Partial construction of an arrangement of lines and its application to optimal partitioning of bichromatic point set

PAPER Special Section on Discrete Mathematics and Its Applications

Partial Construction of an Arrangement of Lines and Its Application to Optimal Partitioning of Bichromatic Point Set

Tetsuo ASANO† *and* Takeshi TOKUYAMA††, *Members*

SUMMARY This paper presents an efficient algorithm for constructing *at-most-k* levels of an arrangement of n lines in the plane in time $O(nk + n\log n)$, which is optimal since $\Omega(nk)$ line segments are included there. The algorithm can sweep the *at-most-k* levels of the arrangement using $O(n)$ space. Although Everett et al.[7] recently gave an algorithm for constructing the *at-most-k* levels with the same time complexity independently, our algorithm is superior with respect to the space complexity as a sweep algorithm. Then, we apply the algorithm to a bipartitioning problem of a bichromatic point set: For r red points and b blue points in the plane and a directed line L, the figure of demerit $f_d(L)$ associated with L is defined to be the sum of the number of blue points below L and that of red ones above L. The problem we are going to consider is to find an optimal partitioning line to minimize the figure of demerit. Given a number k, our algorithm first determines whether there is a line whose figure of demerit is at most k, and further finds an optimal bipartitioning line if there is one. It runs in $O(kn + n\log n))$ time $(n = r + b)$, which is subquadratic if k is sublinear.

key words: arrangement of lines, clustering, computational geometry, duality transform, topological walk

1. Introduction

The use of duality transform and arrangement of lines is a powerful approach to a number of geometric problems. However, once we decide to construct the entire arrangement of lines the running time must be at least quadratic in the problem size, that is often too expensive. In some cases, however, only partial construction or partial sweeping of the arrangement is sufficient instead of the whole construction. In such cases, if the partial arrangement has subquadratic size and is constructed efficiently, we may overcome the quadratic bound. If the part of the arrangement is the interior of a given convex polygon, several efficient algorithms for the partial construction are known: among all, the topological walk algorithm [2] is optimal with respect to both processing time and working space (i.e. the space complexity except that for storing the output).

This paper presents an algorithm for constructing and sweeping a more complicated part of arrangement in time roughly proportional to its combinatorial complexity, that is, the number of intersections in the portion. More formally, we are interested in constructing the *at-most-k* levels (see Sect. 2 for the definition) of an arrangement of n lines. Our algorithm runs in $O(nk+n\log n)$ time, which is optimal since it is known that the at-most-k levels contain $\Omega(nk)$ line segments and $\Omega(n\log n)$ lower bound is induced from the reduction to the convex hull construction problem. The algorithm is based on the topological walk algorithm, and use $O(n)$ working space.

Then, we apply the algorithm to a bipartitioning problem of a bichromatic point set: For r red points and b blue points in the plane and a directed line l, the figure of demerit $f_d(l)$ associated with l is defined to be the sum of the number of blue points below l and that of red ones above l. Given a number k, we want to find a line l which minimizes the figure of demerit $f_d(l)$ if the associated value of $f_d(l)$ is not greater than k. We present an algorithm which solves the above problem in $O(kn + n\log n)$ time $(n = r + b)$, which is subquadratic if k is sublinear. This problem is related to the *weak separation* problem that finds the parallel belt separating blue and red points completely (i.e. all red points are above the belt and all blue points are below the belt) which minimizes the number of points in the belt. Although Houle [10] gave an optimal $O(n\log n)$ time algorithm for the weak separation problem, no subquadratic algorithm is known for the bipartitioning problem.

Very recently, Everett, Robert and van Kreveld [9] independently studied the same problems as ours, although the algorithms are very different from ours. Their algorithm constructs the at-most-k levels of an arrangement in time $O(nk+n\log n)$ based on the optimal intersection reporting algorithm of Chazelle and Edelsbrunner [3]. Our algorithm uses less working space. Both of our algorithm and the algorithm of Everett et al. include sophisticated data structures or algorithms as their subroutines. However, we have a simpler version of the algorithm running in $O(nk + n\alpha(n)\log n)$ time, where $\alpha(n)$ is the inverse Ackerman function. Everett et al. also present an algorithm for the bipartitioning problem which runs in time $O(nk\log k + n\log n)$, which is inferior to ours.

