INVITED PAPER *Special Section on the 5th Karuizawa Workshop on Circuits and Systems*

Geometric Algorithms for Linear Programming

Hiroshi IMAI[†], *Member*

SUMMARY Two computational-geometric approaches to linear programming are surveyed. One is based on the prune-and-search paradigm and the other utilizes randomization. These two techniques are quite useful to solve geometric problems efficiently, and have many other applications, some of which are also mentioned.

key words: *computational geometry, linear programming, prune-and-search, randomization*

1. Introduction

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

$$\min \boldsymbol{c}^T\boldsymbol{x}$$
$$\text{s.t. } A\boldsymbol{x} \geqq \boldsymbol{b}$$

where $\boldsymbol{c}, \boldsymbol{x} \in \boldsymbol{R}^d$, $\boldsymbol{b} \in \boldsymbol{R}^n$, $A \in \boldsymbol{R}^{n\times d}$, A, $\boldsymbol{b}$, $\boldsymbol{c}$ are given and elements $x_1, \cdots, x_d$ 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. Among many types of linear programming problems, the low-dimensional linear programming is of interest, for example, in computer graphics where 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 may be treated in computational geometry, which is a field in computer science treating geometric problems in a unified way from the viewpoint of algorithms (Preparata and Shamos[(8)]) and has produced many efficient algorithms to construct the convex hull, Voronoi diagram, arrangement, etc. The purpose of this paper is to survey some of fruitful results in this direction.

For linear programming, computational geometry has yielded many linear-time algorithms when the dimension is regarded as a constant. The first linear-time algorithm was developed by making use of the prune-and-search paradigm (Megiddo[(6)]). 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.

Another approach to low-dimensional linear programming is to utilize randomization (Clarkson[(3)]), which is found to be quite powerful tool in computational geometry. Randomization introduces probabilistic behavior in algorithms, and often leads to simple algorithms suitable for implementation.

We will cover these two computational-geometric approaches to linear programming. This paper proceeds as follows. We first explain the prune-and-search paradigm, and its application to the two-dimensional linear programming problem in detail. Two applications of the prune-and-search technique to some special linear programming problems are then mentioned. We also describe randomized algorithms for linear programming. In conclusion, some recent results in this direction are touched upon.

2. Prune-and-Search Paradigm and Its Application to the 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 (Blum, Floyd, Pratt, Rivest and Tarjan[(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

Manuscript received November 16, 1992.
† The author is with the Faculty of Science, The University of Tokyo, Tokyo, 113 Japan.

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 the 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

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

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[6] et al. Since this is simple enough to describe compared with the other method 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:

$$\min\ c_1x_1+c_2x_2$$

$$\text{s.t. } a_{i1}x_1+a_{i2}x_2\geqq a_{i0}\quad(i=1,\cdots,n)$$

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 Fig.1). Here, instead of considering the problem in this general form, we restrict our attention to the following problem.

$$\min\ y$$

$$\text{s.t. } y\geqq a_ix+b_i\quad(i=1,\cdots,n)$$

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 1 depicts this restricted problem of $n=7$.

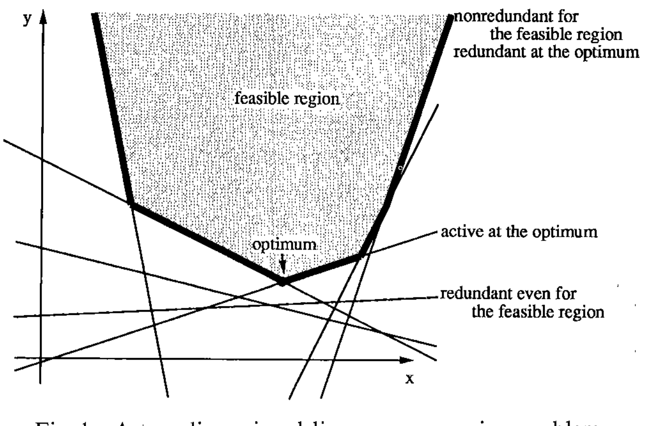

y
x
nonredundant for the feasible region redundant at the optimum
feasible region
active at the optimum
optimum
redundant even for the feasible region

Fig. 1 A two-dimensional linear programming problem.

