Computing 52, 269–279 (1994)

Computing
© Springer-Verlag 1994
Printed in Austria

Optimal Algorithms for Some Intersection Radius Problems

B. K. Bhattacharya*, Burnaby, **S. Jadhav**[†], Kanpur, **A. Mukhopadhyay**[†], Kanpur, and **J.-M. Robert**[‡], Montreal

Received June 1, 1992; revised January 6, 1994

Abstract — Zusammenfassung

Optimal Algorithms for Some Intersection Radius Problems. The intersection radius of a set of n geometrical objects in a d-dimensional Euclidean space, E^d, is the radius of the smallest closed hypersphere that intersects all the objects of the set. In this paper, we describe optimal algorithms for some intersection radius problems. We first present a linear-time algorithm to determine the smallest closed hypersphere that intersects a set of hyperplanes in E^d, assuming d to be a fixed parameter. This is done by reducing the problem to a linear programming problem in a $(d + 1)$-dimensional space, involving $2n$ linear constraints. We also show how the prune-and-search technique, coupled with the strategy of replacing a ray by a point or a line can be used to solve, in linear time, the intersection radius problem for a set of n line segments in the plane. Currently, no algorithms are known that solve these intersection radius problems within the same time bounds.

AMS Subject Classifications: 52.A30, 52.A10

Key words: Intersection radius, prune-and-search, algorithms, complexity, computational geometry.

Optimale Algorithmen für den Durchschnitts-Radius. Wir bezeichnen als Radius des Durchschnitts einer Menge von n geometrischen Objekten im d-dimensionalen Euklidischen Raum E^d Radius der kleinsten abgeschlossenen Hyperkugel, welche einen nichtleeren Durchschnitt mit allen Objekten besitzt. In der vorliegenden Arbeit beschreiben wir optimale Algorithmen zuer Bestimmung einiger solcher Radien. Zuerst stellen wir einen Algorithmus mit linearem Zeitbedarf vor, wenn die Objekte Hyperebenen in E^d mit festem d sind. Er beruht auf der Reduktion des Problems auf eine $(d + 1)$-dimensionale Lineare Optimierungsaufgabe mit $2n$ linearen Nebenbedingungen. Wir beschreiben auch die Lösung des Durchschnitts-Radius Problems für n Strecken in der Ebene. Dazu benutzen wir neben Breitensuche die Ersetzung von Halbstrahlen durch Punkte oder Gerade. Bisher waren keine Algorithmen bekannt, welche diese Probleme in den gleichen Zeitschranken lösen.

1. Introduction

Let S be a finite set of objects in a d-dimensional Euclidean space. The stabbing problem consists of finding an object (the stabber), which intersects each member of S. Typically, the stabber is a line, or a hyperplane, or a disk etc., and S is a set of

* School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada.
[†] Department of Computer Science and Engg., Indian Institute of Technology, Kanpur, 208016, India.
[‡] School of Computer Science, McGill University, 3480 University St., Montreal, P.Q., Canada, H3A 2A7.

points, or line segments, or hyperspheres, or polytopes or any mix of these etc. A survey of some recent results is available in [10].

Recently, new attempts have been made to solve the stabbing problem with more complicated stabbers such as polygons, or to find the best stabber, where the best is one which optimises some measure defined on the class of stabbers in question. Goodrich and Snoeyink [8] presented an $O(n \log n)$ algorithm to find a convex polygon whose boundary intersects each of n parallel line segments. Meijer and Rappaport [13] showed that a perimeter minimizing polygon that intersects each of n parallel line segments can be found in $O(n \log n)$ time. Bhattacharya an Toussaint [3] gave an $O(n \log^2 n)$ algorithm for computing the shortest line segment that intersects a set of n given line segments in the plane. This bound was later improved to $O(n \log n)$ [1].

In this paper, we take our stabber to be a disk. The stabbing problem in this case is also known as the intersection radius problem [9]. When the objects to be intersected are only points, the intersection radius problem is the well-known 1-centre problem [4, 16, 17]. In d-dimensional space, this problem can be solved in time linear in the input size if the parameter d is assumed to be fixed [7, 11].