Manuscript received July 5, 1993.

†The author is with the Faculty of Engineering, Osaka Electro-Communication University, Neyagawa-shi, 572 Japan.

††The author is with the IBM Research, IBM Tokyo Research Laboratory, Yamato-shi, 242 Japan.

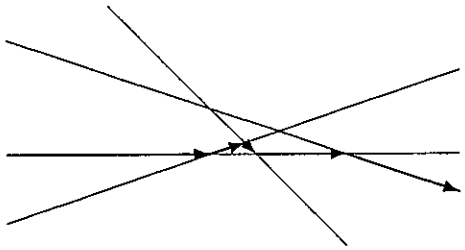

Fig. 1 2-level of an arrangement of 4 lines.

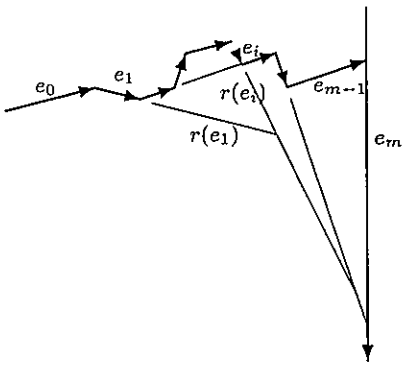

Fig. 2 Definition of rays and lower horizon tree T.

2. Preliminaries

Let $\mathcal{L}$ be a set of n lines in the plane and $\mathcal{A}(\mathcal{L})$ be its arrangement. A k-level of $\mathcal{A}(\mathcal{L})$ consists of a sequence of line segments of $\mathcal{A}(\mathcal{L})$ such that at most $k-1$ lines lie strictly below it and at most $n-k$ lines strictly above it. See Fig. 1 for an example of a k-level in which the 2-level is shown by bold lines.

All i-levels of $\mathcal{A}(\mathcal{L})$ for $1 \leqq i \leqq k$ are referred to as *at-most-k* levels, which is the portion of $\mathcal{A}(\mathcal{L})$ bounded by the 1-level (often called the *lower envelope*) and the k-level [6]. Similarly, at-least-k levels is the portion bounded by the k-level and the n-level. Several complexity results are known [5],[6],[14].

Lemma 1 [5],[13]: For an arrangement of n lines in the plane, the complexity of its k-level is $O(\sqrt{k}n/\log^* k)$, and it is constructed in $O(n\log n + n\sqrt{k}n\log^2 k/\log^* k)$ time.

Lemma 2 [14]: For an arrangement of n lines in the plane, the total complexity of the *at-most-k* levels is $O(kn)$.

Now, the problem is whether the *at-most-k* levels can be constructed in time roughly proportional to its complexity, $O(kn)$. This paper presents such an algorithm, which runs in $O(kn + n\log n)$ time, based on Topological Walk Algorithm proposed by the authors [2].

3. Algorithm for Constructing *at-most-k* Levels

This section describes an algorithm for constructing an *at-most-k* levels of an arrangement of n lines. Given a set of n lines in the plane, we first construct its k-level by the algorithm of Cole, Sharir and Yap [5]. The resulting k-level is a sequence of line segments monotone in the x-direction. Let it be $(e_0, e_1, \cdots, e_{m-1})$. We direct those line segments from left to right (we assume as usual that there is no vertical line). Then, we add an additional vertical line at $x = +\infty$. Obviously the line intersects e_{m-1}. Removing the portion of the vertical line above the intersection with e_{m-1} and the portion of e_{m-1} to the right of the intersection results in another sequence of line segments which ends in the downward vertical half line. Let $Q = (e_0, e_1, \cdots, e_{m-1}, e_m)$ be the resulting sequence.

