of course, computational-geometric algorithms are much more efficient than general algorithms for mathematical programming when applied to the problems in low dimensions. For example, a linear programming problem in a fixed dimension can be solved in linear time, while, for a general linear programming problem, only weakly polynomial time algorithms are known. Although not so much attention has been paid to combine techniques in computational geometry and those in mathematical programming, both of them should be investigated more from each side in future.

In this paper, we first survey many useful results for linear programming in computational geometry such as the prune-and-search paradigm and randomization and those in mathematical programming such as the interior-point method. We also demonstrate how computational geometry and mathematical programming may be combined for several geometric problems in low and high dimensions. Efficient sequential as well as parallel geometric algorithms are touched upon.

1 Introduction

Linear programming is an optimization problem which minimizes a linear objective function under linear inequality constraints:

$$\begin{aligned} &\min\ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ &\text{s.t. } A\boldsymbol{x} \geq \boldsymbol{b} \end{aligned}$$

where $\boldsymbol{c}, \boldsymbol{x} \in \mathbf{R}^d$, $\boldsymbol{b} \in \mathbf{R}^n$, $A \in \mathbf{R}^{n \times d}$, A, $\boldsymbol{b}$, $\boldsymbol{c}$ are given and elements $x_1, \ldots, x_d$ of $\boldsymbol{x}$ are d variables. Linear programming would be the best optimization paradigm that is utilized in real applications, since it can be used as a powerful model of various discrete as well as continuous systems and even a large-scale linear programming problem with thousands of variables or more may be solved within a reasonable time.

Wide applicability of linear programming makes it very interesting to investigate algorithms for linear programming in the field of computer science. From the viewpoint of theory of algorithms, linear programming has many fertile aspects as follows.

First, linear programming can be viewed as both of discrete optimization and continuous optimization. In its original formulation, the problem is a continuous optimization problem. On the other hand, by investigating the feasible region $\{ \boldsymbol{x} \mid A\boldsymbol{x} \geq \boldsymbol{b} \}$, which is a convex polyhedron, it is seen that there exists an optimum solution which is an extreme point of the convex polyhedron

*Supported in part by the Grant-in-Aid of the Ministry of Education, Science and Culture of Japan.

if rank $A = d$, and, since the number of extreme points of A is bounded by $\binom{n}{d}$ (or $O(n^{\lfloor d/2 \rfloor})$ by the upper bound theorem of the polytope), the problem can in principle be solved by checking all the extreme points in a discrete manner. Basically using this discrete property, an algorithm for linear programming, called simplex method, was developed by Dantzig [5], which traverses a path of adjacent extreme points of the polyhedron to an optimum point. The simplex algorithm has been used as a useful tool in the commercial world.

Second, the time complexity of linear programming is very close to a threshold between tractable problems and intractable problems. In a famous book on NP-completeness by Garey and Johnson [6], linear programming is listed as one of a few big problems that have not been known to fall in which of tractable or intractable class at that time. In fact, the simplex method may traverse all the extreme points of the convex polyhedron in the worst case. Since the number of extreme points may be exponential, the simplex method is regarded as a non-polynomial method in general. However, in 1979, Khachian [18] showed that linear programming is polynomial-time solvable by using an ellipsoid method. Furthermore, in mid 1980's, Karmarkar [16] proposed an interior-point method for linear programming, which has better polynomial-time complexity in the worst case and may run faster on the average. Unlike the simplex method which traverses a path on the boundary of the convex polyhedron, the interior-point method works in the interior of the polyhedron. Since the interior of the polyhedron has continuous structure in itself, the interior-point method may be said to be an algorithm based on the continuous structure of linear programming.

Third, linear programming is still not known to admit any strongly polynomial-time algorithm. Khachian's ellipsoid method as well as Karmarkar's interior-point method yields weakly polynomial-time algorithms by nature since both are based on Khachian's ingenious approximation idea of solving a discrete problem by a continuous approach. For network flow problems, which form a useful subclass of linear programming, strongly polynomial-time algorithms are already developed (e.g., [25]), and this issue has been a big open problem concerning the complexity of linear programming.

Fourth, in some applications such as computer graphics, there arise linear programming problems such that d is much smaller than n or d is a constant, say three. This type of linear programming problems may be treated in computational geometry. Computational geometry, whose name was christened by Shamos in mid 1970's [27], is a field in computer science which treats geometric problems in a unified way from the viewpoint of algorithms. Efficient algorithms to construct the convex hull, Voronoi diagram, arrangement, etc., have been developed as fruitful results of the field. Until now, computational geometry has laid emphasis on low-dimensional geometric problems, especially problems in the two- or three-dimensional space. Also, most of computational geometric algorithms are discrete ones. Computational geometry from now should tackle higher dimensional problems and adopt more continuous approaches. One of useful techniques which computational geometry is now using and somehow has connection continuous structure is randomization. This introduces probabilistic behavior in algorithms, and to analyze such randomized algorithms we need to treat geometric problems in a more general setting. For linear programming, computational geometry yields a linear-time algorithm when the dimension is regarded as a constant. To derive such an algorithm, the prune-and-search paradigm is used. This paradigm was originally used in obtaining a linear-time algorithm for selection. Through computational geometry, the prune-and-search paradigm is generalized to higher dimensional problems, and, besides linear programming, produces many useful algorithms.

In this paper, we discuss some of the above-mentioned results concerning linear programming. This is done from the viewpoint of the author, and hence topics treated here might be biased compared with the too general title of this paper. Through highlighting such results, which would be of practical importance by themselves, we try to give some idea on future research directions in the related fields. Although, up to here in this introduction, linear programming is emphasized from the viewpoint of mathematical programming, computational geometry will also be treated as a big subject of this paper. We first describe the prune-and-search paradigm and its applications for linear programming in computational geometry in section 2 and then describe the interior-point method for linear programming in mathematical programming in section 3. In section 4, combining two paradigms in computational geometry and mathematical programming to obtain an efficient linear programming algorithm is discussed. In fact, it is definitely useless to distinguish linear programming in computational geometry and linear programming in mathematical programming, and we may treat and understand these problems in a more unified manner.

2 Linear Programming in Computational Geometry

In this section we first explain the prune-and-search paradigm, and its application to the two-dimensional linear programming problem in detail. Then two applications of the prune-and-search technique to some special linear programming problems are mentioned. We also describe randomized algorithms for linear programming.

2.1 Prune-and-search paradigm and its application to two-dimensional linear programming

In many of algorithmic paradigms, given a problem, its subproblems of smaller size are solved to obtain a solution to the whole problem. This is because the smaller the problem size is the more easily the problem may be solved. The prune-and-search paradigm tries to reduce the problem size by a constant factor by removing redundant elements at each stage, whose application to the two-dimensional linear programming problem is described below.