There is no known algorithm in the literature which solves the intersection radius problem for hyperplanes or line segments. There are, however, some variants of this problem which have been studied by various researchers. As for example, when the stabber is a vertical line segments and the objects to be intersected are lines, the intersection radius problem is simply the Chebyshev approximation of points in the dual space [2, 15]. This problem can easily be solved in linear time by transforming it into a linear programming problem.

In this paper, we present algorithms, that solve the intersection radius problem when the objects to be intersected are hyperplanes in E^d or just line segments in the plane.

The organization of the paper is as follows: Section 2 deals with the intersection radius problem for hyperplanes in E^d. Section 3 discusses the intersection radius problem for line segments in the plane. Conclusions and directions for future work are discussed in Section 4.

2. Intersection Radius Problem for Hyperplanes

A hyperplane H in E^d is defined by the set of points $\{x \in E^d | h' \cdot x + h_{d+1} = 0\}$, where $h' = (h_1, h_2, \ldots, h_d)$ is a non-zero vector normal to H. We will represent a hyperplane by the $(d+1)$-tuple $h = (h_1, h_2, \ldots, h_{d+1})$. Given a $(d+1)$-tuple x, we will let x' stand for the d-tuple formed by taking the first d co-ordinates of x.

The (Euclidean) distance between a point $p \in E^d$ and a hyperplane h is given by

$$\delta(p,h) = \frac{|p \cdot h' + h_{d+1}|}{\|h'\|},$$

where $\|h'\|$ is the (Euclidean) norm of the vector h'.

Let $S = \{h_1, h_2, \ldots, h_n\}$ be a set of n hyperplanes in E^d in canonical form (normal to each h_i has unit norm), where h_i is the $(d+1)$-tuple $(h_{i,1}, h_{i,2}, \ldots, h_{i,(d+1)})$. A hypersphere C intersects the hyperplane h_i if and only if its distance from the centre of C is less than its radius r (i.e. $\delta(c, h_i) \leq r$). Thus the problem of finding the smallest hypersphere that intersects all the members of S can be formulated as the following linear programming problem in E^{d+1}:

Minimize $\quad r$

Subject to $\quad \delta(c, h_i) \leq r$

$\quad r \geq 0$

$\quad c \in E^d$

or,

Minimize $\quad r$

Subject to $\quad c \cdot h_i' + h_{i,d+1} \leq r$

$\quad c \cdot h_i' + h_{i,d+1} \geq -r$

$\quad r \geq 0$

$\quad c \in E^d$

This problem can be solved using the algorithm of Dyer, Megiddo or Clarkson [5, 7, 12]. We could even use a recent simpler and more efficient randomized algorithm of Seidel [14] that runs in $O(n \cdot (d+1)!)$ expected time.

Thus when the objects are hyperplanes in E^d, the smallest intersection radius problem can be solved in linear time, assuming d to be a fixed parameter.

3. Intersection Radius Problem for Line Segments in the Plane

Each line segment can be considered to be the intersection of two oppositely directed rays having the endpoints of the line segments as their respective tails. Thus the intersection radius problem for a set of n line segments can be reduced to the intersection radius problem for a set of $2n$ rays in the plane. The latter problem can be solved by combining the prune-and-search strategy of Megiddo [11] with the novel idea of replacing a ray by a line or a point in the pruning step. We do this by breaking up the problem into the following two subproblems:

Subproblem 1: *Given a set of lines and points in the plane, compute the smallest radius disk that intersects these.*

Subproblem 2: *Given a set of rays, points and lines in the plane, show how a fraction of the rays can be replaced by lines or points such that the intersection radius of the new set is the same as that of the original one.*