We construct a tree T incrementally (Fig. 2). Initially, T consists of one edge e_m. Starting with e_m, we traverse the sequence Q to the left in the reverse order $e_m, e_{m-1}, \cdots, e_1, e_0$. At each corner between e_i and e_{i+1} during the traverse, if $e_i \rightarrow e_{i+1}$ is a right turn, we do nothing. Otherwise, we emanate a ray $r(e_i)$ as an extension of e_i to the right starting at a point just beyond the intersection (e_i, e_{i+1}) until it encounters T. Then, we add the ray $r(e_i)$ to T. Note that the ray $r(e_i)$ never intersects segments in Q before hitting T, otherwise the intersection is a left turn on the right of e_i but not contained in T, which is a contradiction.

This operation results in a directed tree T rooted at negative infinity on the half line e_m (see Fig. 2). In the above operation, if a ray $r(e_i)$ first hits another ray $r(e_j)$ which has been added to T so far, the starting point of e_i lies to the left of e_j due to the monotonicity of the sequence $(e_0, e_1, \cdots)$. By the construction of T, the ray $r(e_i)$ does not hit any e_k, $k > i$ before hitting $r(e_j)$. Therefore, the slope of $r(e_i)$ is larger than that of $r(e_j)$. This means that the resulting tree is equivalent to a *lower horizon tree* [7] used in the topological sweep.

It is easily seen that the lower horizon tree T partitions the portion of the arrangement below the k-level into convex regions. Furthermore, as is verified in the literature [2], the depth-first search of the lower horizon tree gives a sequence of regions to visit, which has the following properties.

(1) Every region is visited.

(2) The length of the sequence is at most twice larger than the number of regions.

(3) For each edge (directed from left to right) of the lower horizon tree, the region lying to its right is visited before the one lying to its left (see Fig. 3).

This implies an ordering of the regions to be processed. Actually we construct an arrangement of lines according to the ordering, using Topological Walk algorithm of Asano-Guibas-Tokuyama [2].

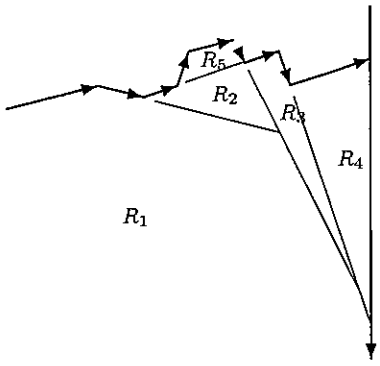

Fig. 3 Partition of the arrangement by the lower horizon tree and the order of visiting regions.

Topological Walk works as follows (we refer to Ref. [2] for details): Let P be a convex polygon and $\mathcal{L}$ be a set of lines in the plane. For each line l of $\mathcal{L}$, we compute its left and right intersections $p_L(l)$ and $p_R(l)$, respectively, with P. The boundary of the convex polygon P can be decomposed into upper and lower chains. Starting with the rightmost vertex of the upper chain, we traverse the boundary of P in the clockwise manner. Each time we encounter a left endpoint $p_L(l)$, we emanate the ray from $p_L(l)$ along the line l toward the interior of P until it hits its boundary or the rays added so far. Once we encounter a right endpoint of some line, we let the lines whose left endpoints appear thereafter wait at the boundary. The above operations result in a tree, which coincides with the upper horizon tree used in Topological Sweep [7]. Then, we implement the depth-first search on the tree while updating the tree at *twigs* (a branch which has two leaf nodes as neighbors). When the portion of the boundary at which some line is waiting is traversed in the depth-first search, the waiting edge is incorporated in the upper horizon tree.

In our case we visit convex regions defined by the lower horizon tree associated with the k-level in the order induced by depth-first search on the lower horizon tree. Since each such region is convex, we can apply Topological Walk to enumerate all intersections in the region. Two versions of Topological Walks are given in Asano-Guibas-Tokuyama [2]: One (we call it Simple Topological Walk) needs no sophisticated data structure, and processes an arrangement interior of a polygon in $O(N + m \log m)$ where N is the complexity of the partial arrangement and m is the number of intersections between the boundary of the polygon and the arrangement. The other (we call it Topological Walk with finger search trees) needs to equip *finger search tree* [12] to the *initial lower horizon tree* [2], and processes the arrangement in $O(N)$ time provided the intersections between the boundary of the polygon and the arrangement are sorted. We use the Topological Walk with finger search tree in our algorithm, although we remark later that we can replace it with Simple Topological Walk with very small increase of the complexity.