The prune-and-search paradigm is one of useful paradigms in the design and analysis of algorithms. It was used in linear-time selection algorithms [2]. As mentioned above, a key idea of this paradigm is to remove redundant elements by a constant factor at each iteration. In the case of selecting the $k(=k_0)$th element $x$ among $n$ elements, at the ith iteration, the algorithm finds a subset of s_i elements which are either all less than $x$ or all greater than $x$. Then, we may remove all the elements in the subset and, for $k_i = k_{i-1} - s_i$ and $k_{i-1}$ according as these elements in the subset are less or greater than $x$, respectively, find k_ith element among the remaining elements in the next step. Roughly speaking, finding the subset of elements whose size is guaranteed to be at least a constant factor $\alpha < 1$ of the current size can be done in time linear to the current size. Then, the total time complexity is bounded in magnitude by

$$n + (1-\alpha)n + (1-\alpha)^2 n + \cdots \le \frac{1}{\alpha} n.$$

A linear-time algorithm is thus obtained.

Let us see how this prune-and-search paradigm may be used to develop a linear-time algorithm for the two-dimensional linear programming problem, as shown by Megiddo [22] et al. Since this is simple enough to describe compared with the other methods in this paper, we here try to give a rather complete description of this algorithm. A general two-dimensional linear programming problem with $n$ inequality constraints can be described as follows:

$$\begin{array}{ll} \min & c_1x_1 + c_2x_2 \\ \text{s.t.} & a_{i1}x_1 + a_{i2}x_2 \ge a_{i0} \qquad (i = 1, \ldots, n) \end{array}$$

If one can illustrate the feasible region satisfying the inequality constraints in the (x_1, x_2)-plane, which is simply a convex polygon if bounded, the problem would be very easy to solve illustratively (see Figure 2.1). Here, instead of considering the problem in this general form, we restrict our attention to the following problem.

$$\begin{array}{ll} \min & y \\ \text{s.t.} & y \ge a_ix + b_i \qquad (i = 1, \ldots, n) \end{array}$$

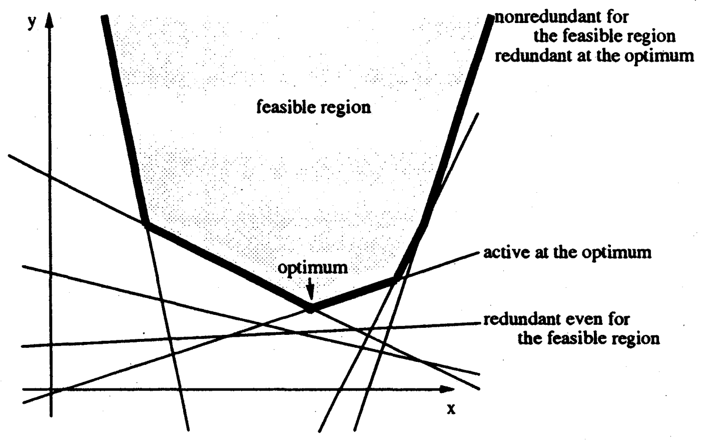
y
x
nonredundant for the feasible region redundant at the optimum
feasible region
active at the optimum
optimum
redundant even for the feasible region

Figure 2.1. A two-dimensional linear programming problem

This is because this special problem is almost sufficient to devise a linear-time algorithm for the general two-dimensional problem, and its simpler structure is better in order to exhibit the essence of the prune-and-search technique. Figure 2.1 depicts this restricted problem of $n = 7$.

We consider the problem as defined above. Define a function $f(x)$ by

$$f(x) = \max\{\, a_i x + b_i \mid i = 1, \ldots, n \,\}.$$

Then, the problem is equivalent to minimizing $f(x)$. The graph of $y = f(x)$ is drawn in bold lines in Figure 2.1. $f(x)$ has a nice property as follows. First we review the convexity of a function. A function $g \colon \mathbf{R} \to \mathbf{R}$ is convex if

$$g(\lambda x_1 + (1-\lambda)x_2) \le \lambda g(x_1) + (1-\lambda) g(x_2)$$

for any $x_1, x_2, \lambda \in \mathbf{R}$ with $0 < \lambda < 1$. If the above inequality always holds strictly, $g(x)$ is called strictly convex. Consider the problem of minimizing a convex function $g(x)$. x' is called a local minimum solution if, for any sufficiently small $\epsilon > 0$,

$$g(x') \le g(x' + \epsilon) \qquad \text{and} \qquad g(x') \le g(x' - \epsilon).$$

x' is a global minimum solution if the above inequality holds for arbitrary ϵ. Due to the convexity, any local minimum solution is a global minimum solution. Also, if $g(x)$ is strictly convex, there is at most one global minimum solution. Suppose there is a global minimum x^* of $g(x)$. Given x, we can determine, without knowing the specific value of x^*, which of $x < x^*$, $x = x^*$, $x > x^*$ holds by checking the following conditions locally for sufficiently small $\epsilon > 0$:

(g1) if $g(x + \epsilon) < g(x)$, $x \le x^*$;

(g2) if $g(x - \epsilon) < g(x)$, $x \ge x^*$;

(g3) if $g(x + \epsilon), g(x - \epsilon) \ge g(x)$, x is a global minimum solution.

Finally, for k convex functions $g_i(x)$ $(i = 1, \ldots, k)$, a function $g(x)$ defined by

$$g(x) = \max\{\, g_i(x) \mid i = 1, \ldots, k \,\}$$

is again convex.

Now, return to our problem of minimizing $f(x) = \max\{\, a_i x + b_i \mid i = 1, \ldots, n\}$. $f(x)$ is a continuous piecewise linear function. From the above discussions, we have the following.

(f1) $f(x)$ is convex (since $a_i x + b_i$ is trivially convex).

(f2) Given $x$, we can determine in which side of $x$ an optimum solution x^*, if it exists, lies in $O(n)$ time (first, compute $I = \{ i \mid a_i x + b_i = f(x) \}$, $a^+ = \max\{ a_i \mid i \in I \}$ and $a^- = \min\{ a_i \mid i \in I \}$, which can be done in $O(n)$ time; if $a^+ > 0$, $x^* \leq x$; if $a^- < 0$, $x^* \geq x$; if $a^+ \geq 0$ and $a^- \leq 0$, $x$ is an optimum solution; cf. (g1~3)).

Note that this property (f2) is obtained mostly from the convexity of $f$. This property is a key in search steps in the prune-and-search technique.