We will use the symbols l, p and r to denote a line, a point and a ray respectively, and finite sets of these by L, P and R respectively. The respective cardinalities of these sets are n_l, n_p and n_r. The functions $g_l(x, y)$, $g_p(x, y)$ and $g_r(x, y)$ have the following definitions:

$$g_l(x, y) = \max\{d_l(l_i, (x, y)) \mid l_i \in L\};$$

$$g_p(x, y) = \max\{d_p(p_i, (x, y)) \mid p_i \in P\};$$

$$g_r(x, y) = \max\{d_r(r_i, (x, y)) \mid r_i \in R\},$$

where $d_l(l_i, (x, y))$, $d_p(p_i, (x, y))$ and $d_r(r_i, (x, y))$, respectively, denote the (Euclidean) distance of the point (x, y) from a line l_i, a point p_i and a ray r_i. We define $g(x, y)$ as

$$g(x, y) = \max\{g_l(x, y), g_p(x, y), g_r(x, y)\}.$$

It can be easily shown that the functions $g_l(x, y)$, $g_p(x, y)$ and $g_r(x, y)$ are all convex so that $g(x, y)$ is also convex.

Let S be any finite set of lines, points and rays.

The convexity of this last function enables us to answer two key questions about the minimum radius disk that intersects S. The first is that if we constrain the center of the minimum radius disk to lie on a given line l then to which side of a given point (x, y) on l does the constrained center lie. The second is that given a line l to which side of it does the centre of the minimum radius disk lie. Without any loss of generality, we can take l to be the x-axis of an orthogonal frame of reference.

Let us answer the first question. Clearly, we can compute $g(x, 0)$ for the set S in $O(|S|)$ time. Let S' be the subset of objects whose distance from $(x, 0)$ is $g(x, 0)$ or, more simply, those that touch the disk. Since $g(x, 0)$ is convex, if the contact points of all the objects in S' lie to the left (right) of the vertical line through $(x, 0)$ then the centre of the constrained minimum radius disk lies to the left (right) of $(x, 0)$. Otherwise, $(x, 0)$ itself is the required centre.

The second question is also easily answered. We compute a minimum radius disk whose center is constrained to lie on the line l. If the contact points of the objects in S' span an arc greater than or equal to a semi-circle of the disk-boundary, then the disk found is the required minimum radius disk. Otherwise, the centre lies in the same half space, determined by l, as in which the line segment, joining the mid-point of the chord of the above spanning arc to the disk center, lies. This again follows from the convexity of the function $g(x, y)$.

3.1. Intersection Radius Problem for Points and Lines

In this section we present a linear time algorithm to determine a minimum radius disk that intersects a given set of points and lines. When the objects to be intersected are only points, the algorithm of Dyer or Megiddo [6, 11] for the 1-centre problem can be used. The situation is more complex when lines are also included in the set of objects to be intersected. Our algorithm uses the basic prune-and-search technique of Dyer or Megiddo, referred to above. We describe the technique for lines only. We will then argue that the addition of points does not change the concept underlying the algorithm.

3.1.1. A Constrained Version of the Problem

We first discuss the problem of computing the intersection radius for a set of lines, with the centre of the disk constrained to lie on a line l, assumed to be the x-axis of an orthogonal frame of reference. We also assume that S has at most two lines parallel to l, since any other lying between these two does not change the intersection radius.

Furthermore, out of these two lines we can prune away the one that is closer to l. So there is no loss of generality in assuming that there is at most one line parallel to l. To the remaining n' $(\geq (n-1))$ lines we apply the prune-and-search strategy in the following way.

We first identify a group of roughly half the lines that do not intersect an interval on l which contains the centre of the constrained optimum radius disk. This can be done in linear time as follows.

We compute the median $x_{(m)}$ of the intersections of all the lines with l. Then we determine the value of $g(x_{(m)}, 0)$ at this point and use this to locate the constrained centre on l with respect to $(x_{(m)}, 0)$. If the constrained centre $(x^*, 0)$ lies to the right (left) of this median point, the required half consists of those lines whose intersections lie to the left (right) of the median. We label the lines l_i, $i = 1, 2, \ldots, \lfloor n'/2 \rfloor$. It will suffice to discuss the case in which the constrained centre lies to the right.

Consider the set of line-pairs (l_{2i-1}, l_{2i}), $i = 1, 2, \ldots, \lfloor n'/4 \rfloor$.