As a preprocessing, after constructing the k-level in $O(n \log n + n\sqrt{k} \log^2 k)$ time, we sort all the endpoints on the k-level in $O(\sqrt{kn})$ additional time.

To implement Topological Walk with finger search tree in a region R, we need to sort the left intersections of lines with the bounded boundary of R. The key observation here is that the left endpoint of a line with R is located on the lower convex chain of R if R is a region in the decomposition induced by the lower horizon tree. If there is a left endpoint of a line on the upper chain of R, the slope of the line is smaller than the slope of the intersecting edge of R, thus, R should be cut by the line in the lower horizon tree decomposition. In fact, there is no "waiting edge" mentioned above.

Thus, we only need to enumerate and sort the endpoints on the lower convex chain. The endpoints on the lower convex chain is given from the topological walk in the previous stages (we only need linear time merge operation to obtain the sorted list).

Summarizing the above discussion, we have the following algorithm.

Constructing *at-most-k* Levels:

(Step 1) Construct the k-level using the algorithm by Cole, Sharir and Yap [5].

(Step 2) Build the lower horizon tree T associated with the k-level.

(Step 3) Decompose the region below the k-level into small convex regions by the lower horizon tree T.

(Step 4) Sort all the endpoints on the k-level.

(Step 5) Perform a depth-first search on T to determine the order to visit those convex regions.

(Step 6) According to the order we construct an arrangement of each convex region based on Topological Walk Algorithm.

The steps (1) through (5) are done in $O(n \log n + n\sqrt{k} \log^2 k)$ time. In the last step the portion of the arrangement within each convex region is constructed in time linear in its complexity if we apply Topological Walk with finger search trees. Thus the step is completed in time $O(nk + n \log n)$. The overall time complexity is $O(n \log n + n\sqrt{k} \log^2 k + nk) = O(nk + n \log n)$.

Theorem: Given a set of n lines in the plane and an integer $k \leq n$, the *at-most-k* levels of the arrangement of the lines can be constructed in $O(nk + n \log n)$ time.

It should be noted that although the above algorithm describes only how to enumerate all the intersections it is not so hard to modify it so as to build graph structure to represent the arrangement. Also, we can

label each edge its level during the walk in $O(1)$ additional time. This leads to the decomposition of the arrangement into levels.

The algorithm only needs $O(n\sqrt{k})$ working space (i.e. neglecting the space to store the output), that is better than that of $O(nk + n\log n)$ in Evelett et al.'s algorithm [9]. It is important to reduce the working space in practical implementation of an algorithm, since we sometimes use systems with both small local memory and large main memory such that the access to the main memory is much expensive than that to the local memory.

Moreover, we can reduce the working space complexity to $O(n)$ as follows: We run the Cole-Sharir-Yap's algorithm, which computes the k-th level from left to right. If the algorithm find n vertices of the k-th level, we cut the arrangement by a vertical line through the n-th vertex. Then, we run our algorithm for the part that is to the left of this vertical line. When the part has been swept, we resume the Cole-Sharir-Yap's algorithm until the next n vertices of the k-th level is found. We continue until all the vertices of the partial arrangement are swept. This methods create $O(k\sqrt{k})$ new intersections between the vertical lines and the partial arrangement, but the total time complexity remains $O(nk + n\log n)$. The space complexity becomes $O(n)$.