For simplicity, we assume there is a unique solution x^* minimizing $f(x)$ (that is, we omit the cases where min $f(x)$ is $-\infty$ or min $f(x)$ is attained on some interval; both cases can be handled easily by slightly changing the following algorithm). In the following, we show that

(f3) we can find an $x$ in $O(n)$ time such that, by determining in which side of $x$ the minimum x^* lies (this can be done in $O(n)$ time from (f2)), we can find at least $n/4$ constraints such that a linear programming obtained by removing these constraints still has the same optimum solution with the original problem (we call these constraints redundant).

As mentioned above, the prune-and-search technique thus finds a constant factor of redundant constraints, to be pruned, with respect to the current number of constraints for the two-dimensional linear programming. Then, applying this recursively, we can finally remove all the redundant constraints and get an optimum solution in time proportional to

$$n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \left(\frac{3}{4}\right)^3 n + \cdots$$

which is less than $4n$. That is, if (f3) is true, we obtain a linear-time algorithm for this linear programming problem.

To show (f3), first consider two distinct constraints $y \geq a_i x + b_i$ and $y \geq a_j x + b_j$. If $a_i = a_j$, according as $b_i < b_j$ or $b_i \geq b_j$, we can discard the ith or $j$th, respectively, constraint immediately, because, then, $a_i x + b_i$ is smaller or not smaller, respectively, than $a_j x + b_j$ for any $x$. If $a_i \neq a_j$ (suppose without loss of generality $a_i > a_j$), there exists $x_{ij}$ satisfying $a_i x_{ij} + b_i = a_j x_{ij} + b_j$. When $x_{ij} > x^*$ (resp. $x_{ij} < x^*$), we see the ith (resp. $j$th) constraint is redundant, since $a_i x^* + b_i$ is smaller (resp. greater) than $a_j x^* + b_j$.

Based on this observation, match $n$ constraints into $n/2$ disjoint pairs $\{(i,j)\}$, For matched pairs with $a_i = a_j$, discard one of the two constraints according to the above procedure. For the other pairs, compute $x_{ij}$, and find the median x' among those $x_{ij}$ (recall that we can find the median of n' numbers in $O(n')$ time as mentioned above) and test on which side of x' the minimum x^* lies. If x' is an optimum solution, we have done. Otherwise, since x' is the median, a half of $x_{ij}$'s lies in the opposite side of x' with x^*. For a pair $(i,j)$ with $(x' - x^*)(x' - x_{ij}) \leq 0$, we can determine on which side of $x_{ij}$ the x^* lies, and hence discard one of them as above. Thus, in $O(n)$ time, we can discard at least $n/4$ constraints, and have shown (f3).

As noted in the discussion, even if min $f(x)$ is $-\infty$ or is attained on some interval, we can detect it very easily in $O(n)$ time bound. Thus, we have shown that the special linear programming problem of the above-mentioned form with two variables and $n$ constraints can be solved in $O(n)$ time. The general linear programming problem can be solved in linear time in an analogous way.

Still, this application of the prune-and-search paradigm to the two-dimensional linear programming is quite similar to the case for linear-time selection. The prune-and-search technique can be generalized to higher dimensions, and then algorithms obtained through it are really computational-geometric ones. The higher-dimensional prune-and-search technique works as follows. An underlying assumption of the general technique is that an optimum solution is determined by at most a constant number of the objects, which is $d$ in the nondegenerate case for linear programming. In general terms, the algorithm searches relative to $n$ objects in the $d$-dimensional space by recursively solving a constant number of sub-problems in the $(d-1)$-dimensional space recursively, thus reducing the size of the problem by a constant factor in the $d$-dimensional space. Roughly speaking, the following characteristics of the pruning technique allow a linear-time algorithm to be devised for

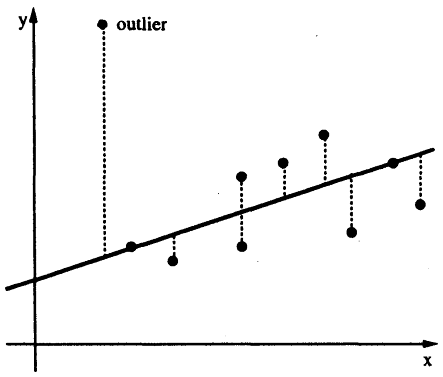

Figure 2.2. Linear L_1 approximation of points

a search relative to n objects in the d-dimensional space. At each iteration, the algorithm prunes the remaining objects by a constant factor, α, by applying a test a constant number of times. The test in the d-dimensional space is an essential feature of the algorithm since the complexity of the algorithm depends on the test being able to report the relative position of an optimum solution in linear time with respect to the number of remaining objects. The test in the d-dimensional space is performed by solving the $(d-1)$-dimensional subproblems as mentioned above. Hence, there are a total of $O(\log n)$ steps, with the amount of time spent at each step geometrically decreasing as noted above, taking linear time in total.

This approach was first adopted by Megiddo [22] et al. By this approach, linear programming in a fixed dimension can be solved in $O(2^{2^d})$ time, which is linear in n. However, this time complexity is doubly exponential in d, and the algorithm may be practical only for small d. This complexity has been improved in several ways (e.g., see [3]), but is still exponential with respect to d. We will return to this issue in the next section. Before it, we mention two applications of the prune-and-search technique to larger special linear programming problems.

2.2 Applications of the prune-and-search paradigm to special linear programming problems

Here, we mention two applications of the prune-and-search paradigm to special linear programming problems which are not a two-dimensional linear programming problem. One is on linear L_1 approximation of n points in the plane by Imai, Kato and Yamamoto [12] and the other is on the assignment problem with much fewer demand points than supply points by Tokuyama and Nakano [29].

(1) Linear L_1 approximation of points

Approximating a set of n points by a linear function, or a line in the plane, called the line-fitting problem, is of fundamental importance in numerical computation and statistics. The most frequently used method is the least-squares method, but there are alternatives such as the L_1 and the L_∞ (or Chebyshev) approximation methods. Especially, the L_1 approximation is more robust against outliers than the least-squares method, and is preferable for noisy data.

Let S be a set of n points p_i in the plane and denote the (x, y)-coordinate of point p_i by (x_i, y_i) $(i = 1, \ldots, n)$. For an approximate line defined by $y = ax + b$ with parameters a and b, the following error criterion, minimizing the L_1 norm, of the approximate line to the point set S defines the L_1 approximation:

$$\min_{a,b} \sum_{i=1}^{n} |y_i - (ax_i + b)|$$

Figure 2.2 depicts the case of $n = 10$ with an outlier.