For each parallel line-pair it is clear that we can prune away the one that is closer to x_m. The pruning mechanism is non-trivial for non-parallel pairs, since we have to do a finer location of the constrained centre.

Each non-parallel pair (l_{2i-1}, l_{2i}) has an associated pair of angle bisectors. The intersection with l of one of these does not lie between the intersections of the lines themselves (Fig. 1). Let d_i be this intersection point.

As before, we locate the constrained centre with respect to the median $d_{(m)}$ of these intersections. Either $x^* = d_{(m)}$, or we have the following cases:

Case 1: $x^* < d_{(m)}$
Consider a d_i that lies to the right of $d_{(m)}$. Since x^* lies to the right of all the l_i's, the angle bisector that intersects l at d_i, its associated pair, together with the lines that

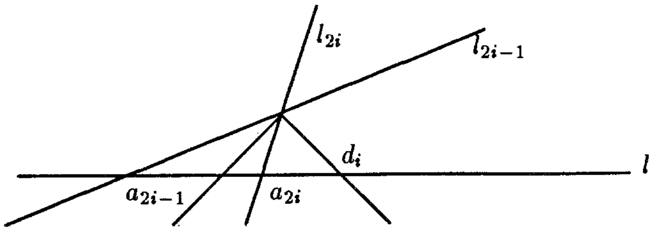

Figure 1. d_i is the intersection of the bisector of l_{2i-1} and l_{2i} which does not lie between a_{2i-1} and a_{2i}

these bisect give rise to the configuration of Fig. 2. Since x^* lies as shown we can prune away one of the lines of the pair, (l_{2i-1}, l_{2i}). Easy arithmetic shows that we can prune roughly one-eighth of the lines we started with.

Case 2: $x^* > d_{(m)}$
It is easy to see that in this case also approximately one-eight of the lines we started with can be thrown away (Fig. 2). ■

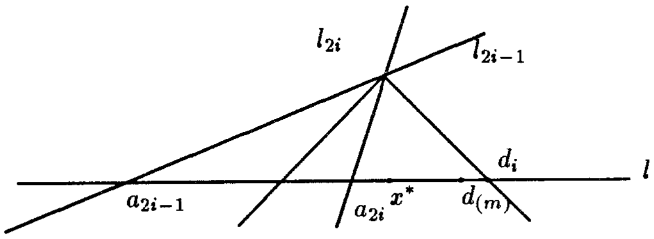

Figure 2. $d_{(m)}$ lies between d_i and x^*

The above pruning takes $O(n)$ time. We repeat this process until no more lines can be pruned, when the optimal solution can be obtained by some brute-force method. If the initial set of lines contained a line parallel to l, the intersection radius is the maximum of the optimal solution obtained and the distance of this parallel line from l.

This is the well-known prune-and-search paradigm of Meggido and Dyer. The total running time of this algorithm is easily seen to be $O(n)$.

If S also contains points, each pruning step is carried out in two substeps. In the first substep we prune points, followed by lines in the second or the other way round. To prune points first, we pair them arbitrarily, and find the median point of the intersections of the bisectors of these pairs with l. We then determine the minimum radius spanning disk C, centred at the median point, for the set S. Next, to determine on which side of the centre of C the constrained centre lies, we examine how the points of tangencies of lines that touch the disk C and the points that lie on its circumference are distributed with respect to the vertical line through the centre of

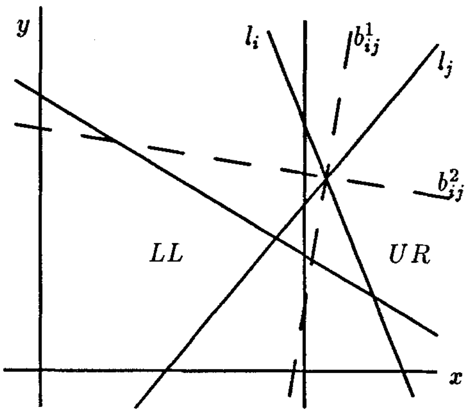

Figure 3. One of the bisectors of l_i and l_j (b_{ij}^1 in the figure) does not intersect the interior of LL

C. When we prune lines next, again points are ignored only in the steps in which we determine x_m and d_m.

In summary, we note that when pruning objects of one kind, the objects of the other kind become transparent whenever we need to find a point on l to serve as the centre of a minimum radius disk for all the objects in the set currently under consideration.