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

$$f(x)=\max\{a_ix+b_i|i=1,\cdots,n\}.$$

Then, the problem is equivalent to minimizing $f(x)$. The graph of $y=f(x)$ is drawn in bold lines in Fig.1. $f(x)$ has a nice property as follows. First we review the convexity of a function. A function g: $\boldsymbol{R}\rightarrow\boldsymbol{R}$ is convex if

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

for any $x_1, x_2, \lambda\in\boldsymbol{R}$ 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 $\varepsilon>0$,

$$g(x')\leqq g(x'+\varepsilon)\text{ and }g(x')\leqq g(x'-\varepsilon).$$

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 $\varepsilon>0$:
(g1) if $g(x+\varepsilon)<g(x)$, $x\leqq x^*$;
(g2) if $g(x-\varepsilon)<g(x)$, $x\geqq x^*$;
(g3) if $g(x+\varepsilon), g(x-\varepsilon)\geqq g(x)$, x is a global minimum solution.
Finally, for k convex functions $g_i(x)$ $(i=1,\cdots,k)$, a function $g(x)$ defined by

$$g(x)=\max\{g_i(x)|i=1,\cdots,k\}$$

is again convex.

Now, return to our problem of minimizing $f(x)=\max\{a_ix+b_i|i=1,\cdots,n\}$. $f(x)$ is a continuous piecewise linear function. From the above discussions, we have the following.
(f1) $f(x)$ is convex (since a_ix+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 $I=\{i|a_ix+b_i=f(x)\}$, $a^+=\max\{a_i|i\in I\}$ and $a^-=\min\{a_i|i\in I\}$, which can be done in $O(n)$ time; if $a^+>0$, $x^*\leqq x$; if $a^-<0$, $x^*\geqq x$; if $a^+\geqq0$ and $a^-\leqq0$, 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 $y \geqq a_i x+b_i$ and $y \geqq a_j x+b_j$. If $a_i=a_j$, according as $b_i<b_j$ or $b_i \geqq b_j$, we can discard the ith or jth, 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. jth) 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 $(x'-x^*)(x'-x_{ij}) \leqq 0$, 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 prunes a constant fraction of n objects in the d-dimensional space by recursively solving a constant number of sub-problems in the $(d-1)$-dimensional space, thus reducing the size of the problem by a constant factor in the d-dimensional space. In other words, the following characteristics of the pruning technique allow a linear-time algorithm to be devised for 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[(6)] 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 Clarkson[(3)]), but is still exponential with respect to d. We will return to this issue in Sect. 4. Before it, we mention two applications of the prune-and-search technique to larger special linear programming problems.

3. 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[(5)] and the other is on the assignment problem with much fewer demand points than supply points by Tokuyama and Nakano.[(10)]
(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) $(i=1,\cdots, n)$. 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:

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

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

$$\min \sum_{i=1}^{n} c_i$$

$$\text{s.t. } y_i-(ax_i+b)\leqq c_i$$

$$-y_i+(ax_i+b)\leqq c_i$$

Here, x_i, y_i $(i=1,\cdots, n)$ 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[5] 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:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{k} w_{ij}x_{ij}$$

$$\text{s.t. } \sum_{i=1}^{n} x_{ij}=n_j \ (j=1,\cdots, k),\ \sum_{j=1}^{k} x_{ij}=1 \ (i=1,\cdots, n)$$

$$0\leqq x_{ij}\leqq 1$$

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[10] 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

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

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

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

$T(g;\ j)$ $(j=1,\cdots, k)$ 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 can depict an example. Then, a theorem in Ref. (10) 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[7] apply the $(k-1)$-dimensional prune-and-search technique to this geometrically interpreted problem and obtained an $O(((k+1)!)^2n)$-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[10] give a randomized algorithm to solve this problem. We will return to this problem at the end of the next section.

4. 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 Sect.2. 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: $y_0 < a_i x_0 + b_i$.