This problem can be formulated as the following linear programming problem with $n+2$ variables $a,\ b,\ c_i\ (i=1,\dots,n)$:

$$\begin{aligned}\min \quad & \sum_{i=1}^{n} c_i \\ \text{s.t.} \quad & y_i - (ax_i + b) \le c_i \\ & -y_i + (ax_i + b) \le c_i\end{aligned}$$

Here, $x_i,\ y_i\ (i=1,\dots,n)$ are given constants. This linear programming problem has $n+2$ variables and more inequalities, and hence the linear-time algorithm for linear programming in a fixed dimension cannot be applied. However, the problem is essentially a two-dimensional problem. By using the point-line duality transformation, which is one of the best tools used in computational geometry, we can transform this problem so that the two-dimensional prune-and-search technique may be applied. In the L_1 approximation problem, however, any infinitesimal movement of any point in S changes the norm (and possibly the solution), and, in that sense, redundant points with respect to an optimum solution do not exist. Hence, direct application of the pruning technique does not produce a linear-time algorithm.

Imai, Kato and Yamamoto [12] give a method of overcoming this difficulty by making full use of the piecewise linearity of the L_1 norm to obtain a linear-time algorithm. Furthermore, in his master's thesis, Kato generalizes this result to higher dimensional L_1 approximation problem. Although these algorithms are a little complicated, they reveal how powerful the prune-and-search technique is in purely computational-geometric settings.

(2) Assignment problem

The assignment problem is a typical problem in network flow. The assignment problem with n supply vertices and fewer k demand vertices is formulated as follows:

$$\begin{aligned}\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{k} w_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_{ij} = n_j \quad (j=1,\dots,k), \qquad \sum_{j=1}^{k} x_{ij} = 1 \quad (i=1,\dots,n) \\ & 0 \le x_{ij} \le 1\end{aligned}$$

where x_{ij} are variables and $\sum_{j=1}^{k} n_j = n$ for positive integers n_j. Since this is an assignment problem, x_{ij} should be an integer, but all the extreme points of the polytope of this problem are integer-valued and hence this problem can be formulated as a simple linear programming problem of kn variables.

For this assignment problem, Tokuyama and Nakano [29] give the following nice geometric characterization. For this assignment problem, consider a set S of n points p_i in the k-dimensional space whose coordinates are defined by

$$p_i = (w_{i1}, w_{i2}, \dots, w_{ik}) - \frac{1}{k}\sum_{j=1}^{k} w_{ij}(1,1,\dots,1).$$

Each point in S is on the hyperplane H: $x_1 + x_2 + \cdots + x_k = 0$. For a point $g = (g_1, g_2, \dots, g_k)$ on the hyperplane H, define

$$T(g;j) = \bigcap_{h=1}^{k} \{\, (x_1,\dots,x_k) \text{ on the hyperplane } H \mid x_j - x_h \le g_j - g_h \,\}$$

$T(g;j)\ (j=1,\dots,k)$ partition the hyperplane H. This partition is called an optimal splitting if each $T(g;j)$ contains n_j points from S. In the case of $k=3$, the hyperplane is just a plane, and we

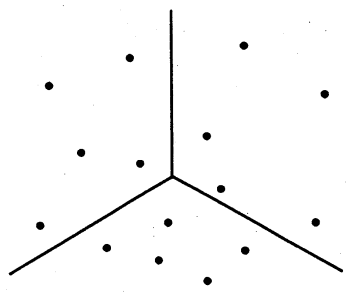

Figure 2.3. An optimal splitting for $n = 15$, $k = 3$ and $n_1 = n_2 = n_3 = 5$

can depict an example. Figure 2.3 gives such an example with $k = 3$ and $n_1 = n_2 = n_3 = 5$. Then, a theorem in [29] states that there exists an optimal splitting for any n_j satisfying the condition, and, for the optimal splitting by g, x_{ij} defined by $x_{ij} = 1$ if p_i is in $T(g; j)$ and $x_{ij} = 0$ otherwise is an optimum solution to the assignment problem.

Thus, the assignment problem is reduced to a geometric problem of finding an optimal splitting. Again, as in L_1 linear approximation, this problem has kn variables and more inequality constraints, and the linear-time algorithm for linear programming in a fixed dimension cannot be applied. Numata and Tokuyama [24] apply the $(k-1)$-dimensional prune-and-search technique to this geometrically interpreted problem and obtained an $O(((k+1)!)^2 n)$-time algorithm. This is linear if k is regarded as a constant, although even for k of moderate size the complexity becomes too big. For $k = 2, 3$, this algorithm may work well in practice. A linear-time algorithm for the assignment problem with a constant k has not been known before, and such an algorithm becomes available through the geometric interpretation explained above.

Besides this algorithm, Tokuyama and Nakano [29] give a randomized algorithm to solve this problem. We will return to this problem at the end of the next subsection.

2.3 Randomized algorithm for linear programming

The prune-and-search technique thus produces linear-time algorithms for linear programming in a fixed dimension, which is theoretically best possible. However, the time complexity depends upon the dimension d exponentially, and hence, even for d of moderate size, the algorithms become inefficient in practice. One of ways to overcome this difficulty is to use randomization, which has been recognized as a powerful tool in computational geometry. Here, randomization does not mean to assume any probabilistic distribution on the problem, say on the inequality constraints in this case. Instead, randomization introduces probabilistic behavior in the process of algorithms. By randomization, it becomes possible somehow to investigate the average case complexity of problems besides the worst case complexity, which is quite nice from the practical point of view. Also, it is often the case that randomized algorithms are rather simple and easy to implement.

Here, we first explain a randomized algorithm for linear programming proposed by Clarkson [3] briefly. Consider a two-dimensional linear programming problem treated in section 2.1. Recall that the problem can be regarded as finding two active inequality constraints at an optimum solution and removing all the other redundant constraints.

Take a subset S_0 of $\sqrt{n}$ constraints among n constraints randomly and independently. Solve a linear programming problem with this subset S_0 of constraints using the same objective function to obtain an optimum solution (x_0, y_0) for this subproblem. In case the other $n - \sqrt{n}$ constraints are satisfied at this optimum solution (i.e., there is no i with $y_0 < a_i x_0 + b_i$), this optimum solution for the subset of constraints is an optimum solution for all the constraints, and we are done.