Finally, (as we promised) we remark that we can use Simple Topological Walk instead of Topological Walk with finger search tree if we may increase the time complexity to $O(nk + n\alpha(n)\log n)$ ($\alpha(n)$ is the inverse Ackerman function) by using a lemma given by Evelett et al. (Lemma 1 in Ref. [9]). Let $left_k(l)$ and $right_k(l)$ be the leftmost intersection and rightmost intersection of a line l in an arrangement $\mathcal{A}$ with the at-most-k level of the arrangement. Let $seg_k(l)$ be the segment between $left_k(l)$ and $right_k(l)$ (it may be a half line or a line). Evelett et al. proved that the part of the arrangement $\mathcal{A}_k$ below the upper envelope of $\{seg_k(l) : l \in \mathcal{A}\}$ has complexity $O(kn)$. Since at-most-k level of $\mathcal{A}$ is a subset of $\mathcal{A}_k$, it suffices to walk in $\mathcal{A}_k$. The upper envelope of a set of n segments have $O(n\alpha(n))$ vertices on it. The Simple Topological Walk needs $O(kn+f(n)\log n)$ time, where $f(n)$ is the complexity of the initial lower horizon tree. Thus, we can walk in $\mathcal{A}_k$ in $O(kn + n\alpha(n)\log n)$ time by using Simple Topological Walk. It is also easy to reduce the working space complexity to $O(n)$ for this version, too.

4. Finding Optimal Partitioning of Bichromatic Point Set

Let R and B be point set of in a plane. We call a point in R a *red point* and that of B a *blue point*. The number of red points is r, and that of blue points is b. Let $n = r + b$.

For a directed line l in the plane, l^+ (resp. l^-) denotes the half plane above (resp. below) of l. Let $R(l^+)$ be the set of red points in l^+, and $r(l^+)$ be the cardinality of it. Similarly, we define $B(l^-)$ and $b(l^-)$. An example is shown in Fig. 4.

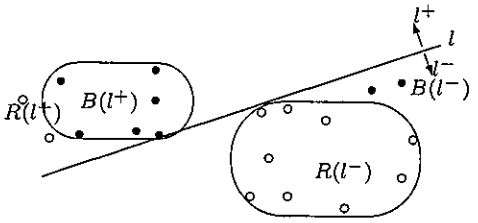

Fig. 4 Partitioning bichromatic point sets. Filled and empty circles represent blue and red points, respectively. $r(l^+) = |R(l^+)| = 2, b(l^-) = |B(l^-)| = 2$, and we have $f_d(l) = r(l^+) + b(l^-) = 4$.

It is well known that we can determine in linear time whether there is a line completely separating point sets. The problem is formulated as a 2-dimensional linear programming problem and thus can be solved by the Megiddo's linear-time algorithm [11]. We define $f_d(l) = r(l^+) + b(l^-)$, and call it the *figure of demerit*. In that case $f_d(l)$, the figure of demerit of the separating line l, is zero. Here we are interested in the case in which there is no such separating line, that is, the two convex hulls have non-empty intersection.

The problems we focus on is the following:

P(1): Optimal separating line finding Find a line l which minimizes $f_d(l) = r(l^+) + b(l^-)$.

In many practical cases the figure of demerit is expected to be small compared with the total number of points. So, the following problem is also meaningful.

P(2): Optimal separating line finding for constraint case Given a number k, find a line l which minimizes $f_d(l)$ if the associated value of $f_d(l)$ is not greater than k.

These problems resembles to the weak separation problem, which finds a pair of parallel lines l and l' which maximize $b(l^+) + r(l'-)$ under the condition that $r(l^+) = b(l'^-) = 0$. Houle [9] gave an $O(n\log n)$ time algorithm for solving weak separation problem. Although the weak separation can be found efficiently, it is sometimes needed to find a separating line instead of a pair of separating lines.

Moreover, the problem P(2) is related to the two dimensional linear programming problem without a feasible solution. Finding a solution which violates least number of constraints of a non-feasible LP is very important in practical applications. Given a red point, (a, b), we consider the half plane $y > ax + b$ of the dual plane. Similarly, a blue point (c, d) is corresponding to the half plane $y < cx + d$. Then, the dual of the problem P(2) is as follows:

P(3): Find the best solution in a non-feasible LP Con-

sider a set of n linear inequalities on two variables. Suppose that there exists no feasible solution of this system of inequalities. Find a point which violates the least number of inequalities if there exists a point violating at most k inequalities.