If n_l and n_p be the number of lines and points, respectively, it is easy to see that we prune away at least one-eighth of the total number of objects, viz. $n_l + n_p$. We repeat this process until we cannot prune any more objects. The constrained centre can then be determined by a brute-force algorithm. The whole process requires linear time.

3.1.2. The Unconstrained Centre Problem

Assume that S contains lines only. As before, we will indicate later how to handle the addition of points.

We pair up the lines in S arbitrarily and compute the angle bisectors of each pair. In the degenerate case of a pair of lines being parallel, the angle bisectors reduce to a single line parallel to and equidistant from the lines that make up the pair. When a pair of lines have distinct angle bisectors, these make up an associated pair.

The following observations are crucial. Given a pair of intersecting lines, if we can locate the region containing the centre in a quadrant defined by the angle bisectors of the pair then we can prune away one of the lines. For a pair of parallel lines we can do the same if we can locate this region in a half plane determined by their "angle bisector". We indicate below how we could do this for a fraction of such pairs.

Consider first the subset of vertical bisectors. We locate the unconstrained centre with respect to a median vertical bisector l_1. Assume that it lies in the left half-space L_1 of this median line. Clearly about half of the vertical bisectors do not intersect L_1.

We now compute the median slope of the non-vertical bisectors and pair up arbitrarily in this subset a bisector which has slope greater than the median slope with one which has a smaller slope. The bisectors in each of these pairs necessarily intersect as they have unequal slopes.

Next we compute the intersections with the y-axis of all the unpaired bisectors whose slopes are equal to the median slope. We again locate the centre with respect to a line l_2 which passes through the median of the above intersections and has slope equal to the median slope. Assume that the centre lies in the half plane L_2 below this line. Clearly about half of the bisectors that lie above this line do not intersect L_2.

The centre now lies in $L_1 \cap L_2$. To be able to prune any lines, we need to refine the location of the centre still further as in the worst case we may neither have any vertical bisectors nor any whose slopes are equal to the median slope. Therefore, we do the following with respect to the pairs of intersecting bisectors.

We first locate the centre with respect to a vertical line through the median of the x-coordinates of their intersections. Assume that the centre lies to the left of this line. We now project, parallel to the median slope, the intersection points that lie to right of this line onto the y-axis. We then locate the centre with respect to a line, parallel to the median slope, passing through the median of these projections. Assume that the centre lies below this line so that it lies in the lower left quadrant LL, determined by this line and the earlier one (Fig. 3).

This ensures that at least a fourth of the pairs of bisectors in this class have their intersections in the upper right quadrant UR. Consider one such pair. Since the bisector with slope smaller than the median slope does not intersect LL at least one-eighth of the bisectors whose slopes are not equal to the median slope do not intersect LL. We note that this argument does not depend upon which quadrant the unconstrained centre lies in.

Thus we have a set of bisectors which do not intersect the region $R = LL \cap L_1 \cap L_2$ in which the centre lies (Fig. 4). In the worst case even this set of bisectors does not enable us to prune any lines because there may be no associated pairs of bisectors or bisectors due to a pair of parallel lines in this set. Further, in the worst case, it may happen that all of the associated bisectors of this bunch intersect R.

We need to do one more and final refinement of the location of the centre with respect to these associated bisectors that intersect R.

If we repeat the above steps with the above associated set of bisectors we get a region R', containing the centre and a fraction of these bisectors which do not intersect it.