Otherwise, we compute a set S_1 of constraints i which violate the computed solution: $y_0 < a_i x_0 + b_i$. Figure 2.4 illustrates the case of $n = 9$ where $\sqrt{n} = 3$ constraints are randomly sampled (denoted by bold lines) and there are constraints (denoted by dotted lines) violating the optimum for the sampled constraints. This set S_1 necessarily has at least one of two active constraint at the

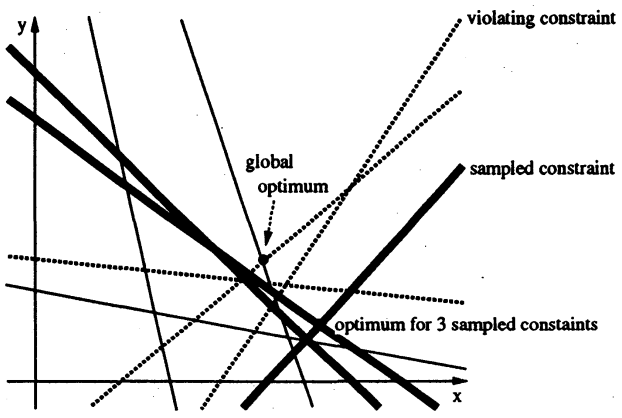

Figure 2.4. A randomized algorithm for linear programming

global optimum solution, which may be observed by overlaying the two active constraints forming the optimum with the current subset of constraints. In Figure 2.4, exactly one constraint active at the global optimum is included in S_1. The size of S_1 depends on the subset of $\sqrt{n}$ constraints chosen through randomization. It can be shown that the expected size of S_1 is $O(\sqrt{n})$, which is a key of this randomized algorithm. That is, by randomly sampling a subset S_0 of $\sqrt{n}$ constraints, we can find a set S_1 of constraints of expected size $O(\sqrt{n})$ which contains at least one of active constraints at the global optimum solution.

We again sample another set S_2 of $\sqrt{n}$ constraints, and this time solve a linear programming problem with constraints in $S_1 \cup S_2$. Let S_3 be a set of constraints violating the optimum solution to the subproblem for $S_1 \cup S_2$. Again, it can be seen that the expected size of S_3 is $O(\sqrt{n})$ and that $S_1 \cup S_3$ contains at least two among two active constraints (hence, exactly two in this two-dimensional case) at the global optimum solution. Since $S_1 \cup S_3$ contains those two active constraints at the optimum solution and its size is $O(\sqrt{n})$ on the average, the original problem for n constraints is now reduced to that for $O(\sqrt{n})$ constraints. Then, recursively applying this procedure solves the problem efficiently.

Based on the idea outlined above, Clarkson [3] gives a Las Vegas algorithm which solves the linear programming problem in a fixed dimension d rigorously in $O(d^2n + t(d)\log n)$ running time with high probability close to 1, where $t(d)$ is a function of d and is exponential in d. By randomization, the constant factor of n becomes dependent on d only polynomially, unlike the linear-time algorithm based on the prune-and-search paradigm. This is achieved by using random sampling in the algorithm. However, still there is a term in the complexity function which is exponential in d, and the algorithm is not a strongly polynomial algorithm.

A main issue in the design and analysis of this randomized algorithm is to evaluate the expected number of violating constraints to the optimum for the sampled set S_0. In this case, this evaluation can be performed completely in a discrete way. However, in more general case, the continuous model of probability may be used as in [9, 31], and, in this sense, introducing randomization in geometric algorithms may lead to investigating continuous structures of geometric problems more.

Now, return to the assignment problem with k demand vertices and n supply vertices mentioned in the previous subsection. The prune-and-search paradigm yields the $O(((k+1)!)^2n)$-time algorithm for it as mentioned above. Tokuyama and Nakano [29] propose a randomized algorithm, making use of random sampling, with randomized time complexity $O(kn + k^{3.5}n^{0.5}\log n)$. This algorithm is optimum for $k \ll n$, since the complexity becomes simply $O(kn)$ then. Thus, for the assignment problem with $k \ll n$, randomization gives a drastic result. It should be emphasized again that this becomes possible by establishing a nice bridge between geometry and combinatorial optimization, and by applying the randomization paradigm suitable for geometric problems.

These randomized algorithms are regarded as fruitful results by combining computational geometric results with linear programming and its special case. We will return to this issue in section 4 after explaining a continuous approach.

3 Linear Programming in Mathematical Programming

For linear programming, a nonlinear approach has recently been shown to be efficient for large-scale linear programming problems in mathematical programming, and the so-called interior-point method is now recognized as a powerful method for linear programming. In this section, among many interior-point algorithms, we explain the multiplicative penalty function method proposed by Iri and Imai [15]. Then, we show that, by applying the general interior-point method, an improved algorithm may be obtained for the planar minimum-cost flow problem, which is a special case of linear programming. This is another kind of merit of the interior-point method for linear programming. Applying the interior-point method to constructing parallel algorithms for linear programming is also discussed.

3.1 Multiplicative penalty function method for linear programming

The multiplicative penalty function method, proposed by Iri and Imai [15], is an interior method which minimizes the convex multiplicative penalty function defined for a given linear programming problem with inequality constraints by the Newton method. In this sense, the multiplicative penalty function method can be said to be the simplest and most natural method among interior-point methods using penalty functions. The multiplicative penalty function may be viewed as a affine variant of the potential function of Karmarkar [16]. In [15], the local quadratic convergence of the method was shown, while the global convergence property was left open. Zhang and Shi [32] prove the global linear convergence of the method under an assumption that the line search can be performed rigorously. Imai [11] shows that the number of main iterations in the multiplicative penalty function method is $O(n^2L)$, where at each iteration a main task is to solve a linear system of equations of d variables to find the Newton direction. Iri [14] further shows that the original version of the multiplicative penalty function method runs in $O(n^{1.5}L)$ main iterations and that an algorithm with an increased penalty parameter runs in $O(nL)$ iterations. In that paper [14], it is also shown that the convex quadratic programming problem can be solved by the multiplicative penalty function method within the same complexity.

There are now many interior-point algorithms for linear programming. For those algorithms, the interested readers may refer to [16, 19, 23].

We consider the following linear programming problem in the form mentioned in the introduction. In the sequel, we further assume the following (cf. [15]):
(1) The feasible region $X = \{ \boldsymbol{x} \mid A\boldsymbol{x} \geq \boldsymbol{b} \}$ is bounded.
(2) The interior Int X of the feasible region X is not empty.
(3) The minimum value of $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$ is zero.
The assumption (3) might seem to be a strong one, but there are several techniques to make this assumption hold for a given problem. Then the multiplicative penalty function for this linear programming problem is defined as follows:

$$F(\boldsymbol{x}) = (\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x})^{n+\delta} \Big/ \prod_{i=1}^{n}(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} - b_i) \qquad (\boldsymbol{x} \in \mathrm{Int}\ X)$$

where $\boldsymbol{a}_i^{\mathrm{T}} \in \mathbf{R}^d$ is the i-th row vector of A and δ is a constant greater than or equal to 1. This function is introduced in [15], and there δ is set to one. In [14], δ is set to approximately n. In Figure 3.1, contours of $F(\boldsymbol{x})$ in the two-dimensional case with $n = 4$ are shown. Under these assumptions, when $F(\boldsymbol{x}) \to 0$, the distance between $\boldsymbol{x}$ and the set of optimum solutions converges to zero. The multiplicative penalty function method directly minimizes the penalty function $F(\boldsymbol{x})$ by the Newton method, starting from some initial interior point.

