Applications of Parametric Searching in Geometric Optimization *

Pankaj K. Agarwal[†] Micha Sharir[‡] Sivan Toledo[§]

June 17, 2002

Abstract

We present several applications in computational geometry of Megiddo's parametric searching technique. These applications include: (1) Finding the minimum Hausdorff distance in the Euclidean metric between two polygonal regions under translation; (2) Computing the biggest line segment that can be placed inside a simple polygon; (3) Computing the smallest width annulus that contains a given set of given points in the plane; (4) Given a set of n points in 3-space, finding the largest radius r such that if we place a ball of radius r around each point, no segment connecting a pair of points is intersected by a third ball. Besides obtaining efficient solutions to all these problems (which, in every case, either improve considerably previous solutions or are the first non-trivial solutions to these problems), our goal is to demonstrate the versatility of the parametric searching technique.

1 Introduction

In this paper we present several applications in computational geometry of the *parametric searching* technique of Megiddo [34]. This technique, which we briefly review below, is a powerful and ingenious tool for solving efficiently a variety of optimization problems.

*Pankaj Agarwal has been supported by National Science Foundation Grant CCR-91-06514. Micha Sharir has been supported by Office of Naval Research Grant N00014-90-J-1284, by National Science Foundation Grant CCR-89-01484, and by grants from the U.S.–Israeli Binational Science Foundation, the G.I.F. — the German Israeli Foundation for Scientific Research and Development, and the Fund for Basic Research administered by the Israeli Academy of Sciences. Sivan Toledo has been supported by the Special Interdisciplinary Program at Tel-Aviv University.

†Department of Computer Science, Duke University.

‡Courant Institute of Mathematical Sciences, New York University, and School of Mathematical Sciences, Tel Aviv University.

§School of Mathematical Sciences, Tel Aviv University; Current address: Laboratory for Computer Science, Massachusetts Institute of Technology.

Although it has been applied successfully to several problems in computational geometry [1, 2, 3, 4, 20, 21, 36, 38], its potential for problems in geometric optimization does not seem to be widely recognized as yet. Many problems of this kind, which could be easily attacked by the technique, are either solved by more complicated and more ad-hoc techniques, or are simply left unsolved. The purpose of this paper is to present efficient solutions, via the parametric searching technique, to several problems of this kind, with the by-product goal of publicizing the technique and making it more accessible to the computational geometry community.

The parametric searching technique can be described in the following general terms (which are not as general as possible, but suffice for our purposes). Suppose we have a decision problem $\mathcal{P}(d)$ that depends on a real parameter d, and is *monotone* in d, meaning that if $\mathcal{P}(d_0)$ is true for some d_0, then $\mathcal{P}(d)$ is true for all $d < d_0$. Our goal is to find the maximum d for which $\mathcal{P}(d)$ is true (or, if none exists, the supremum of all d for which $\mathcal{P}(d)$ is true). Suppose further that $\mathcal{P}(d)$ can be solved by a (sequential) algorithm $A_s(d)$ whose input is a set of data objects (independent of d) and d, and whose control flow is governed by comparisons, each of which amounts to testing the sign of some low degree polynomial in d. Megiddo's technique then runs A_s "generically" at the unknown maximum $d^\star$. Whenever A_s reaches a branching point that depends on some comparison with associated polynomial $p(d)$, it computes all its roots and runs A_s with the value of d equal to each of these roots. This yields an interval between two adjacent roots, known to contain $d^\star$, and thus enables A_s to determine the sign of $p(d^\star)$, thereby resolving the comparison and allowing the generic execution to proceed. As the algorithm proceeds, the interval known to contain $d^\star$ keeps shrinking as a result of resolving further comparisons, and at the end either the interval becomes a singleton, which is thus the desired $d^\star$, or else $d^\star$ can be shown to be equal to its upper endpoint.

The cost of the procedure just described is generally too high, because the number of times A_s is invoked within the generic execution is proportional to the number of comparisons in the generic A_s. To speed up the execution, Megiddo proposes to replace the generic algorithm by a parallel algorithm A_p. If A_p uses P processors and runs in T_p parallel steps, then each parallel step involves at most P *independent* comparisons. We can then compute the roots of all polynomials associated with these comparisons, and perform a binary search to locate d^* among them using A_s at each binary step. If A_s has running time T_s, then the cost of simulating a parallel step of A_p is $O(P + T_s \log P)$, for a total of $O(PT_p + T_pT_s \log P)$. In most cases the second term dominates the running time. (Since the parallel algorithm is simulated sequentially, we can use the comparison model of Valiant [40], which measures parallelism only in terms of comparisons being made, and ignores all other operations. This observation simplifies the technique considerably.)

This brief overview of parametric searching does not cover all aspects of the technique. Various extensions and variants include a trick due to Cole [20], which in certain cases improves the running time of the procedure by a logarithmic factor, a variant due to Matoušek

[31] and others which replaces in certain applications the parallel generic algorithm by a randomized (sequential) one, leading to simplified solutions, and a variant due to Frederickson and Johnson [24, 25], where the optimal solution $d^\star$ is an element of an implicitly given matrix, whose elements satisfy certain monotonicity properties. There are various other extensions of the technique. For example, Megiddo's subsequent linear-time algorithm for linear programming [35] can be regarded as an optimized variant of the parametric searching technique.

Since its design, about 13 years ago, the parametric searching technique has been successfully applied to a variety of optimization problems. In computational geometry it has been applied to the slope selection problem [21], computing the center of a set of points in 2 and 3 dimensions [36], selecting distances in the plane [1], certain 2-center problems for planar point sets [4], range searching and ray shooting [3], and extremal polygon containment problems [38]. This is still a relatively small crop, given the large body of literature on geometric optimization problems.

In this paper we demonstrate the power of the parametric searching technique by applying it to solve a variety of additional geometric optimization problems. Roughly speaking, the recipe for such an application is first to solve the *fixed-size problem* (i.e. the decision problem $\mathcal{P}(d)$) by an efficient sequential algorithm and an efficient parallel one (in Valiant's model). Then the application of parametric searching is almost routine and yields efficient solution to the related optimization problem.

The problems that we solve in this paper are (see also the subsequent sections for additional discussion of the results and comparison with previous work):

Biggest stick: Computing the longest line segment that can be placed inside a simple n-gon. We present an algorithm with running time $O(n^{8/5+\epsilon})$, for any $\epsilon > 0$,[1] considerably improving the previous algorithm of [18] whose running time is $O(n^{1.9999})$.

Minimum width annulus: Computing the smallest-width annulus that contains a given set of n points in the plane. We give an algorithm with running time $O(n^{8/5+\epsilon})$, significantly improving the quadratic-time algorithm of [23].

Minimum Hausdorff distance between polygons: Finding the minimum Hausdorff distance in the Euclidean metric between two polygonal regions in the plane under translation. This is a hard instance of a general pattern matching problem, studied in [7, 28, 29]. It was left untreated in [29], and was solved by a brute-force inefficient method in [7]. We solve it in time $O((mn)^2 \log^3(mn))$, where m and n are the number of edges of the given polygons. This is about three orders of magnitude faster than the algorithm of [7].

[1] Throughout this paper, ϵ denotes an arbitrarily small positive constant. The meaning of such a complexity bound is that, for any $\epsilon > 0$, the algorithm can be calibrated so that its running time admits the given bound, where the constant of proportionality usually depends on ϵ.

Complete mutual visibility among spheres: Given a set of n points in 3-space, find the largest radius r so that if we place a ball of radius r around each point, the balls are pairwise disjoint, and no segment connecting a pair of points is intersected by a third ball. This problem arises in the context of optical interconnections between processors in 3-space. We present an $O(n^2 \log^5 n)$ algorithm for this problem, which, as far as we know, is the first nontrivial solution.

Although the common theme of our solutions is the application of parametric searching, the bulk of the technical contribution of this paper is in the solutions of the corresponding fixed-size problems, which are by no means easy. They require the application of a variety of sophisticated geometric techniques, such as range searching, point location among algebraic varieties, computing Minkowski sums, and output-sensitive hidden surface removal in 3-space. We also remark that the challenge is not only in solving these fixed-size problems efficiently by a sequential algorithm, but also to design efficient parallel algorithms (in Valiant's model) for these problems.

The paper is organized as follows. We present a solution to the biggest stick problem in Section 2 and to the minimum-width annulus problem in Section 3. Section 4 studies the problem of computing the minimum Hausdorff distance between two polygons, and Section 5 solves the complete mutual visibility problem for spheres in 3-space. We conclude with some final remarks in Section 6.

2 The Biggest Stick Problem

In this section we obtain an improved solution to a problem posed by M. McKenna in 1986: Given a simple polygon P with n edges, find the "biggest stick" (i.e. longest line segment) that can be placed inside P (i.e. be disjoint from the exterior of P). It is easy to design an algorithm for solving this problem in time $O(n^2)$, and the goal is to obtain subquadratic solutions. Chazelle and Sharir [18] have given such a subquadratic solution, which runs in time $O(n^{1.9999})$ and is based on Collins' cylindrical algebraic decomposition technique [22]. The running time of their algorithm can be improved to $O(n^{1.9})$ using the results of Chazelle et al. [14] on point location among algebraic surfaces. In this section we give a considerably improved solution, whose running time is $O(n^{8/5+\epsilon})$. We note that if the endpoints of the stick are constrained to lie at vertices of P then a faster solution is known [6].

Our solution is based on the following approach, also used by the previous algorithms mentioned above. We find, in linear time, a chord e that partitions P into two subpolygons, P_1, P_2, such that each contains at most $2n/3$ vertices [11]. We recursively find the biggest stick in P_1 and in P_2. Then we compute the biggest stick within P which crosses e, and the final answer is the largest of these three candidate sticks. To compute a biggest stick that crosses e we proceed as follows.