The problem P(3) can be solved in the same time complexity as P(2). We show below an algorithm for solving P(2) in $O(kn + n\log n)$ time, which is subquadratic if k is sublinear.

We dualize the problem, and consider the arrangement of lines. We assume we have computed the red i-levels for $i = 1, 2, ..., k$ and blue j-level for $j = b-k, b-k+1, ..., b$. We denote $red(i)$ for the red i-level, and $blue(j)$ for the blue j-level.

The algorithm consists of two steps. First one is the initial setting, and the second one is the main routine.

At the initial step, we find the lowest (i.e. the smallest indexed) red level intersecting the $blue(b-k)$. If $red(1)$ is above $blue(b-k)$, we set the level to $red(1)$. If $red(k)$ is below $blue(b-k)$, then we return *false* and exit the algorithm. This level is found in $O(nk)$ time. Without loss of generality, we can assume that the red level is $red(1)$.

Now we start the main routine.

Algorithm:

1. $p = k+1$.
2. For $i = 1$ to k;
3. Find the highest blue level $blue(b-j)$ intersecting $red(i)$ such that $i + j < p$
4. If there is such j as above, set $p = i + j$
5. endfor

The number p above is called the target number. For finding the lowest blue level $blue(b-j)$, we apply the plane sweep method. Notice that, as the output of topological walk, the vertices on each level are sorted with respect to the x coordinate value.

We first locate the left endpoint (at $x = -\infty$) of the red level $red(i)$ in the blue arrangement. Suppose it is just between $blue(b-s)$ and $blue(b-s-1)$. If $s < p-i-1$, we set $t = s$, otherwise we set $t = p - i - 1$. We sweep on $red(i)$ and $blue(b-t)$ from the left until we find an intersection between them. Once we find it, we update t to $t-1$. Then, while keeping the current position in the red level, we sweep the new $blue(b-t)$ from its left end until it goes beyond the current position. Note that the new $blue(b-t)$ never intersects $red(i)$ before the current position since it lies above the old $blue(b-t)$ to the left of the current position. Then, we perform the plane sweep again for $red(i)$ and $blue(b-t)$. We report t as the sweep end.

It is clear that the algorithm correctly finds an optimal partitioning line. The sweep operation concerning $red(i)$ needs the time proportional to the complexity of $red(i)$ and those of $blue(b-t)$ for $t = p-i-1, p-i-2..., t' = p'-i$, where t' is the output of the subroutine, and p' is the target number for the next step (where i is replaced by $i+1$.). Notice that a blue level is used only once in the main routine.

Thus, the complexity of the main routine is $O(kn)$ time. We assumed we have computed at-most-k levels of the arrangement of red lines and at-least-$(b-k)$ levels of the arrangement of blue lines. The at-least-$(b-k)$ levels is converted to the at-most-k levels if we reflect the arrangement with respect to the x axis. Hence, from the result of Sect. 3, the total time complexity for solving P(2) is $O(kn + n\log n)$.

We remark that although both the arrangement of at-most-k levels of the red lines and that of at-least-$(b-k)$ levels of blue lines have complexity $O(kn)$, the union of these partial arrangements can intersects at many points. Thus, the simple plane sweep cannot solve the problem in $O(kn\log n)$ time easily. We can design a simpler $O(kn + n\log n)$ time algorithm which only decide whether there exists a partitioning line whose demerit is at most k. Based on that idea, Everett et al. independently gave an $O(kn\log k + n\log n)$ time algorithm using the binary search on k.

5. Conclusions

In this paper we have presented an optimal algorithm for constructing the *at-most-k* levels of an arrangement of n lines in the plane in time roughly proportional to the complexity of the portion. Then, the algorithm was extended to solve a bipartitioning problem of a bichromatic point set. Given n points in total, the algorithm obtained can answer the question whether there is a line separating the bichromatic point set into two so that the number of points lying in the wrong sides is at most k in time $O(nk + n\log n)$ time and if the answer is "yes" then it outputs an optimal separating line minimizing the number of points lying in the wrong sides in time $O(nk)$. If k is sublinear then this algorithm finds an optimal separating line in subquadratic time. This is important from a theoretical point of view, since once the whole arrangement is constructed quadratic time is needed.