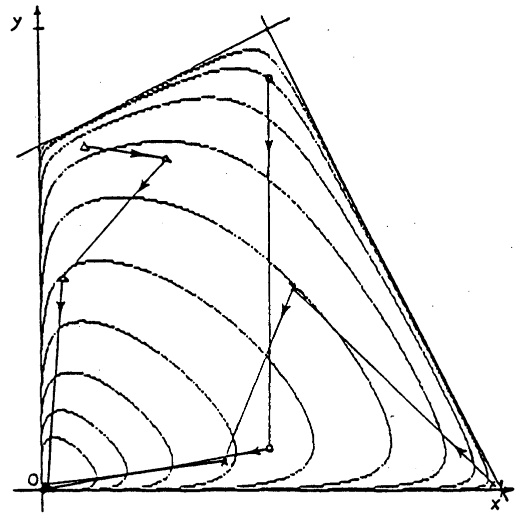

Figure 3.1. An example of the multiplicative penalty function $F(\boldsymbol{x})$ (from [15]): $\min\{ x + y \mid x \geq 0,\ y \geq 0,\ 2-2x-y \geq 0,\ 3+2x-4y \geq 0 \}$; $F(\boldsymbol{x}) = (x+y)^5/[xy(2-2x-y)(3+2x-4y)]$; contours for function values 4^i for $i = -4, \ldots, 5$

Specifically, define $\boldsymbol{\eta} = \boldsymbol{\eta}(\boldsymbol{x})$ and $H = H(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathrm{Int}\ X$ by

$$\boldsymbol{\eta}(\boldsymbol{x}) \equiv \frac{\nabla F(\boldsymbol{x})}{F(\boldsymbol{x})} = \nabla(\log F(\boldsymbol{x})) = (m+1)\frac{\boldsymbol{c}}{\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}} - \sum_{i=1}^{m} \frac{\boldsymbol{a}_i}{\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} - b_i}$$

$$H(\boldsymbol{x}) \equiv \frac{\nabla^2 F(\boldsymbol{x})}{F(\boldsymbol{x})} = \nabla^2(\log F(\boldsymbol{x})) + \boldsymbol{\eta}(\boldsymbol{x})\boldsymbol{\eta}(\boldsymbol{x})^{\mathrm{T}}$$

$\boldsymbol{\eta}(\boldsymbol{x})$ is the gradient divided by $F(\boldsymbol{x})$ and $H(\boldsymbol{x})$ is the Hessian divided by $F(\boldsymbol{x})$. Then, the Newton direction $\boldsymbol{\xi}$ at $\boldsymbol{x}$ is defined as follows:

$$H(\boldsymbol{x})\boldsymbol{\xi} = -\boldsymbol{\eta}(\boldsymbol{x})$$

It can be seen from simple calculation that $H(\boldsymbol{x})$ is positive definite under the assumptions, and hence $F(\boldsymbol{x})$ is a strictly convex function. Therefore, in Figure 3.1, all the contours (or level sets) are convex. Although $F(\boldsymbol{x})$ is highly nonlinear, it still retains a very good property on the convexity. Then a main iteration of the algorithm consists of computing the gradient, Hessian and Newton direction at the current solution, then performing line search along the Newton direction, and updating the current solution to a solution of the line search. The algorithm repeats this procedure until the current solution becomes close enough to the optimum solution. In Figure 3.1, three series of iterations are depicted.

One of main points in the analysis of the multiplicative penalty function method is to give an estimate on the value of the quadratic form $H \equiv \boldsymbol{\xi}^{\mathrm{T}} H(\boldsymbol{x})\boldsymbol{\xi}$ for the Newton direction. In general, $H > 1/2$. For $\delta > 1$, $H < \delta/(\delta - 1)$. Making use of this estimate, Iri [14] derived the bound $O(nL)$ on main iterations. This complexity is the same with that of Karmarkar [16]. Now, interior-point algorithms having $O(\sqrt{n}L)$ bound on main iterations are also known (e.g., [28, 19, 23, 30]). Using fast matrix multiplication partly, Vaidya [30] gives an algorithm which requires $O((n + d)^{1.5}dL)$ arithmetic operations on L-bit numbers.

3.2 Implication of the interior-point method to parallel algorithms

In general, linear programming is shown to be P-complete, and hence it would be difficult to design a parallel algorithm for linear programming whose parallel time is polylogarithmic in d

and n. However, the interior-point method may be used to devise parallel algorithms whose time complexity is sublinear in d and n. This is also the case for linear programming problems arising in geometric situations.

In the interior-point method, each iteration itself can be highly parallelized. In fact, as mentioned repeatedly, a main step of the iteration is to solve the linear system of equations, which can be parallelized very efficiently in polylogarithmic time in theory (e.g., see [26]).

This implies that, applying interior-point algorithms requiring $O(\sqrt{n}L)$ main iterations to a linear programming problem, a parallel algorithm for the problem with $O(\sqrt{n}L \text{ polylog } n)$ parallel time complexity can be obtained. This parallel time is sublinear in n, although it is linear in L. For some network flow problem whose constraint matrix is totally unimodular and whose costs and capacities are constants, L can be considered to be a constant, and hence this approach gives a totally sublinear parallel algorithm for it. Even if L is not a constant, parallel algorithms whose complexity is sublinear in n are of great interest. This approach is adopted in [8].

Considering the application of the interior-point method to planar network flow, which will be explained in the next subsection, the nested dissection technique can also be parallelized [26], and a parallel algorithm with $O(\sqrt{n}\log^3 n \log(n\gamma))$ parallel time using $O(n^{1.094})$ processors can be obtained [13], where γ is the largest absolute value among edges costs and capacities represented by integers. Also, this parallel algorithm is best possible with respect to the sequential algorithm, to be mentioned in the next subsection, that is, the parallel time complexity multiplied by the number of processors is within a polylog factor of the sequential time complexity. This result roughly says that the planar minimum cost flow problem can be solved in parallel in $O(n^{0.5+\epsilon})$ time with almost linear number of processors, which is currently best possible.

Thus, treating a special linear programming problem in a more general setting, better parallel algorithms may be obtained. Here, it may be said that the continuous approach makes this possible.

3.3 Applying the interior-point method for planar minimum cost flow

The interior-point method can be used to derive a new algorithm having the best time complexity to some special linear programming problems [13, 30]. In this section we describe an application of the interior-point method to the planar minimum cost flow problem given by Imai and Iwano [13]. Also, some results of preliminary computational experiments are shown.