Without loss of generality assume that e is vertical and lies on the y axis, and that the right (resp. left) side of e is in P_1 (resp. P_2). Let ρ be a rightward directed ray emanating from e. Using a standard duality transformation, we can map the line supporting ρ to a point ρ^*. We will refer to the point ρ^* as the *dual* of ρ. The duality yields a planar map M_1, each of whose faces is the set of points dual to lines supporting the rays emanating from e and hitting first (the interior of) some fixed edge a of P_1 (i.e., the portion of ρ between e and a lies inside the closed P_1). Every edge g of M_1 is the locus of points dual to the rays that either hit a fixed vertex v of P_1, or touch a vertex v of P_1 before hitting an edge a of P_1, and every vertex of M_1 is a point dual to a ray that either passes through two vertices of P_1 before hitting an edge of P_1, or passes through a vertex and hits another vertex of P_1. We can define a similar map M_2 for P_2. By a result of Chazelle and Guibas [17], each M_i is a convex planar subdivision having $O(n)$ faces, edges and vertices, and it can be computed in $O(n \log n)$ time (actually, in $O(n)$ time, using the recent polygon triangulation algorithm of Chazelle [13]).

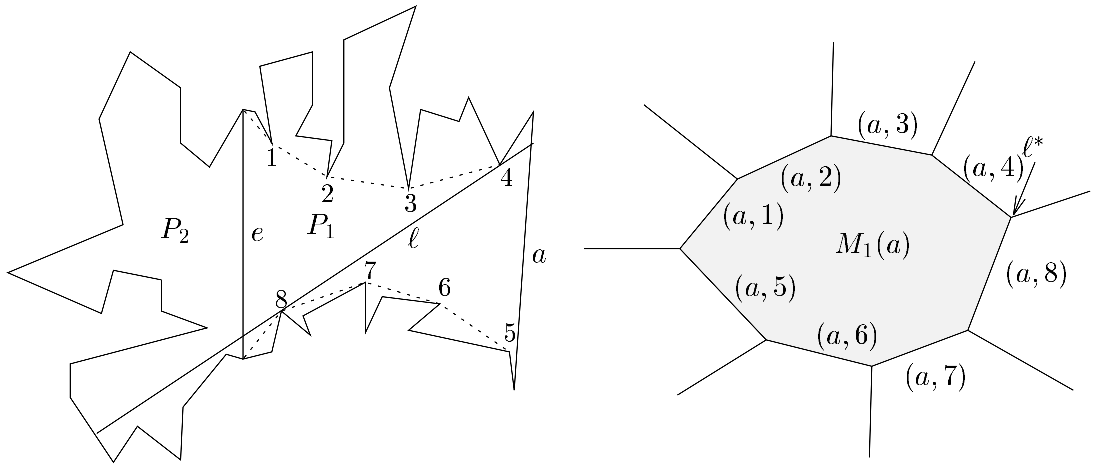

Figure 1: Polygon and its visibility map

It is easy to check that a biggest stick B placed inside P and crossing e must touch two vertices of P. Hence, there are three cases to consider, depending on the vertices $p, q \in P$ that B touches:

1. both p and q are in P_1;
2. both p and q are in P_2;
3. p is in P_1 and q is in P_2.

For each of the three subcases we find the longest segment that can be placed inside P

and crosses e, and then choose the longest of the three segments.

If a segment lying inside P crosses e and touches two vertices p, q of P_1, then the point z dual to the line supporting B is a vertex of M_1. By locating z in M_2 we can determine the length of the longest segment crossing e and passing through p and q, which can be placed inside P. Hence, by locating all vertices of M_1 in M_2, we can determine in $O(n \log n)$ time a longest segment that satisfies the first condition. Similarly, we can determine in $O(n \log n)$ time a longest segment that satisfies the second condition.

The hard case is when p lies in P_1 and q lies in P_2. The point dual to the line supporting such a segment is an intersection point between an edge of M_1 and an edge of M_2. The number of such intersections can be $\Theta(n^2)$ in the worst case, so we cannot afford to compute all of them explicitly. Consequently, we have to use a more clever approach.

Let us fix a length $\delta > 0$, and consider the decision subproblem of determining whether a line segment of length at least δ can be placed inside P such that it touches a vertex of P_1 and another vertex of P_2. We preprocess the edges of one of the maps, say M_2, for efficient range searching queries of a particular kind (detailed below), and then query the resulting structure with range queries derived from the edges of M_1. These queries collectively determine whether there exists a critical placement of the segment with the required properties, thereby solving the fixed-size subproblem.

Recall that every edge g of M_i corresponds to a pair (v, a), where v is a vertex and a is an edge of P_i; the points of g are dual to rays ρ that pass through v and hit a behind (or at) v, so that the portion of ρ between its intersections with e and a, excluding the point v, lies in the interior of P_i.

We regard each edge $g \in M_1$ as the xy-projection of an arc γ in 3-space. For a point $z \in g$, let $\chi(z)$ denote the length of the portion of the line dual to z between its intersections with e and a. Then the height of γ above z, denoted $\gamma(z)$, is equal to $\delta - \chi(z)$. In this manner, the edges of M_1 are mapped into a collection $\mathcal{G}$ of arcs in 3-space.

Next, we map each edge $h \in M_2$, associated with a pair (w, b) of a vertex w and an edge b of P_2, to an arc η in 3-space, so that the xy-projection of η is h and the height of η above any $z \in h$, denoted $\eta(z)$, is the length of the portion of the line dual to z between its intersections with e and with b. Let σ be the intersection of h with an edge g of M_1. Then η passes above γ at σ (i.e., $\eta(\sigma) - \gamma(\sigma) \geq 0$) if and only if the line dual to σ contains a placement of a line segment with length δ crossing e and lying inside P.

The problem has therefore been reduced to the following. Given a collection $\mathcal{G}$ of n arcs in 3-space, corresponding to the edges of M_1 in the manner described above, and a second collection $\mathcal{H}$ of n arcs, corresponding to the edges of M_2, determine whether there exists a pair of arcs, $\gamma \in \mathcal{G}$, $\eta \in \mathcal{H}$, such that η passes above γ.

We solve this problem in two stages. First, following the "hereditary segment tree" technique of [15], we construct, in $O(n \log^2 n)$ time, a family

$$\mathcal{M} = \{ (M_{1,1}, M_{2,1}), (M_{1,2}, M_{2,2}), \dots \}$$

of $O(n \log n)$ canonical pairs of *sets* of edges, such that

1. Each $M_{1,i}$ is a subset of the edges of M_1, and each $M_{2,i}$ is a subset of the edges of M_2.
2. $\sum_i (|M_{1,i}| + |M_{2,i}|) = O(n \log^2 n)$,
3. each edge $g \in M_{1,i}$ intersects every edge of $M_{2,i}$, for each i, and
4. for every pair of intersecting edges $g \in M_1, h \in M_2$, there is a pair $(M_{1,i}, M_{2,i})$ such that $g \in M_{1,i}$ and $h \in M_{2,i}$.

Consider one of the pairs of subsets, say $(M_{1,1}, M_{2,1})$. Since every pair of segments in $(M_{1,1}, M_{2,1})$ intersect, we can extend the segments to full lines without introducing new intersection points. Let $\mathcal{G}_0$ and $\mathcal{H}_0$ denote the sets of arcs (actually curves) corresponding to the lines in $M_{1,1}$ and $M_{2,1}$, respectively; let $|\mathcal{H}_0| = \xi$, $|\mathcal{G}_0| = \zeta$. We want to determine whether any arc of $\mathcal{G}_0$ lies above any arc of $\mathcal{H}_0$.

Recall that each arc $\gamma \in \mathcal{G}_0$ (resp. $\gamma \in \mathcal{H}_0$) is associated with a vertex v_γ and an edge a_γ of the polygon P_1 (resp. P_2), so we can map γ to a point $\hat{\gamma} = (v_\gamma^1, v_\gamma^2, a_\gamma^1, a_\gamma^2)$ in $\mathbb{R}^4$, where $(v_\gamma^1,\ v_\gamma^2)$ are the coordinates of the vertex v_γ and $y = a_\gamma^1 x + a_\gamma^2$ is the equation of the line supporting the edge a_γ (the preceding filtering segment tree technique allows us to ignore the endpoints of a_γ and to regard it as a full line). Let $S = \{\hat{\gamma} \mid \gamma \in \mathcal{G}_0\}$ be the resulting set of ζ points in $\mathbb{R}^4$. We associate a 4-variate function $F_\eta(x_1, x_2, x_3, x_4)$ with each arc $\eta \in \mathcal{H}_0$, such that $F_\eta(\hat{\gamma})$ expresses the difference in height between η and the arc $\gamma \in \mathcal{G}_0$ at the point of intersection between their xy-projections. Let $\ell_{\gamma,\eta}$ denote the line passing through the vertices v_γ and v_η, and let z_γ (resp. z_η) denote the intersection point of $\ell_{\gamma,\eta}$ and the line containing a_γ (resp. a_η); see Figure 2. Then $F_\eta(\hat{\gamma}) = d(z_\eta, z_\gamma) - \delta$. Our goal is thus to determine whether there exists $\gamma \in \mathcal{G}_0$ such that

$$\max_{\eta \in \mathcal{H}_0} F_\eta(\hat{\gamma}) \geq 0\,. \tag{2.1}$$

Let σ_η denote the surface $F_\eta(x_1, x_2, x_3, x_4) = 0$ in $\mathbb{R}^4$; let $\Sigma = \{\sigma_\eta | \eta \in \mathcal{H}_0\}$. Notice that, if we fix η and also fix the first three coordinates $(v_\gamma^1, v_\gamma^2, a_\gamma^1)$ in $\mathbb{R}^4$, the corresponding line $\ell_{\gamma,\eta}$ and the point z_η are fixed, and z_γ is the intersection point of $\ell_{\gamma,\eta}$ and some line with slope a_γ^1. Hence (as long as $a_\gamma^1 \neq \text{slope}(\ell_{\gamma,\eta})$), for each function F_η and any triple (x_1, x_2, x_3) for which these corresponding slopes are distinct, there is a unique $\bar{x}_4 = x_4(x_1, x_2, x_3)$ such that $F_\eta(x_1, x_2, x_3, \bar{x_4}) = 0$, that is, a unique point $(x_1, x_2, x_3, \bar{x}_4)$ on the surface σ_η. Moreover, as x_4 increases beyond $\bar{x}_4$, F_η becomes positive or negative, depending on the values of x_1, x_2, x_3. Assume, for specificity, that P_1 lies to the right of e (in a sufficiently small