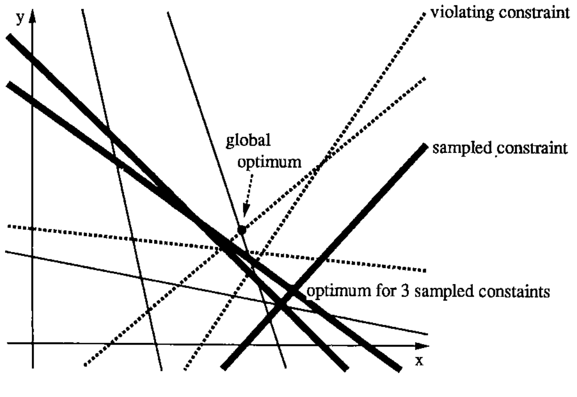


Fig. 2 A randomized algorithm for linear programming.

Figure 2 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 constraints at the global optimum solution, which may be observed by overlaying the two active constraints forming the optimum with the current subset of constraints. In Fig.2, 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 $O(d^2 n + t(d) \log n)$ 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 Haussler and Welzl[4], 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 section. The prune-and-search paradigm yields the $O((k+1)!)^2 n)$-time algorithm for it as mentioned above. Tokuyama and Nakano(10) 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.

5. Concluding Remarks

In this paper we have surveyed two computational-geometric algorithms for linear programming, and described their applications to other problems. Recently, another type of randomized algorithm for linear programming is investigated, and several new nice results are obtained.(1),(9) Applying these techniques to other problems and also combining them with nonlinear approaches such as the interior-point method for linear programming would be interesting as future work.

Acknowledgment

This work was partially supported by the Grant-in-Aid of the Ministry of Education, Science and Culture of Japan.

References

(1) Matoušek, J., Sharir, M. and Welzl, E., "A Subexponential Bound for Linear Programming," *Proceedings of the 8th Annual ACM Symposium on Computational Geometry,* pp. 1-8, 1992.

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

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

(4) Haussler, D. and Welzl, E., "Epsilon-Nets and Simplex Range Queries," *Proceedings of the 2nd Annual ACM Symposium on Computational Geometry,* pp. 61-71, 1986.

(5) Imai, H., Kato, K. and Yamamoto, P., "A Linear-Time Algorithm for Linear L_1 Approximation of Points," *Algorithmica,* vol. 4, no. 1, pp. 77-96, 1989.

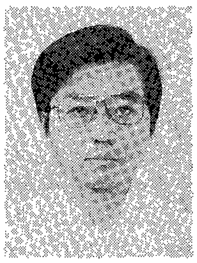
(6) Megiddo, N., "Linear Programming in Linear Time when the Dimension is Fixed," *Journal of the Association for Computing Machinery,* vol. 31, pp. 114-127, 1984.

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

(8) Preparata, F. and Shamos, M. I., *Computational Geometry: An Introduction,* Springer-Verlag, New York, 1985.

(9) Seidel, R., "Low Dimensional Linear Programming and Convex Hulls Made Easy," *Discrete and Computational Geometry,* vol. 6, pp. 423-434, 1991.

(10) Tokuyama, T. and Nakano, J., "Geometric Algorithms for a Minimum Cost Assignment Problem," *Proceedings of the 7th Annual ACM Symposium on Computational Geometry,* pp. 262-271, 1991.



Hiroshi Imai was born in November 21, 1958 at Kobe, Japan. He obtained B. Eng. in Mathematical Engineering, and M.Eng. and D.Eng. in Information Engineering, University of Tokyo in 1981, 1983 and 1986, respectively. In 1986–1990, he was an associate professor of Department of Computer Science and Communication Engineering, Kyushu University. He was also a visiting associate professor at School of Computer Science, McGill University in 1987 and a visiting scientist at IBM T. J. Watson Research Center in 1988. Since 1990, he has been an associate professor at Department of Information Science, University of Tokyo. His research interests include algorithms, computational geometry, and optimization. He is a member of IPSJ, OR Soc. Japan and ACM.