In these years, much research has been done on the minimum cost flow problem on networks. The minimum cost flow problem is the most general problem in network flow theory that admits strongly polynomial algorithms. Recently, the interior point method for linear programming has been applied to the minimum cost flow problem, which is a very special case of linear programming, by Vaidya [30] from the viewpoint of sequential algorithms and by Goldberg, Plotkin, Shmoys and Tardos [8] from the viewpoint of parallel algorithms. For the planar minimum cost flow problem, the best strongly polynomial time algorithm is given by Orlin [25], and has complexity of $O(m(m+n\log n)\log n)$ for networks with m edges and n vertices.

If we restrict networks, the efficiency of the interior-point method may be further enhanced. Here, we consider the minimum cost flow problem on networks having good separators, that is, $s(n)$-separable networks. Roughly, an $s(n)$-separable graph of n vertices can be divided into two subgraphs by removing $s(n)$ vertices such that there is no edge between the two subgraphs and the number of vertices in each subgraph is at most αn for a fixed constant $\alpha < 1$. A planar grid graph is easily seen to be $\sqrt{n}$-separable [7]. General planar graphs are $O(\sqrt{n})$-separable, which is well known as the planar separator theorem by Lipton and Tarjan [21].

For a system of linear equations $\tilde{A}\tilde{x} = \tilde{b}$ with $n \times n$ symmetric matrix $\tilde{A}$, if the nonzero pattern of $\tilde{A}$ corresponds to that of the adjacency matrix of an $s(n)$-separable graph, the system $\tilde{A}\tilde{x} = \tilde{b}$ can be solved more efficiently than general linear systems. This is originally shown for planar grid graphs by George [7], and the technique is called nested dissection. The technique is generalized to $s(n)$-separable graphs by Lipton, Rose and Tarjan [20]. For A corresponding to a planar graph, the system can be solved in $O(n^{1.188})$ time and $O(n\log n)$ space by further making use of the fast

Table 3.1. Preliminary computational results for the minimum cost flow problem on $k \times k$ grid network

Problem size		Affine-scaling: A		Simplex: S		Time ratio
k	#vertices	#iter.	time (s)	#iter.	time (s)	S/A
20	400	12	2.88	942	0.98	0.34
30	900	11	9.38	2482	4.83	0.51
40	1600	11	23.12	5186	16.12	0.70
50	2500	12	50.92	9298	37.15	0.73
60	3600	12	90.73	14412	78.23	0.86
70	4900	12	149.55	21304	146.48	0.98
80	6400	12	226.77	30532	245.02	1.08
90	8100	12	330.40	38923	382.85	1.16
100	10000	12	461.52	53161	593.52	1.29
110	12100	12	627.15	66526	902.77	1.44
120	14400	12	822.51	86740	1273.25	1.55

matrix multiplication by Coppersmith and Winograd [4].

When applying the interior method, a main step is to solve a linear system of equation to compute a search direction such as the Newton direction as mentioned above. In solving the linear system $\tilde{A}\tilde{x} = \tilde{b}$ for the linear programming problem of the form mentioned in the beginning of the introduction, we mainly have to solve a linear system with a $d \times d$ symmetric matrix $\tilde{A}$ defined by

$$\tilde{A} = A^{\mathrm{T}}DA$$

where D is a diagonal matrix whose diagonal elements are positive. For most interior-point methods proposed so far, their search direction can be computed relatively easily if the linear system for this $\tilde{A}$ is solvable by some efficient method.

In the case of network flow, the main part of the matrix A is the incidence matrix of underlying graph. Suppose A is exactly the incidence matrix. For this A, the nonzero pattern of $\tilde{A} = A^{\mathrm{T}}DA$ exactly corresponds to the nonzero pattern of the adjacency matrix of the same graph.

Therefore, when the network is an $s(n)$-separable graph, the technique of nested dissection can be applied in the main step of the interior-point method. By elaborating this combination of the interior-point method with restart procedure by Vaidya [30] and the nested dissection technique in more detail, for the planar minimum cost flow problem, $O(n^{1.594}\sqrt{\log n}\log(n\gamma))$ time and $O(n \log n)$ space [13], where γ is the largest absolute value among edges costs and capacities represented by integers. Parallel results corresponding to this is mentioned at the end of the preceding subsection.

When γ is polynomially bounded in n, the sequential time complexity becomes $O(n^{1.6})$ which is best among existing algorithms by a factor roughly $O(n^{0.4})$. These results can be generalized to the minimum cost flow problem on $s(n)$-separable networks such as three-dimensional grid networks. Also, in computational geometry, there arise linear programming problems having special structures (e.g., [10]) which may be used to enhance the efficiency in solving the linear system.

In concluding this subsection, we show results of a computational experiment of using the nested dissection technique to the minimum cost flow problem on grid networks. In this experiment, as an interior-point method, the affine-scaling method, which would be the simplest interior-method, is used, and at each iteration the linear system is solved by the nested dissection technique. Table 3.1 gives the results. Also, to compare its result withe some other method, the running time of the network simplex code, taken from a book [17], is also shown. In the simplex method, one iteration corresponds to one pivoting. It should be noted that, in comparing these two methods, simply comparing the running times listed here is meaningless, since these two methods use different stopping criteria and both implementations are not best possible at all. Instead, in the table, it should be observed how the running times of both methods increase as the size of networks increases.

From this table, it is seen that running times of the affine-scaling interior method increases more slowly than that for the network simplex method. This indicates that as the size of linear programming becomes large, the interior-point method becomes relatively more efficient than the simplex method in this case. It is now widely recognized that this tendency mostly holds for general cases.

4 Combining Techniques in Computational Geometry and Mathematical Programming

At the end of section 2, we have seen how some of computation geometric technique can be combined with pure linear programming theory through randomization. In this section, we mention a result by Adler and Shamir [1] which combines a randomized algorithm with an interior-point method. This is really a nice example of combining results in computational geometry and mathematical programming.

The algorithm is basically similar to Clarkson's randomized algorithm [3]. Instead of simple uniform random sampling, it uses self-adjusting weighted random sampling and tries to keep the number of sampled constraints smaller with respect to d. That is, instead of maintaining all the violating constraints in the Clarkson's algorithm, this algorithm increases, for each violating constraint, the probability for the constraint to be sampled in later steps. By this strategy, throughout the algorithm, it simply samples $O(d^2)$ constraints, not $\sqrt{n}$ constraints at each stage.

In the original randomized algorithm, a subproblem is solved recursively by the same randomized algorithm until the size of the subproblem becomes some constant. Instead, any interior-point method may be used to solve the subproblem. Since the size of this subproblem is relatively smaller than the whole size (in fact, the number of constraints in subproblems in the process of this algorithm is bounded by $O(d^2)$ with d variables as mentioned above), the randomized algorithm may gain some merit from this. We skip the details and just mention final results in [1]. Based on the strategy mentioned above, it can be shown that the linear programming problem is solvable in expected time $O((nd + d^4L)d\log n)$. This improves much the time complexity if $d \ll n$, which is quite interesting from the viewpoint of the interior-point method.