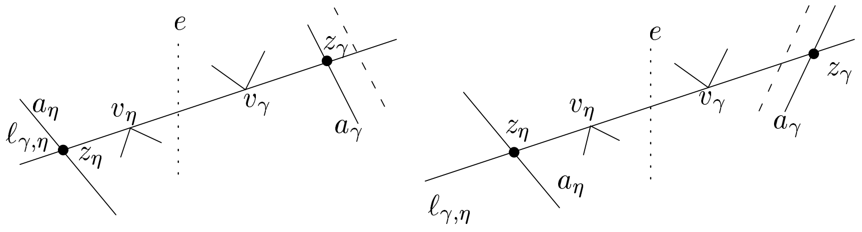

Figure 2: Illustration of $F_\eta(\hat{\gamma})$: (a) slope $(\ell_{\gamma,\eta}) >$ slope(a_γ); (b) slope $(\ell_{\gamma,\eta}) <$ slope(a_γ)

neighborhood of e). If x_3 is greater (resp. smaller) than the slope of the line $\ell_{\gamma,\eta}$, then the value of F_η is negative (resp. positive) for $x_4 > \bar{x}_4$ (see Figure 2).

For each $\eta \in \mathcal{H}_0$, we can thus regard the surface σ_η as the graph of a function $x_4 = \Phi_\eta(x_1, x_2, x_3)$. The function is defined for all (x_1, x_2, x_3) for which the corresponding slope of $\ell_{\gamma,\eta}$ is different from x_3. Property (3) of canonical pairs in $\mathcal{M}$ implies that each Φ_η is defined at the (x_1, x_2, x_3)-coordinates of every point of S.[1] [1]: Verify it!!

The problem at hand can thus be restated as a problem in which we want to determine whether there is any point of S lying on the "good" side of some surface in Σ. However, the problem gets complicated because the good side may be above the surface or below it, depending on the (x_1, x_2, x_3) coordinates of the point and on the surface parameters. Our approach is to first decompose the problem into subproblems, in each of which these good sides are known and fixed, and then solve each subproblem separately and see if any of them yields a good pair of point and surface.

Suppose, as a special case, that

$$\frac{v_\eta^2 - v_\gamma^2}{v_\eta^1 - v_\gamma^1} > a_\gamma^1 \tag{2.2}$$

for each pair $\eta \in \mathcal{H}_0, \gamma \in \mathcal{G}_0$ (i.e., the slope of the line $\ell_{\gamma,\eta}$ is greater than that of the edge a_γ), then $\max_\eta F_\eta(\hat{\gamma}) \geq 0$ if and only if $\hat{\gamma}$ lies above (or on) at least one surface σ_η, that is, $\hat{\gamma}$ does not lie in the (open) bottommost cell $B(\Sigma)$ of the arrangement of the surfaces. In this case the problem of determining whether any arc of $\mathcal{H}_0$ lies above any arc of $\mathcal{G}_0$ thus reduces to determining whether any point of S lies in $B(\Sigma)^c$, where $B(\Sigma)^c$ denotes the complement of $B(\Sigma)$. A similar formulation holds if the inequality (2.2) is reversed for all pairs γ, η. However, for technical reasons to be detailed below, we will require that in each subproblem one of the following stronger conditions is satisfied for all pairs γ, η:

$$\tfrac{v_\eta^2 - v_\gamma^2}{v_\eta^1 - v_\gamma^1} \quad > \max\{a_\gamma^1, a_\eta^1\} \tag{2.3}$$

$$\frac{v_\eta^2 - v_\gamma^2}{v_\eta^1 - v_\gamma^1} < \min\{a_\gamma^1, a_\eta^1\} \tag{2.4}$$

$$a_\gamma^1 < \frac{v_\eta^2 - v_\gamma^2}{v_\eta^1 - v_\gamma^1} < a_\eta^1 \tag{2.5}$$

$$a_\eta^1 < \frac{v_\eta^2 - v_\gamma^2}{v_\eta^1 - v_\gamma^1} < a_\gamma^1 . \tag{2.6}$$

We will show below how to decompose our problem into subproblems of these types. But first we present an efficient solution of a subproblem in which the condition (2.3) holds for all γ and η; the other cases can be handled in a fully symmetric fashion.

First Algorithm

Let Σ and S be two collections of ξ surfaces and ζ points as defined above, so that they satisfy condition (2.3). Let r be some sufficiently large constant. We compute a $\frac{1}{r}$-net $R \subseteq \Sigma$ of size $t = O(r \log r)$ in linear time.[2] If there is a pair $\hat{\gamma} \in S$ and $\sigma \in R$ such that $\hat{\gamma}$ lies above (or on) σ, then we already know that $\max_\eta F_\eta(\hat{\gamma}) \geq 0$, so we can stop right away. Otherwise, we decompose the bottommost cell $B(R)$ below the lower envelope of R into constant-size cells as follows. Fix a surface σ_i of R. For every other surface $\sigma_j \in R - \{\sigma_i\}$, we project the intersection surface $\sigma_i \cap \sigma_j$ orthogonally onto the hyperplane $x_4 = 0$. Let Σ_i^* denote the set of resulting $t-1$ surfaces in $\mathbb{R}^3$. We decompose the arrangement of Σ_i^* into a family Δ_i of $O(t^3\beta(t)) = O(r^3 \log^3 r\, \beta(r))$ cells, using the algorithm of [14], where $\beta(.)$ is an extremely slowly growing function depending on the maximum degree of the surfaces in Σ_i^* (which is a constant) and on the inverse Ackermann function.

For each cell $\tau \in \Delta_i$, let

$$\tau' = \{\, (x_1, x_2, x_3, x_4) \mid (x_1, x_2, x_3) \in \tau \text{ and } x_4 < \Phi_i(x_1, x_2, x_3) \,\},$$

where Φ_i is the function that represents the surface σ_i. If τ' does not intersect any surface of R, we add τ' to the final decomposition of $B(R)$. Repeating the above step for all surfaces in R gives a decomposition Δ of $B(R)$. Since each cell of Δ_i contributes at most one cell to Δ, $|\Delta| = O(r^4 \log^4 r\, \beta(r))$.

For each cell $\tau \in \Delta$, we compute Σ_τ, the set of surfaces in Σ intersecting the interior of τ, and S_τ, the set of points of S lying in τ. Thus, we obtain $O(r^4 \log^4 r\, \beta(r))$ subproblems, where the subproblem corresponding to a cell $\tau \in \Delta$ involves Σ_τ and S_τ. Let $\mathcal{H}_\tau$ and $\mathcal{G}_\tau$ denote the set of arcs corresponding to Σ_τ and S_τ, respectively, and let $\xi_\tau = |\Sigma_\tau|$, $\zeta_\tau = |S_\tau|$.

[2] Specializing from the general concept, we call a subset $R \subseteq \Sigma$ of a set of n (algebraic) surfaces a $\frac{1}{r}$-net, $r < n$, if every (open) cell of constant complexity, of the form obtained in the stratification algorithm of [14], which does not intersect any surface of R, intersects at most n/r surfaces of Σ; see [27] for a more formal definition. Haussler and Welzl [27] showed that a random subset of Σ of size $O(r \log r)$ is a $\frac{1}{r}$-net with high probability. Later Matoušek [32] gave an $O(nr^{O(1)})$-time deterministic algorithm for computing a $\frac{1}{r}$-net of size $O(r \log r)$.

Note that $\sum_\tau \zeta_\tau = \zeta$ and $\xi_\tau \le \xi/r$. We solve each subproblem recursively. The recursion stops when $\xi_\tau \ge \zeta_\tau^4$. In this case, we flip the roles of Σ_τ and S_τ. We map the arcs of $\mathcal{H}_\tau$ to a set $\Sigma_\tau^* = \{\hat\eta \mid \eta \in \mathcal{H}_\tau\}$ of points, and the arcs of $\mathcal{G}_\tau$ to a set $S_\tau^* = \{\sigma_\gamma \mid \gamma \in \mathcal{G}_\tau\}$ of surfaces. We also reverse the direction of the x_4-axis.

Since P_2 lies to the left of the chord e, by (2.3) and the above discussion, there is a pair $\gamma \in \mathcal{G}_\tau$, $\eta \in \mathcal{H}_\tau$ such that $F_\gamma(\hat\eta) \ge 0$ if and only if $\hat\eta$ lies above the surface σ_γ (with the reversed direction of the x_4-axis). We therefore proceed as above — recursively applying the decomposition technique to determine whether any point of Σ_τ^* lies above the bottommost cell $B(S_\tau^*)$. But we now continue the recursion until we are left with constant number of points or surfaces (in which case we use any naive brute-force algorithm), and do not flip the surfaces and points any more.