Finally, it is a challenging and practically important problem to design an efficient algorithm for the higher dimensional version of the problem P(3). Even an approximation algorithm should be very useful in many applications.

Acknowledgement

This work was partially supported by Grant in Aid for Scientific Research of the Ministry of Education, Science and Cultures of Japan.

References

[1] Aggarwal, A. and Chandra, A.K., "Virtual Memory Algorithms," *Proc. 20th ACM Symp. on Theory of Computing*, pp.173–185, 1988.

[2] Asano, Te., Guibas, L.J. and Tokuyama, T., "Walking on an Arrangement Topologically," *Proc. 7th ACM Symp. on Computational Geometry*, pp.297–306, 1991.

[3] Chazelle, B.M. and Edelsbrunner, H., "An Optimal Algorithm for Intersecting Line Segments in the Plane," *J. ACM*, vol.39, pp.1–54, 1992.

[4] Chazelle, B.M., Guibas, L.J. and Lee, D.T., "The Power of Geometric Duality," *BIT*, vol.25, pp.76–90, 1985.

[5] Cole, R., Sharir, M. and Yap, C.K., "On k-hulls and related problems," *SIAM J. Comput.*, vol.16, pp.61–77, 1987.

[6] Edelsbrunner, H., *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1986.

[7] Edelsbrunner, H. and Guibas, L.J., "Topologically Sweeping an Arrangement," *J. Comput. Sys. Sci.*, vol.38, pp.165–194, 1989.

[8] Edelsbrunner, H., O'Rourke, J. and Seidel, R., "Constructing Arrangement of Lines and Hyperplanes with Applications," *SIAM J. Comput*, vol.15, pp.341–363, 1986.

[9] Everett, H., Robert, J.-M. and van Kreveld, M., "An Optimal Algorithm for Computing ($\leq k$)-Levels, with Applications to Separation and Transversal Problems," *Proc. 9th ACM Symp. on Computational Geometry*, pp.38–46, 1993.

[10] Houle, M., "Algorithms for weak and wide separation of sets," *Discrete Applied Math.*, vol.45, no.2, pp.139–159, 1993.

[11] Megiddo, N., "Linear-time Algorithms for Linear Programming in R^3 and Related Problems," *SIAM J. Comput.*, vol.12, pp.759–776, 1983.

[12] Mehlhorn, K., *Data Structures and Algorithms 1: Sorting and Searching*, Springer-Verlag, 1984.

[13] Pach, J., Steiger, W. and Szemerédi, E., "An Upper Bound on the Number of Planar k-sets" *Discrete and Comput. Geometry*, vol.7, pp.109–123, 1992.

[14] Welzl, E., "More on k-sets of finite sets in the plane," *Discrete and Comput. Geometry*, vol.1, pp.95–100, 1986.

Tetsuo Asano received the B.S., M.S., and Ph.D. degrees in Engineering from Osaka University in 1972, 1974, and 1977, respectively. He is currently a professor of Osaka Electro-Communication University. His research interest includes Computational Geometry, Discrete Algorithms, Combinatorial Optimization and their applications. Dr. Asano is a member of IEEE, ACM, SIAM, EATCS, IPSJ, and ORS. He is a member of the editorial boards of Int. J. of Computational Geometry and Applications, and Discrete and Computational Geometry.

Takeshi Tokuyama received the B.S., M.S., and Ph.D. degrees in Mathematics from University of Tokyo in 1979, 1981, and 1985, respectively. Since 1986, he has been a researcher of IBM Tokyo Research Laboratory, and currently an advisory researcher. He was assigned to IBM T. J. Watson Research Center from April 1992 to March 1993. His research interests include computational geometry, algorithm theory, combinatorial optimization, discrete mathematics, and their applications. He received a research award from Information Processing Society of Japan in 1992. Dr. Tokuyama is a member of Information Processing Society of Japan, ACM, Mathematical Society of Japan, and Japan SIAM.