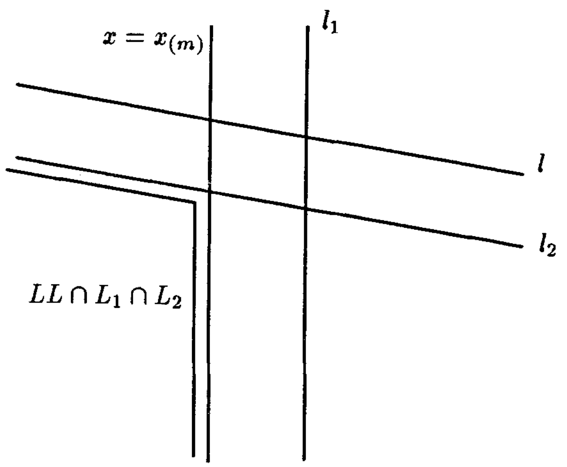

Figure 4. The region $R = LL \cap L_1 \cap L_2$

Each of these, together with its associated pair from the earlier set of bisectors we found that do not intersect R, contain in one of its quadrants the region $R \cap R'$. Region $R \cap R'$ contains the centre. Therefore one of the lines whose angle bisectors these are can be pruned. At the same time we can prune away one of the lines of each parallel pair whose "angle bisector" does not intersect $R \cap R'$.

An easy calculation shows that at least $\lfloor n/64 \rfloor$ of the lines are pruned away.

We repeat this process until no more lines can be discarded. At this stage we use some brute force method to find the true centre. This gives us an algorithm whose time complexity is $O(n)$ in the worst-case.

When points are included in the set of objects to be intersected, we go about the pruning step in exactly the same way as in the constrained case. In each such step we throw away a well-determined fraction of the points an the lines. Repeating this process we get a linear time algorithm for the unconstrained centre problem.

3.2. Intersection Radius for Rays

The constrained problem for a set R of n rays in the plane can be reduced to the problem of solving the same problem for a set of lines and points as detailed below.

For each ray $r_i \in R$, consider the line l_i normal to it and passing through its tail. We compute the median $x_{(m)}$ of the intersections of these normals with l and locate to which side of this median on l the constrained centre lies. Suppose the constrained centre lies to the right of this median. Then for each normal which intersects l to the left of this median point we replace the corresponding ray by a line or point according to the following replacement rule:

If the ray and the median point lie in the same half space of the two half spaces determined by the normal then the ray is replaced by the line which contains the ray. Otherwise, the ray is replaced by its tail point (*Fig.* 5).

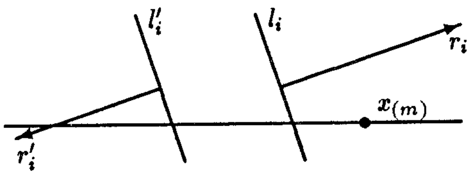

Figure 5. Ray r_i is replaced by the supporting line of r_i, r_i' is replaced by its tail point

It is important to note that neither of these replacements changes the intersection radius of the original set of rays. Thus at least $\lfloor n_r/2 \rfloor$ rays are replaced by either a point or a line and our new set of objects consists therefore of lines, points and rays.

Next, all the points and lines are considered for pruning as described in the Section 3.1. We thus discard a fraction of the rays from further consideration. These two substeps are iterated until no more objects can be discarded, when any brute-force method can be applied to compute the intersection radius. It is easy to check that the algorithm runs in linear time.

To solve the unconstrained version of the intersection radius problem for a set of rays, we need to replace a fraction of the rays by points or lines in linear time. This can be done as follows.

As in the constrained case, we start with the normals through the tails of the rays. Proceeding identically as in the case of the unconstrained problem for lines, we determine a region R which contains the true centre and is not intersected by at least one-eighth of the lines. We do not need to iterate twice as in the case of lines. Only one iteration suffices for the replacement of a fraction of rays. The ray corresponding to each of these lines can therefore be replaced by a line or a point.

Thus in a single iteration we replace about one-eighth of the rays by lines or points. Next we use the two-step pruning process on the set of lines and points generated so far to throw away a fraction of them. Repeating these two substeps on the remaining set of rays we get a linear time algorithm.