We now analyze the running time of the above procedure. Let $T(\xi,\zeta)$ denote the maximum running time of the procedure involving ξ surfaces and ζ points. First consider the 'bottom part' of the procedure (after flipping the roles of $\mathcal{G}_\tau$ and $\mathcal{H}_\tau$). We get the following recurrence

$$T(\zeta_\tau,\xi_\tau) = \sum_{j=1}^{cr^4\log^4 r\beta(r)} T\left(\frac{\zeta_\tau}{r}, \xi_{\tau_j}\right) + O(\xi_\tau + \zeta_\tau)\,,$$

where $\sum_j \xi_{\tau_j} = \xi_\tau$ and c is some constant. $T(\zeta_\tau,\xi_\tau) = O(\xi_\tau + \zeta_\tau)$ if ξ_τ or ζ_τ is less than some fixed constant. Following the same analysis as in [14], one can show that the solution of the above recurrence is $O(\zeta_\tau^{4+\epsilon} + \xi_\tau \log \zeta_\tau) = O(\xi_\tau^{1+\epsilon})$ time, since $\xi_\tau \ge \zeta_\tau^4$.

Next, for the top part of the procedure, we get the following recurrence.

$$T(\xi,\zeta) = \begin{cases} \displaystyle\sum_{j=1}^{cr^4\log^4 r\beta(r)} T\left(\frac{\xi}{r}, \zeta_{\tau_j}\right) + O(\xi+\zeta) & \text{if } \xi < \zeta^4 \\ O(\xi^{1+\epsilon}) & \text{if } \xi \ge \zeta^4\,, \end{cases}$$

where $\sum_\tau \zeta_\tau = \zeta$ and c is some appropriate constant. The solution of the above recurrence is (see e.g. [5])

$$T(\xi,\zeta) = O(\xi^{4/5+\epsilon}\zeta^{4/5} + \xi^{1+\epsilon} + \zeta^{1+\epsilon}) \tag{2.7}$$

(where ϵ is a function of r, and can be made arbitrarily small by increasing r).

We leave it to the reader to verify that appropriately modified variants of this procedure will correctly handle any subproblem satisfying one of the other conditions (2.4)–(2.6), with the same bound on their running time.

Second Algorithm

Next, we extend the algorithm to the general case, where none of (2.3)–(2.6) hold uniformly for all pairs of arcs. Essentially, the extended algorithm decomposes the problem into subproblems, each satisfying one of these conditions, and then applies the previous algorithm to each subproblem separately.

For an arc $\eta \in \mathcal{G}_0 \cup \mathcal{H}_0$, let h_η denote the surface

$$(x - v_\eta^1) \cdot \left[y - a_\eta^1(x - v_\eta^1) - v_\eta^2\right] = 0\,, \tag{2.8}$$

and let ψ_η denote the surface

$$(x - v_\eta^1) \cdot \left[y - z(x - v_\eta^1) - v_\eta^2\right] = 0 \tag{2.9}$$

in $\mathbb{R}^3$. We compute a $\frac{1}{r}$-net $R \subseteq \mathcal{H}_0$ of size $O(r \log r)$ (r is a sufficiently large constant), and decompose the arrangement $\{h_\eta, \psi_\eta \mid \eta \in R\}$ in $\mathbb{R}^3$ into a family Δ of $O(r^3 \log^3 r\, \beta(r))$ constant-size cells using the algorithm of [14]. We associate with each cell $\tau \in \Delta$ a subset $\mathcal{H}_\tau \subseteq \mathcal{H}_0$ and another subset $\mathcal{G}_\tau \subseteq \mathcal{G}_0$. An arc $\eta \in \mathcal{H}_0$ is in $\mathcal{H}_\tau$ if either h_η or ψ_η intersects τ, and an arc $\gamma \in \mathcal{G}_0$ is in $\mathcal{G}_\tau$ if the point $(v_\gamma^1, v_\gamma^2, a_\gamma^1)$ lies in τ. Any arc $\eta \in \mathcal{H}_0 - \mathcal{H}_\tau$ is such that each of the left-hand sides of (2.8), (2.9) has a fixed sign over all points $(x, y, z) \in \tau$. We decompose $\mathcal{H}_0 - \mathcal{H}_\tau$ into four subsets, denoted $A_\tau^{(\delta_1,\delta_2)}$, for $\delta_1, \delta_2 \in \{-1, +1\}$, where $\eta \in A_\tau^{(\delta_1,\delta_2)}$ if the corresponding fixed signs of (2.8), (2.9) are δ_1, δ_2, respectively.

We now have to determine, for each cell $\tau \in \Delta$, whether there is an arc η either in one of the sets $A_\tau^{(\delta_1,\delta_2)}$ or in $\mathcal{H}_\tau$ such that $\max_{\gamma \in \mathcal{G}_\tau} F_\eta(\hat{\gamma}) \geq 0$. Notice that, by construction, each of the four pairs $(A_\tau^{(\delta_1,\delta_2)}, \mathcal{G}_\tau)$ satisfy one of the conditions (2.3)–(2.6), so we can use the first algorithm to determine whether any arc of $\mathcal{G}_\tau$ lies above any arc of $A_\tau^{(\delta_1,\delta_2)}$. We repeat this step for all cells $\tau \in \Delta$. By (2.7), the total time spent is $O(\xi^{4/5+\epsilon}\zeta^{4/5+\epsilon} + \xi^{1+\epsilon} + \zeta^{1+\epsilon})$ (since r is a constant).

Next, we recursively solve the $O(r^3\beta(r))$ subproblems, where the subproblem corresponding to a cell τ requires determining whether any arc of $\mathcal{H}_\tau$ lies above any arc of $\mathcal{G}_\tau$. The recursion stops when $\xi_\tau \geq \zeta_\tau^4$. As in the previous algorithm, we now flip the roles of $\mathcal{G}_\tau$ and $\mathcal{H}_\tau$ — we choose a $\frac{1}{r}$-net of $\mathcal{G}_\tau$ of size $O(r \log r)$ and map the arcs of $\mathcal{H}_\tau$ to points, and continue as above. Following a similar argument, the total time spent by the algorithm after flipping the roles of $\mathcal{G}_\tau$ and $\mathcal{H}_\tau$ is $O(\xi_\tau^{1+\epsilon})$. Therefore, we get the following recurrence

$$T(\xi, \zeta) = \begin{cases} O(\xi^{4/5+\epsilon}\zeta^{4/5} + \xi^{1+\epsilon} + \zeta^{1+\epsilon}) + \displaystyle\sum_{i=1}^{cr^3 \log^3 r \beta(r)} T\left(\frac{\xi}{r}, \zeta_i\right) & \text{if } \xi < \zeta^4 \\ O(\xi^{1+\epsilon}) & \text{if } \xi \geq \zeta^4\,, \end{cases}$$

where $\sum_\tau \zeta_\tau = \zeta$ and c is some appropriate constant. The solution of this recurrence is also

$$T(\xi, \zeta) = O(\xi^{4/5+\epsilon}\zeta^{4/5} + \xi^{1+\epsilon} + \zeta^{1+\epsilon})\,. \tag{2.10}$$

We apply the entire procedure to all canonical pairs $(M_{1,i}, M_{2,i})$ in $\mathcal{M}$. By (2.10) and the second property of canonical pairs, we can conclude that the running time of the overall algorithm is $O(n^{8/5+\epsilon'})$ for a slightly larger, but still arbitrarily small $\epsilon' > 0$. Putting everything together, we obtain an algorithm with $O(n^{8/5+\epsilon})$ time, to determine whether a line segment of length δ can be placed inside a given polygon P so that it crosses the diagonal e.

Finding the longest segment

Finally, we apply parametric searching to turn the preceding algorithm into one that computes the biggest segment that can be placed inside P and crosses the diagonal e. To this end, we need an efficient parallel version of the above procedure in Valiant's model [40]. Notice that canonical subsets can be computed sequentially, because the properties (1)–(4) do not depend on the value of δ. So, we only have to parallelize the procedure that, given two collections of arcs $\mathcal{G}_0$ and $\mathcal{H}_0$, determines whether any arc of $\mathcal{G}_0$ lies above any arc of $\mathcal{H}_0$. This is easy to do in Valiant's model. Consider the first algorithm. Since r is chosen to be some constant, the set R can be computed in polylogarithmic -2- time using $O(n)$ processors as described in [16], and the sets Σ_τ and S_τ can be computed in constant parallel time using a linear number of processors. We then obtain a collection of independent subproblems, which can all be processed in parallel. The running time of this procedure is easily seen to be logarithmic and the number of processors can be bounded by the same recurrence as for the sequential running time. The second algorithm can also be parallelized similarly. The only difference is that we now invoke the first procedure at every stage, which will take logarithmic time, so the second algorithm runs in polylogarithmic time with $O(\xi^{4/5+\epsilon}\zeta^{4/5} + \xi^{1+\epsilon} + \zeta^{1+\epsilon})$ number of processors. Omitting some of the (rather routine) details related to parallelization, and plugging all of this into the parametric searching paradigm, we obtain an algorithm with $O(n^{8/5+\epsilon})$ time, for computing the longest segment that can be placed inside P and that crosses the diagonal e.

[2]: Micha! WHat is the exponent??

Going back to our original divide-and-conquer algorithm for finding the biggest stick, the merge step requires $O(n^{8/5+\epsilon})$ time. Hence, we can conclude

Theorem 2.1 *Given a simple polygon P with n edges, one can compute, in time $O(n^{8/5+\epsilon})$, a longest line segment that can be placed inside P.*

3 The Minimum Width Annulus Problem

Next we consider the problem of approximating a planar point set S, of n points, by a circle. One way of obtaining such an approximation is to compute two concentric circles C_1 and C_2 of radii $r_1 < r_2$ such that all points of S lie in the exterior of C_1 and in the interior of C_2, and such that $r_2 - r_1$ is minimized (see Figure 3). In other words, we wish to compute an *annulus* of minimum width that contains all points of S. An $O(n^2)$ algorithm was proposed by Ebara et al. [23]. We present an algorithm whose running time is $O(n^{8/5+\epsilon})$. As it turns out, this application is a variant of the technique used above for the biggest stick problem.