This way of combining the interior-point method with a certain computational-geometric technique is still naïve. Besides this algorithm, randomization would be useful in some parts of the interior-point method, and research along this line, combining the continuous approach and discrete approach, would deserve much attention.

5 Concluding Remarks

We have discussed about linear programming from the standpoints of both computational geometry and mathematical programming, and have explain some fruitful results obtained through combining the idea in both fields.

In the introduction, both of discrete and continuous aspects of linear programming are mentioned. Since discrete structures are easier to handle by computers, theoretical computer science has so far pursued discrete side very much. In recent years, there have been proposed many nonlinear approach to discrete optimization problems. As in linear programming, handing nonlinear phenomena in the theory of discrete algorithms and complexity will be required in future. Even in the low dimensional space treated in computational geometry, we have not yet fully understood how to treat the continuous world in a discrete setting, and this type of new problems will deserve further investigation.

References

[1] I. Adler and R. Shamir: A Randomization Scheme for Speeding Up Algorithms for Linear Programming Problems with High Constraints-to-Variables Ratio. *DIMACS Technical Report 89-7*, DIMACS, 1989.

[2] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest and R. E. Tarjan: Time Bounds for Selection. *Journal of Computer and System Sciences*, Vol.7 (1973), pp.448–461.

[3] K. L. Clarkson: A Las Vegas Algorithm for Linear Programming when the Dimension is Small. *Proceedings of the 29th IEEE Annual Symposium on Foundations of Computer Science*, 1988, pp.452–456.

[4] D. Coppersmith and S. Winograd: Matrix Multiplication via Arithmetic Progressions. *Journal of Symbolic Computation*, Vol.9, No.3 (1990), pp.251–280.

[5] G. Dantzig: *Linear Programming and Extensions.* Princeton University Press, 1963.

[6] M. R. Garey and D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[7] A. George: Nested Dissection of a Regular Finite Element Mesh. *SIAM Journal on Numerical Analysis,* Vol.10, No.2 (1973), pp.345–363.

[8] A. V. Goldberg, S. A. Plotkin, D. B. Shmoys and É. Tardos: Interior-Point Methods in Parallel Computation. *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989, pp.350–355.

[9] D. Haussler and E. Welzel: Epsilon-Nets and Simplex Range Queries. *Proceesings of the 2nd Annual ACM Symposium on Computational Geometry*, 1986, pp.61–71.

[10] H. Imai: A Geometric Fitting Problem of Two Corresponding Sets of Points on a Line. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E74, No.4 (1991), pp.665–668.

[11] H. Imai: On the Polynomiality of the Multiplicative Penalty Function Method for Linear Programming and Related Inscribed Ellipsoids. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E74, No.4 (1991), pp.669–671.

[12] H. Imai, K. Kato and P. Yamamoto: A Linear-Time Algorithm for Linear L_1 Approximation of Points. *Algorithmica*, Vol.4, No.1 (1989), pp.77–96.

[13] H. Imai and K. Iwano: Efficient Sequential and Parallel Algorithms for Planar Minium Cost Flow. *Proceedings of the SIGAL International Symposium on Algorithms* (T. Asano, T. Ibaraki, H. Imai, T. Nishizeki, eds.), Lecture Notes in Computer Science, Vol.450, Springer-Verlag, Heidelberg, 1990, pp.21–30.

[14] M. Iri: A Proof of the Polynomiality of the Iri-Imai Method. Manuscript, to be presented at the 14th International Symposium on Mathematical Programming, 1991.

[15] M. Iri and H. Imai: A Multiplicative Barrier Function Method for Linear Programming. *Algorithmica*, Vol.1 (1986), pp.455–482.

[16] N. Karmarkar: A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, Vol.4 (1984), pp.373–395.

[17] J. L. Kennington and R. V. Helgason: *Algorithms for Network Programming.* John Wiley & Sons, New York, 1980.

[18] L. G. Khachian: Polynomial Algorithms in Linear Programming. *Zh. Vychisl. Mat. i Mat. Fiz.*, Vol.20 (1980), pp.51–68 (in Russian); English translation in *U.S.S.R. Comput. Math. and Math. Phys.*, Vol.20 (1980), pp.53–72.

[19] M. Kojima, S. Mizuno and A. Yoshise: A Polynomial-Time Algorithm for a Class of Linear Complementary Problems. *Mathematical Programming*, Vol.44 (1989), pp.1–26.

[20] R. J. Lipton, D. J. Rose, and R. E. Tarjan: Generalized Nested Dissection. *SIAM Journal on Numerical Analysis,* Vol.16, No.2 (1979), pp.346–358.

[21] R. J. Lipton and R. E. Tarjan: A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics,* Vol.36, No.2 (1979), pp.177–189.

[22] N. Megiddo: Linear Programming in Linear Time when the Dimension is Fixed. *Journal of the Association for Computing Machinery*, Vol.31 (1984), pp.114–127.

[23] R. C. Monteiro and I. Adler: Interior Path Following Primal-Dual Algorithms Part II: Convex Quadratic Programming. *Mathematical Programming*, Vo..44 (1989), pp.43–66.

[24] K. Numata and T. Tokuyama: Splitting a Configuration in a Simplex. *Proceedings of the SIGAL International Symposium on Algorithms* (T. Asano, T. Ibaraki, H. Imai, T. Nishizeki, eds.), Lecture Notes in Computer Science, Vol.450, Springer-Verlag, Heidelberg, 1990, pp.429–438.

[25] J. B. Orlin: A Faster Strongly Polynomial Minimum Cost Flow Algorithm. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988, pp.377–387.

[26] V. Pan and J. Reif: Efficient Parallel Solution of Linear Systems. *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, Providence, 1985, pp.143–152.

[27] F. Preparata and M. I. Shamos: Computational Geometry: An Introduction. Springer-Verlag, New York, 1985.

[28] J. Renegar: A Polynomial-Time Algorithm, based on Newton's Method, for Linear Programming. *Mathematical Programming*, Vol.40 (1988), pp.59–93.

[29] T. Tokuyama and J. Nakano: Geometric Algorithms for a Minimum Cost Assignment Problem. *Proceedings of the 7th Annual ACM Symposium on Computational Geometry*, 1991, pp.262–271.

[30] P. Vaidya: Speeding-Up Linear Programming Using Fast Matrix Multiplication. *Proceeding of the 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989, pp.332–337.

[31] V. N. Vapnik and A. Ya. Chervonenkis: On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and Its Applications*, Vol.16, No.2 (1971), pp.264–280.

[32] S. Zhang and M. Shi: On Polynomial Property of Iri-Imai's New Algorithm for Linear Programming. Manuscript, 1988.