4. Conclusions

We have described optimal algorithms for computing the smallest radius sphere which intersects a set of hyperplanes in E^d, d fixed, and the smallest radius disk which intersects a set of line segments in the plane. The approach used for line segments is quite novel. It would be worth investigating whether a similar approach can be used for other kinds of stabbing problems.

Acknowledgements

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada. The authors wish to thank the anonymous referees for their invaluable suggestions which greatly enhanced the readability of the paper.

References

[1] Bhattacharya, B. K., Czyzowicz, J., Egyed, P., Toussaint, G., Stojmenovic, I., Urrutia, J.: Computing shortest transversals of set. In: Proc. of the Seventh Annual ACM Symp. on Computational Geometry, pp. 71–80, 1991.

[2] Brown K. Q.: Fast intersection of half spaces. Technical Report CMU-CS-78-129, Carnegie Mellon University, Pittsburg, 1978.

[3] Bhattacharya, B. K., Toussaint, G. T.: Computing shortest transversals. Technical Report SOCS 90.6, McGill University, April 1990.

[4] Chrystal, G.: On the problem to construct the minimum circle enclosing *n* given points in the plane. In: Proc. Edinberg Math. Soc., *3*, 30–33 (1885).

[5] Clarkson, K. L.: Linear programming in $O(3^{(k+1)^2})$ time. Informa. Proc. Lett. *22*, 21–24 (1986).

[6] Dyer, M. E.: Linear-time algorithm for two- and three-variable linear programs. SIAM J. Computing *13*, 31–45 (1984).

[7] Dyer, M. E.: On a multidimensional search technique and its application to the Euclidean 1-center problem. SIAM J. Computing *15*, 725–738 (1986).

[8] Goodrich, M. T., Snoeyink, J. S.: Stabbing parallel segments with a convex polygon. In: Dehne, F., Sack, J. R., Santaroo, N. (eds) Proc. of the Workshop on Algorithms and Data Structures, pp. 231–242. Berlin Heidelberg New York Tokyo: Springer 1989 (Lecture Notes in Computer Science vol. 382).

[9] Hadwiger, H., Debrunner, H., Klee, V.: Combinatorial geometry in the plane. Toronto: Holt, Rinehart and Winston, 1964.

[10] Houle, M. E., Imai, H., Imai, K., Robert, J. M.: Weighted orthogonal linear L_∞-approximation and applications. In: Dehne, F., Sack, J. R., Santaroo, N. (eds) Proc. of the Workshop on Algorithms and Data Structures, pp. 183–191. Berlin Heidelberg New York Tokyo: Springer 1989 (Lecture Notes in Computer Science, vol. 382).

[11] Megiddo, N.: Linear-time algorithms for linar programming in R^3 and related problems. SIAM J. Computing *12*, 759–776 (1983).

[12] Megiddo, N.: Linear programming in linear time when the dimension is fixed. JACM *31*, 114–127 (1984).

[13] Meijer, H., Rappaport, D.: Minimum polygon covers of parallel line segments. Technical Report CISC 90-279, Queen's University, Canada, 1990.

[14] Seidel, R.: Linear programming and convex hull made easy. In: Proc. of the Sixth Annual ACM Symp. on Computational Geometry, pp. 211–215, 1990.

[15] Shamos, M. I.: Computational Geometry. PhD thesis, Department of Computer Science, 1978.

[16] Sylvester, J. J.: A question in the geometry situation. Q. J. Pure Appl. Math. *1*, 79 (1857).

[17] Sylvester, J. J.: On Poncelet's approximate linear valuation of the surd forms. Phil. Mag. *20*, 203–222 (1860).

Dr. B. K. Bhattacharya
School of Computing Science
Simon Fraser University
Burnaby, BC V56 1S6
Canada

Mr. S. Jadhav
Dr. A. Mukhopadhyay
Department of Computer Science
and Engineering
Indian Institute of Technology
Kanpur 208016
India

Dr. J.-M. Robert
School of Computer Science
McGill University
3480 University Street
Montreal, P.Q., H3A 2A7
Canada