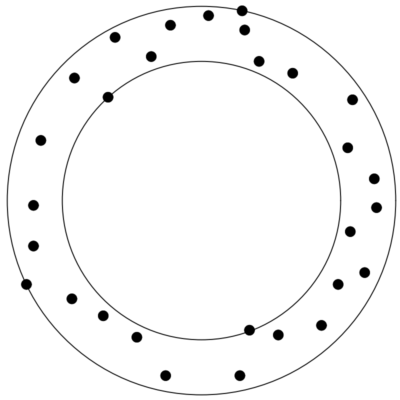

Figure 3: Minimum width annulus

Specifically, let $\mathrm{Vor}_c(S), \mathrm{Vor}_f(S)$ be the closest and the farthest-point Voronoi diagrams of S, respectively. For a point $\xi \in \mathbb{R}^2$ lying in the Voronoi cell $V_c(p_i)$ ($p_i \in S$) of $\mathrm{Vor}_c(S)$, let $D_c(\xi)$ denote the distance between ξ and p_i. Analogously, define $D_f(\xi)$ for $\mathrm{Vor}_f(S)$. Given a point ξ in the plane, the width of the thinnest annulus centered at ξ, which covers S, is $D_f(\xi) - D_c(\xi)$. Thus, our goal is to compute

$$\min_{\xi \in \mathbb{R}^2} D_f(\xi) - D_c(\xi) .$$

As in the case of the biggest stick problem, it suffices to describe an algorithm that, for a given parameter W, can determine whether

$$\min_{\xi \in \mathbb{R}^2} D_f(\xi) - D_c(\xi) \leq W . \tag{3.1}$$

It has been shown in [23] that the desired minimum is attained either at a vertex of one of the two Voronoi diagrams or at an intersection of two diagram edges. By preprocessing $\mathrm{Vor}_c(S), \mathrm{Vor}_f(S)$ for efficient planar point location queries, and by locating each vertex of either diagram in the other diagram, we can test in $O(n \log n)$ time whether any vertex of the two diagrams satisfies (3.1). The hard part is testing the (up to quadratically many) intersection points of edges of the two diagrams, which can be done following the same approach as in the previous section. We will sketch the general idea and leave it for the reader to fill in the details.

Let R denote the set of edges of $\mathrm{Vor}_c(S)$ and let B denote the set of edges of $\mathrm{Vor}_f(S)$. First, we decompose R and B, in $O(n \log^2 n)$ time, into a family of $O(n \log n)$ canonical pairs $(R_1, B_1), (R_2, B_2), \ldots$, which staisfy the properties 1–4 stated in the previous section. Consider one of the canonical pairs (R_i, B_i). Each edge γ of a Voronoi diagram is a portion of a perpendicular bisector of two points $p_\gamma, q_\gamma \in S$, so we can map it to a point $\hat{\gamma} = (v_\gamma^1, v_\gamma^2, m_\gamma, \delta_\gamma)$, where (v_γ^1, v_γ^2) are the coordinates of the midpoint v_γ between p_γ and q_γ, $m_\gamma \geq 0$ is the slope of γ, and $\delta_\gamma \geq 0$ is the distance between p_γ and v_γ. Let $\mathcal{P} = \{\hat{\gamma} \mid \gamma \in R_i\}$. We associate with each edge $\eta \in B_i$ a 4-variate function $F_\eta(x_1, x_2, x_3, x_4)$, such that $F_\eta(\hat{\gamma})$ is equal to

$$D_c(\xi) - D_f(\xi) + W = d(\xi, p_\gamma) - d(\xi, p_\eta) + W ,$$

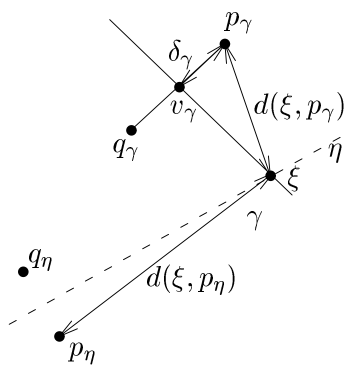

Figure 4: Illustration of $\hat{\gamma}$ and $F_\eta(x_1, x_2, x_3, x_4)$

where p_η is any one of the two points of S defining η, and where ξ is the intersection point of η and γ. For a fixed edge $\eta \in B_i$ and a fixed triple $(v_\gamma^1, v_\gamma^2, m_\gamma)$, the line containing the corresponding γ, and the midpoint v_γ, are both fixed; hence, the intersection point ξ, and the distance $d(\xi, p_\eta)$ are also fixed. It follows that there is at most one $\bar{\delta}_\gamma = x_4(v_\gamma^1, v_\gamma^2, m_\gamma)$ such that $F_\eta(v_\gamma^1, v_\gamma^2, m_\gamma, \bar{\delta}_\gamma) = 0$. Moreover, F_η is positive (resp. negative) for $x_4 > \bar{\delta}_\gamma$ (resp. $x_4 < \bar{\delta}_\gamma$). Hence, $D_f(\xi) - D_c(\xi) \leq W$ for an intersection point ξ of R_i and B_i if and only if there is a point of $\mathcal{P}$ that does not lie below the lower envelope of the surfaces $\{F_\eta(x_1, x_2, x_3, x_4) = 0 | \eta \in B_i\}$. Notice that one can flip the roles of R_i and B_i and still reduce the problem to locating a collection of points below the lower envelope of a collection of surfaces in $\mathbb{R}^4$. Therefore the first algorithm of the previous section can be adapted to solve the above problem in sequential time $O(n^{8/5+\epsilon})$, or in $O(\log n)$ parallel time with $O(n^{8/5+\epsilon})$ processors. Plugging all this into the parametric searching paradigm, we obtain

Theorem 3.1 *Given a set S of n points in the plane, one can compute, in time $O(n^{8/5+\epsilon})$, a minimum width annulus that contains all points of S.*

Remark 3.2: An annulus of minimum area can be computed in linear time using Megiddo's linear programming algorithm.

4 Minimum Hausdorff Distance Between Polygonal Objects

In this section, we consider the following problem: "Let $\mathcal{P}$ be a collection of m objects in the plane and $\mathcal{Q}$ another collection of n objects in the plane. We wish to compute a translation t of $\mathcal{Q}$ which minimizes the *Hausdorff distance* between $\mathcal{P}$ and the translated copy of $\mathcal{Q}$." The Hausdorff distance between two sets A and B of objects is defined as

$$H(A,B) = \max\{h(A,B), h(B,A)\},$$

where

$$h(A,B) = \max_{p \in \cup A} \min_{q \in \cup B} d(p,q)$$

(we assume that the objects of A and B are all compact sets, so the minima and maxima appearing in this formula are all well defined). Here $d(\cdot,\cdot)$ denotes the Euclidean distance between two points. For a set $\pi \subseteq \mathbb{R}^2$ and a vector t, let $\pi \oplus t = \{p + t \mid p \in \pi\}$ be the *Minkowski sum* of π and t, and, for a set A of objects, let $A \oplus t = \{\pi \oplus t \mid \pi \in A\}$. We want to compute

$$D(\mathcal{P},\mathcal{Q}) = \min_{t \in \mathbb{R}^2} H(\mathcal{P}, \mathcal{Q} \oplus t) = \min_{t \in \mathbb{R}^2} H(\mathcal{P} \oplus t, \mathcal{Q}).$$

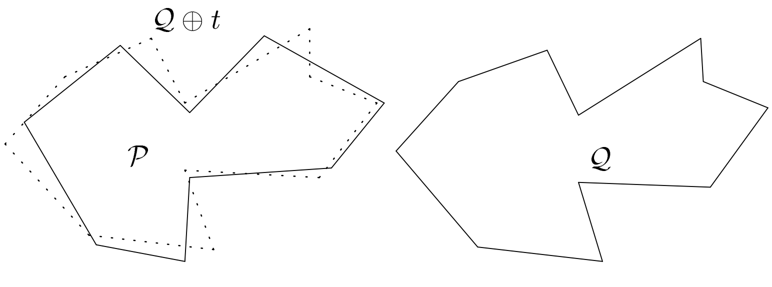

Figure 5: Minimum Hausdorff distance between two polygons

See Figure 5 for an illustration of the problem for the case where $\mathcal{P}$ and $\mathcal{Q}$ are simple polygons.

The value of $D(\mathcal{P},\mathcal{Q})$ gives a measure of the resemblance between $\mathcal{P}$ and $\mathcal{Q}$, so its (efficient) computation has applications in pattern recognition, computer vision, etc. Huttenlocher and Kedem [28] showed that if $\mathcal{P}$ and $\mathcal{Q}$ are sets of m and n points, respectively, then $D(\mathcal{P},\mathcal{Q})$ can be computed in $O((mn)^2\alpha(mn))$ time, where $\alpha(\cdot)$ is the inverse Ackermann function. This bound has been recently improved to $O(mn(m+n)\log mn)$ by Huttenlocher et al. [29]. They also showed that if the distance between two points is measured in the L_1 or L_∞ metrics, the distance $D(\mathcal{P},\mathcal{Q})$, for sets $\mathcal{P}$, $\mathcal{Q}$ each consisting of non-intersecting segments, can be computed in time $O((mn)^2\log mn)$, where $m=|\mathcal{P}|$, $n=|\mathcal{Q}|$. However, their algorithm does not extend to the more useful case of the Euclidean metric. For this case, Alt et al. [7] presented a brute force algorithm with the rather high time complexity $O((mn)^3(m+n)\log(m+n))$.

In this section we show that if $\mathcal{P}$ and $\mathcal{Q}$ are sets each consisting of non-intersecting segments, then $D(\mathcal{P},\mathcal{Q})$, for the Euclidean metric, can be computed in time $O((mn)^2\log^3(mn))$. We first solve the fixed-size problem, which, given a parameter $\delta>0$, determines whether $D(\mathcal{P},\mathcal{Q})\le\delta$. We then convert this procedure, using the parametric search technique, into another algorithm that computes the value of $D(\mathcal{P},\mathcal{Q})$.

We are thus given two sets $\mathcal{P}$, $\mathcal{Q}$, each consisting of non-intersecting segments, and a parameter $\delta>0$, and we wish to determine whether $D(\mathcal{P},\mathcal{Q})\le\delta$. Without loss of generality, we can assume that $\mathcal{P}$ is fixed and we seek a translation of $\mathcal{Q}$ which brings it within distance δ of $\mathcal{P}$. A placement of $\mathcal{Q}$ can be defined by the position of some fixed reference point $O_\mathcal{Q}$ rigidly attached to $\mathcal{Q}$. We assume that the original set $\mathcal{Q}$ is placed so that $O_\mathcal{Q}$ lies at the origin.

$\mathcal{P}_\delta$

e

e_δ

Figure 6: e_δ and $\mathcal{P}_\delta$

Let B_δ denote a disk of radius δ around the origin. For a segment e, let $e_\delta=e\oplus B_\delta=\bigcup\{p+B_\delta \mid p\in e\}$ be the Minkowski sum of e and B_δ. The expanded segment e_δ has the shape of a *racetrack* — a rectangle of width 2δ with two semicircles of radius δ attached to its sides. Let $\mathcal{P}_\delta = \bigcup_{e \in \mathcal{P}} e_\delta$ (see Figure 6). Since the relative interiors of the segments in $\mathcal{P}$ do not intersect each other, the boundaries of e_δ and e'_δ, for $e, e' \in \mathcal{P}$, intersect in at most two points (assuming general position of e, e' [30]; in any case, it is easy to show that the intersection of the boundaries of e_δ, e'_δ consists of at most two connected components). Therefore, by the result of [30], $\mathcal{P}_\delta$ has only $O(m)$ edges (and, symmetrically, $\mathcal{Q}_\delta$ has $O(n)$ edges); here each edge of $\mathcal{P}_\delta$ or of $\mathcal{Q}_\delta$ is either a straight segment or a circular arc.

For a set $A \subseteq \mathbb{R}^2$, let A^c denote the complement $\mathbb{R}^2 - A$. Let $K_{\mathcal{QP}} = \mathcal{P}^c_\delta \ominus \mathcal{Q}$ be the Minkowski difference of $\mathcal{P}^c_\delta$ and $\mathcal{Q}$.

Lemma 4.1 *$K^c_{\mathcal{QP}}$ is the set of translations t of $\mathcal{Q}$ for which $h(\mathcal{Q} \oplus t, \mathcal{P}) \leq \delta$.*

Proof: Let t be a placement of $\mathcal{Q}$ for which $h(\mathcal{Q} \oplus t, \mathcal{P}) \leq \delta$. This is equivalent to asserting that, for every point $\zeta \in \mathcal{Q}$, there is a point $\xi \in \mathcal{P}$, such that $d(\zeta + t, \xi) \leq \delta$. In other words, $h(\mathcal{Q} \oplus t, \mathcal{P}) \leq \delta$ if and only if $(\mathcal{Q} \oplus t) \cap \mathcal{P}^c_\delta = \emptyset$. That is, there is no point $q \in \mathcal{Q}$ and a point p in $\mathcal{P}^c_\delta$ with $q + t = p$ or $t = p - q$. Hence, $h(\mathcal{Q} \oplus t, \mathcal{P}) \leq \delta$ if and only if $t \in K^c_{\mathcal{QP}}$. □

Each edge of $K^c_{\mathcal{QP}}$ is contained in an arc of the form $z - q$, where z is an edge of $\mathcal{P}_\delta$ and q is an endpoint of a segment of $\mathcal{Q}$, or z is a vertex of $\mathcal{P}_\delta$ and q is a segment of $\mathcal{Q}$, or z is a point on a circular arc of $\mathcal{P}_\delta$ whose tangent is parallel to a segment q of $\mathcal{Q}$. Since $K^c_{\mathcal{QP}}$ is defined by $O(mn)$ segments and circular arcs, its combinatorial complexity is $O((mn)^2)$.

In order to define the set of translations t for which $h(\mathcal{P}, \mathcal{Q} \oplus t) \leq \delta$, we flip the roles $\mathcal{P}$ and $\mathcal{Q}$, i.e., we fix $\mathcal{Q}$ and define the set of placements t of $\mathcal{P}$ for which $h(\mathcal{P} \oplus t, \mathcal{Q}) \leq \delta$. By the preceding lemma, this set is $K^c_{\mathcal{PQ}}$, where $K_{\mathcal{PQ}} = \mathcal{Q}^c_\delta \ominus \mathcal{P}$. It now follows that

Lemma 4.2 *$D(\mathcal{P}, \mathcal{Q}) \leq \delta$ if and only if $K^c_{\mathcal{QP}} \cap (-K^c_{\mathcal{PQ}})$ is not empty, where $-K^c_{\mathcal{PQ}} = \{-x \mid x \in K^c_{\mathcal{PQ}}\}$.*

In view of the above discussion, an algorithm for determining whether $D(\mathcal{P}, \mathcal{Q}) \leq \delta$ can be summarized as follows:

1. Compute $\mathcal{P}_\delta$ and $\mathcal{Q}_\delta$.
2. Compute $K_{\mathcal{QP}} = \mathcal{P}^c_\delta \ominus \mathcal{Q}$ and $K_{\mathcal{PQ}} = \mathcal{Q}^c_\delta \ominus \mathcal{P}$.
3. Determine whether $K^c_{\mathcal{QP}}$ and $-(K^c_{\mathcal{PQ}})$ have nonempty intersection.

As for the time complexity of (a sequential version of) the algorithm, Step 1 can be accomplished in time $O((m+n)\log^2(m+n))$ using the algorithm of [30]. Steps 2 and 3 can be performed together by constructing the entire arrangement of arcs that define the edges of $K^c_{\mathcal{QP}}$ and $-K^c_{\mathcal{PQ}}$ and then, for each face in the resulting arrangement, determining whether it lies in both $K^c_{\mathcal{QP}}$ and $-K^c_{\mathcal{PQ}}$. Since the edges of $K_{\mathcal{QP}}$ and $-K^c_{\mathcal{PQ}}$ are defined by $O(mn)$ segments and circular arcs, a sweep line algorithm can perform the above steps in $O((mn)^2 \log mn)$ time. Hence, we can conclude

Theorem 4.3 *Given a collection $\mathcal{P}$ of m non-intersecting segments and another collection $\mathcal{Q}$ of n non-intersecting segments in the plane, one can determine whether $D(\mathcal{P}, \mathcal{Q}) \leq \delta$ in time $O((mn)^2 \log(mn))$.*

Next, in order to apply parametric searching, we need an efficient parallel version of the algorithm. It is well known that the arrangement of a collection of t 'well-behaving' arcs in the plane can be computed in $O(\log t)$ time using $O(t^2)$ processors, see e.g. [1]. Therefore the arrangement of $\{e \oplus B_\delta \mid e \in \mathcal{P}\}$ can be computed in $O(\log m)$ time with $O(m^2)$ processors. After having computed the arrangement, we determine for each face f of the arrangement the number of racetracks e_δ that contain f; we denote this quantity by c_f. For two adjacent faces f, f', we have $|c_f - c_{f'}| = 1$. We first compute a spanning tree of the dual graph of the arrangement, and then convert it to an Eulerian path Π. Using an algorithm of Tarjan and Vishkin [39], Π can be computed in $O(\log m)$ time with $O(m^2)$ processors (see also [1]). Once we have computed Π, we can compute c_f for each face of the arrangement by a parallel prefix computation algorithm, see e.g. [1]. It is easily seen that an edge γ of the arrangement is in the boundary of $\mathcal{P}_\delta$ if and only if $c_f = 0$ for one of the faces adjacent to γ. Testing each edge of the arrangement in parallel, we can compute the edges of P_δ in constant time. Thus $\mathcal{P}_\delta$ can be computed in $O(\log m)$ time with $O(m^2)$ processors. Similarly, one can compute $\mathcal{Q}^c_\delta$ in $O(\log n)$ time using $O(n^2)$ processors. Finally, an intersection between $K^c_{\mathcal{QP}}$ and $-K^c_{\mathcal{PQ}}$ can be detected in $O(\log mn)$ time with $O(m^2 n^2)$ processors by computing the arrangement of arcs defining the edges of $K_{\mathcal{QP}}$ and $-K^c_{\mathcal{PQ}}$ and by determining whether there is a face in the resulting arrangement that lies in both $K^c_{\mathcal{QP}}$ and $-K^c_{\mathcal{PQ}}$. A variant of the procedure that computes P_δ can be used to perform the above two steps. Hence, the overall running time of the algorithm is $O(\log mn)$ using $O(m^2 n^2)$ processors. Applying the parametric search technique to the resulting algorithm, we can conclude

Theorem 4.4 *Given a collection $\mathcal{P}$ of m non-intersecting segments and another collection $\mathcal{Q}$ of n non-intersecting segments in the plane, one can compute the minimum Hausdorff distance between $\mathcal{P}$ and $\mathcal{Q}$, under translation, in time $O((mn)^2 \log^3(mn))$.*

5 Complete Mutual Visibility Among Spheres

Let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a given set of n spheres in $\mathbb{R}^3$, all with the same radius. Let c_i denote the center of S_i, for $i = 1, \ldots, n$. Two spheres S_i and S_j are said to be *mutually* *visible* if the line segment $c_i c_j$ does not intersect (the interior of) any other sphere, and $\mathcal{S}$ is *completely mutually visible* if every pair of spheres in $\mathcal{S}$ is mutually visible.

The problem studied in this section is: "Given a set P of n points in $\mathbb{R}^3$, we wish to determine the largest possible common radius r such that the set of n spheres of radius r, centered at the given points, is (pairwise disjoint and) completely mutually visible."

This problem arises in the context of parallel computations using optical interconnections[-3-] [3]: GIve a reference?? The spheres model the individual processors, and mutual visibility between a pair of spheres models the ability of the two processors to communicate by an optical link. In our problem the locations of the (centers of the) processors are predetermined, and we want to determine how large can the processors be if every pair is to be able to communicate optically.

By running any closest-pair algorithm, e.g., [10], we can determine in $O(n \log n)$ time the largest radius r_0 so that the interiors of all spheres of radius r_0 centered at the points of P are pairwise disjoint. Therefore, we only have to determine the largest radius $r^* \leq r_0$ such that the spheres of radius r^* centered at the points of P are mutually visible. In order to employ the parametric searching technique, we solve the following fixed-size decision problem: "Given a set $\mathcal{S}$ of n pairwise disjoint spheres of unit radius, determine if it is completely mutually visible".

We use the following simple scheme. Fix a sphere of $\mathcal{S}$, say S_n. We determine in $O(n \log^2 n)$ time whether all other spheres of $\mathcal{S}$ are mutually visible from S_n. Repeating this procedure for all spheres of $\mathcal{S}$ yields an $O(n^2 \log^2 n)$ time algorithm to determine whether $\mathcal{S}$ is completely mutually visible.

Let c_i denote the center of S_i, for $i = 1, \ldots, n$. In order to determine whether there is any sphere S_i for which the segment $c_i c_n$ intersects the interior of any other sphere, we sort the spheres of $\mathcal{S} - \{S_n\}$ in the nondecreasing order of the distances of their centers from c_n. Let $S_1, \ldots, S_{n-1}$ be the resulting sequence. Since the spheres in $\mathcal{S}$ are pairwise disjoint and congruent, it can be shown that if $i > j$ then S_i cannot hide S_j when viewed from c_n (i.e., for any point $p \in S_j$, the segment pc_n cannot intersect the interior of S_i). We define the projection of a point $p \in \mathbb{R}^3$ on S_n, denoted p^*, to be the intersection point of the the sphere S_n and the ray emanating from c_n in direction $\vec{c_n p}$. Let S_i^* denote the projection of S_i on S_n, and let $\mathcal{S}^* = \{ S_i^* \mid i < n \}$. $\mathcal{S}^*$ is a collection of spherical caps; the cap S_i^* has c_i^* as its center, for $i = 1, \ldots, n$.

Lemma 5.1 *S_n and S_j are mutually visible if and only if c_j^* does not lie in the union of $S_1^*, \ldots, S_{j-1}^*$.*

Proof: Consider the segment $c_n c_j$. By the above discussion, if $i > j$ then S_i cannot intersect $c_n c_j$. Therefore, S_j and S_n are mutually visible if and only if none of $S_1, \ldots, S_{j-1}$ intersect the segment $c_n c_j$, which immediately implies the lemma. □

In view of the lemma, it suffices to determine for each S_j whether c_j^* lies in $\bigcup_{k<j} S_k^*$. We

construct a minimum height binary tree $\mathcal{B}$ on $\mathcal{S}^*$ whose i^{th}-leftmost leaf stores S_i^*. For each node $\delta \in \mathcal{B}$, let U_δ denote the union of the caps stored at the leaves of the subtree rooted at δ. If δ is a left child of its parent, then we also associate with δ the subset C_δ of centers of the caps stored at the leaves of the subtree rooted at the right sibling of δ. If δ is the root of $\mathcal{B}$ or the right child of its parent, we set $C_\delta = \emptyset$. Since the boundaries of every pair of caps in $\mathcal{S}^*$ intersects in at most two points, U_δ has linear complexity [30]. Moreover, U_δ can be computed in a bottom-up fashion, because $U_\delta = U_{\mathrm{lson}(\delta)} \cup U_{\mathrm{rson}(\delta)}$. It follows from the construction and Lemma 5.1 that S_n is not mutually visible to all the other spheres if and only if there is a node $\delta \in \mathcal{B}$ such that one of the points in C_δ lies in U_δ.

Using a variant of the algorithm of Kedem et al. [30], we can compute U_δ from $U_{\mathrm{lson}(\delta)}$ and $U_{\mathrm{rson}(\delta)}$ in $O(|U_\delta| \log |U_\delta|)$ time.[3] Moreover, the algorithm can easily be modified so that in $O(|C_\delta| \log |U_\delta|)$ additional time one can determine whether any point of C_δ lies in U_δ. Since $\sum_\delta (|U_\delta| + |C_\delta|) = O(n \log n)$, the total time spent is $O(n \log^2 n)$, as claimed earlier.

In order to apply the parametric search technique, we need a parallel version of the above procedure. Since the algorithm in [30] is based on a sweep-line paradigm, it is not so easy to parallelize. We will describe a different algorithm for computing the union of two planar or spherical regions, which is easy to parallelize. For the sake of simplicity we will describe the algorithm assuming that U_δ is a planar map.

Let R, B denote the set of edges in $U_{\mathrm{lson}(\delta)}$ and $U_{\mathrm{rson}(\delta)}$, respectively; let $m = |R| + |B|$. By splitting each edge into two subedges, if required, we can assume that the edges in $R \cup B$ are x-monotone. We also assume that for every edge γ of $U_{\mathrm{lson}(\delta)}$ (resp. $U_{\mathrm{rson}(\delta)}$) we know whether the top or the bottom side of γ lies in $U_{\mathrm{lson}(\delta)}$ (resp. $U_{\mathrm{rson}(\delta)}$). Notice that each intersection point of an edge of R and an edge of B is a vertex of U_δ, the other vertices being those vertices of $U_{\mathrm{lson}(\delta)}$ (resp. $U_{\mathrm{rson}(\delta)}$) lying outside $U_{\mathrm{rson}(\delta)}$ (resp. $U_{\mathrm{lson}(\delta)}$).

To compute the intersection points of R and B, we construct a segment tree $\mathcal{T}$ on the edges of $R \cup B$; see [37] for details. Each node v of $\mathcal{T}$ is associated with a horizontal interval Δ_v, a subset R_v of edges in R, and a subset B_v of edges in B. The x-projection of any edge in $R_v \cup B_v$ covers the interval Δ_v, and an edge of $R \cup B$ is stored in at most $O(\log m)$ nodes along two paths of $\mathcal{T}$. We assume that the edges of $R_v \cup B_v$ are clipped to within the vertical strip $\Delta_v \times [-\infty, +\infty]$. If $\gamma \in R_v$ then, for each ancestor w of v (including v itself), we determine the edges of B_w that intersect γ. Since the arcs of B_w fully cross the strip from left to right and are nonintersecting, they form a list which is totally ordered in the y-direction. Since γ intersects any arc of B_w in at most two points, we can determine all edges that intersect γ in $O(\log m)$ time by doing a multiple binary search, in which we locate the endpoints of γ and the highest and lowest arcs of B_w that intersect γ. After the searches, we simply go over all the arcs that were found, which form a contiguous sublist of B_w, and

[3] Although the original algorithm of [30] computes the union of two planar maps, it can be easily extended to spherical maps. Actually, one can convert U_δ, for any $\delta \in \mathcal{B}$, into a planar map by taking a stereographic projection of caps in S^* on a horizontal plane.

report all the corresponding intersection points. Since the total number of intersections, over all pairs of arcs, is only $O(m)$, we have enough processors to report all intersections in constant parallel time (in Valiant's model). We apply a symmetric procedure to the arcs γ in B_v. Repeating the same procedure at all nodes $v \in \mathcal{T}$, we can determine all intersection points of R and B. Note that the searches also find all vertices of $U_{\mathrm{lson}(\delta)}$ that lie outside $U_{\mathrm{rson}(\delta)}$, and vice versa. Recall that there are only $O(m)$ intersection points. The above procedure can be implemented in $O(\log m)$ time using $O(m \log m)$ processors; see [15, 37] for details.

Once we have computed the intersection points, we can complete the construction of the boundary of U_δ, denoted as ∂U_δ, which involves the determination of the edges of U_δ and the connected components of its boundary, using a sequential algorithm, because it does not involve any comparisons. We first split the edges of $U_{\mathrm{lson}(\delta)}$ and $U_{\mathrm{rson}(\delta)}$ at the intersection points of R and B. It decomposes each connected component of $\partial U_{\mathrm{lson}(\delta)}$ and $\partial U_{\mathrm{rson}(\delta)}$ into maximal chains (the chain is the whole component if it does not contain any intersection point), so that no chain contains an intersection point in its (relative) interior. It is easily seen that either the entire chain appears on ∂U_δ, or it does not appear at all. If a chain is an entire connected component of $U_{\mathrm{lson}(\delta)}$ (resp. $U_{\mathrm{rson}(\delta)}$), then either all of its vertices lie on ∂U_δ, or none of them lie on ∂U_δ, so by picking one vertex of the component and locating it in $U_{\mathrm{rson}(\delta)}$ (resp. $U_{\mathrm{lson}(\delta)}$), we can determine whether it appears on ∂U_δ. On the other hand, if a chain is not the entire component, i.e., its endpoints are intersection points, we can determine whether it appears on ∂U_δ by a local test at one of its endpoints. We discard all chains that do not appear on the boundary of U_δ. We now complete the construction of U_δ by gluing the remaining chains together, i.e., we connect a chain of $U_{\mathrm{lson}(\delta)}$ with a chain of $U_{\mathrm{rson}(\delta)}$ if they share a common endpoint. Since the endpoints of chains are intersection points of R and B, it is easily seen that the chains glue together properly. The total time spent in these two steps is $O(n \log n)$, because apart from local tests it only requires $O(n)$ point location queries in planar subdivisions.

As for locating the points of C_δ in U_δ, we associate a point p of C_δ with a node v of $\mathcal{T}$ if the x-projection of p lies in Δ_v; p is stored at $O(\log m)$ nodes along a path in $\mathcal{T}$. Let C_v be the set of points associated with v. Let γ_v (resp. γ'_v) be the edge of R_v (resp. B_v) lying immediately above p. If there is a node v such that $p \in C_v$, and either the bottom side of γ_v is in $U_{\mathrm{lson}(\delta)}$ or the bottom side of γ'_v is in $U_{\mathrm{rson}(\delta)}$, then $p \in U_\delta$. By doing binary searches in parallel at all $O(\log m)$ nodes where p is stored, we can determine in $O(\log m)$ time whether p lies in U_δ. Since $\sum_v |C_v| = O(|C_\delta| \log m)$, the total number of processors required over all points of C_δ is $O(|C_\delta| \log m)$.

Finally, to determine whether S_n is mutually visible from all other spheres, we run the above procedure on the binary tree $\mathcal{B}$ in a bottom-up fashion, running at all nodes of the same level in parallel. Since a cap of $\mathcal{S}^*$ or a point of C^* is being stored at only one node of $\mathcal{B}$ for any given level, the total number of processors required to run the above procedure is $O(n \log n)$. The total time spent by the algorithm under Valiant's model is

$O(\log^2 n)$. It thus yields an overall parallel algorithm for determining whether $\mathcal{S}$ is mutually visible, which requires $O(\log^2 n)$ time and $O(n^2 \log n)$ processors. Plugging all this in the parametric search paradigm, we can conclude

Theorem 5.2 *Given a set of n points in $\mathbb{R}^3$, we can determine, in time $O(n^2 \log^5 n)$, the largest possible common radius r, so that the spheres with radius r centered at the given points are (pairwise disjoint and) completely mutually visible.*

6 Conclusion

In this paper we applied the parametric searching technique to a number of problems — the biggest stick problem, the minimum width annulus problem, the problem of computing the minimum Hausdorff distance under translation in the Euclidean metric between two polygonal regions, and the problem of finding largest mutually visible spheres. For each of these problems we either obtained the first nontrivial solution, or developed a significantly faster algorithm than the previously best known one. We nevertheless feel that most of our algorithms are not close to optimal and better bounds can be achieved.

We conclude by mentioning some open problems:

1. In Section 2, we described an algorithm for partitioning the lower envelope of n algebraic surfaces of fixed degree in $\mathbb{R}^4$ into a family of $O(n^4\beta(r))$ constant-size cells. We are not aware of any matching lower bound. A better solution for this problem will yield an improved algorithm for the biggest stick and the minimum width annulus problems. We conjecture an upper bound that is nearly cubic in n.

2. Is there a subquadratic algorithm for computing a longest segment that can be placed inside a polygonal region with holes?

3. Is it possible to determine whether the minimum Hausdorff distance between two sets of segments, as in Section 4, is at most δ, without computing $K^c_{\mathcal{PQ}} \cap (-K^c_{\mathcal{QP}})$ explicitly? In particular, can such a computation be carried out in $o((mn)^2)$ time?

4. In general, there is the challenge of applying the parametric searching technique to other problems in computational geometry and geometric optimization. Recently, Chazelle et al. [16] have obtained improved solutions for some other geometric problems, including the computation of the diameter and width of point sets in $\mathbb{R}^3$, using the parametric searching technique.

Acknowledgments

The authors wish to thank Jirka Matoušek for a useful discussion which helped in improving the running time of some of the algorithms. Thanks are also due to referees for their helpful comments.

References

[1] P.K. Agarwal, B. Aronov, M. Sharir and S. Suri, Selecting distances in the plane, *Proc. 6th ACM Symp. on Computational Geometry,* 1990, 321–331. (Also to appear in *Algorithmica.*)

[2] P.K. Agarwal, A. Efrat, M. Sharir and S. Toledo, Computing a segment-center for a planar point set, Tech. Rept. CS-1991-33, Dept. Computer Science, Duke University, 1991.

[3] P.K. Agarwal and J. Matoušek, Ray shooting and parametric search, *Proc. 24th ACM Symp. on Theory of Computing*, 1992, pp. 517–526.

[4] P.K. Agarwal and M. Sharir, Planar geometric location problems, Tech. Rept. 90-58, DIMACS, Rutgers University, August 1990. (Also to appear in *Algorithmica.*)

[5] P.K. Agarwal and M. Sharir, Counting circular arc intersections, *Proc. 7th ACM Symp. on Computational Geometry,* 1991, pp. 10–20.

[6] A. Aggarwal and S. Suri, The biggest diagonal in a simple polygon, *Inf. Proc. Letters* 35 (1990), 13–18.

[7] H. Alt, B. Behrends and J. Blömer, Approximate matching of polygonal shapes, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 186–193.

[8] M. Atallah, R. Cole and M. Goodrich, Cascading divide-and-conquer: A technique for designing parallel algorithms, *SIAM J. Computing* 18 (1989), 499–532.

[9] J.L. Bentley and T. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Trans. on Computers* C–28 (1979), 643–647.

[10] J.L. Bentley and M. Shamos, Divide-and-conquer in multidimensional space, *Proc. 8th ACM Symp. on Theory of Computing*, 1976, pp. 220–230.

[11] B. Chazelle, A polygon cutting theorem, *Proceedings 23rd Annual IEEE Symposium on Foundations of Computer Science*, 1982, pp. 339–349.

[12] B. Chazelle, The polygon containment problem, in *Advances in Computing Research, Vol. I: Computational Geometry,* (F.P. Preparata, Ed.), JAI Press, Greenwich, Connecticut (1983), pp. 1–33.

[13] B. Chazelle, Triangulating a simple polygon in linear time, *Discrete and Computational Geometry* 6 (1991), 485–524.

[14] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, A singly exponential stratification scheme for real semi–algebraic varieties and its applications, *Theoretical Computer Science* 84 (1991), 77–105. (Also in *Proc. 16th Int. Colloq. Automata, Lang. Prog.*, 1989, pp. 179–192.)

[15] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, to appear in *Algorithmica.*

[16] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, Diameter, width, closest line-pair, and parametric searching, *Proc. 8th ACM Symp. on Computational Geometry*, 1992, pp. 120–129.

[17] B. Chazelle and L. Guibas, Visibility and intersection problems in plane geometry, *Discrete Comput. Geom.* 4 (1989), 551–589.

[18] B. Chazelle and M. Sharir, An algorithm for generalized point location and its applications, *J. Symbolic Computation* 10 (1990), pp. 281–309.

[19] K. Clarkson and P. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.* 4 (1989), 387–422.

[20] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, *J. ACM* 31 (1984), 200–208.

[21] R. Cole, J. Salowe, W. Steiger and E. Szemerédi, Optimal slope selection, *SIAM J. Computing* 18 (1989), 792–810.

[22] G. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, *Lecture Notes in Computer Science* 33 (1975), Springer-Verlag, Berlin, pp. 134–183.

[23] H. Ebara, N. Fukuyama, H. Nakano and Y. Nakanishi, Roundness algorithms using the Voronoi diagrams, *First Canadian Conf. Computational Geometry*, 1989.

[24] G. Frederickson, Optimal algorithms for tree partitioning, *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, 1991, pp. 168–177.

[25] G. Frederickson and D. Johnson, Finding the kth shortest paths and p-centers by generating and searching good data structures, *J. Algorithms* 4 (1983), 61–80.

[26] G. Frederickson and D. Johnson, Generalized selection and ranking: sorted matrices, *SIAM J. Computing* 13 (1984), 14–30.

[27] D. Haussler and E. Welzl, ϵ-nets and simplex range queries, *Discrete Comput. Geom.* 2 (1987), 127–151.

[28] D. Huttenlocher and K. Kedem, Efficiently computing the Hausdorff distance for point sets under translation, *Proc. 6th ACM Symp. on Computational Geometry*, 1990, pp. 340–349.

[29] D. Huttenlocher, K. Kedem and M. Sharir, The upper envelope of Voronoi surfaces and its applications, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 194–203.

[30] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete Comput. Geom.* 1 (1986), 59–71.

[31] J. Matoušek, Randomized optimal algorithm for slope selection, *Inf. Proc. Letters* 39 (1991), 183–187.

[32] J. Matoušek, Approximations and optimal geometric divide-and-conquer, *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, pp. 506–511.

[33] J. Matoušek. Efficient partition trees, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 1–9.

[34] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM* 30 (1983), 852–865.

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

[36] N. Naor and M. Sharir, Computing the center of a point set in three dimensions, *Proc. 2nd Canadian Conf. on Computational Geometry* (1990), pp. 10–13.

[37] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer–Verlag, New York, 1985.

[38] M. Sharir and S. Toledo, Extremal polygon containment problems, Tech. Rept. 228/91, Dept. Computer Science, Tel Aviv University, 1991. (See also S. Toledo, Extremal polygon containment problems, *Proc. 7th ACM Symp. on Computational Geometry,* 1991, pp. 176–185.)

[39] R. Tarjan and U. Vishkin, An efficient parallel biconnectivity algorithm, *SIAM J. Comp.* 14 (1985), 862-874.

[40] L. Valiant, Parallelism in comparison problems, *SIAM J. Computing* 4 (1975), 348–355.