Discrete Comput Geom 10:183–196 (1993)

Discrete & Computational
Geometry

Diameter, Width, Closest Line Pair, and Parametric Searching*

Bernard Chazelle,[1] Herbert Edelsbrunner,[2] Leonidas Guibas,[3] and Micha Sharir[4]

[1]Department of Computer Science, Princeton University, Princeton, NJ 08544, USA

[2]Computer Science Department, University of Illinois, Urbana, IL 61801, USA

[3]Computer Science Department, Stanford University, Stanford, CA 94305, USA, and DEC Systems Research Center, Palo Alto, CA 94301, USA

[4]Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA, and School of Mathematics, Tel Aviv University, Ramet Aviv 69 978, Israel

Abstract. We apply Megiddo's parametric searching technique to several geometric optimization problems and derive significantly improved solutions for them. We obtain, for any fixed $\varepsilon > 0$, an $O(n^{1+\varepsilon})$ algorithm for computing the diameter of a point set in 3-space, an $O(^{8/5+\varepsilon})$ algorithm for computing the width of such a set, and an $O(n^{8/5+\varepsilon})$ algorithm for computing the closest pair in a set of n lines in space. All these algorithms are deterministic.

1. Introduction

In 1983 Megiddo proposed an optimization technique, known as *parametric searching*, which has since proven very powerful and versatile. We apply his

*Work by Bernard Chazelle was supported by NSF Grant CCR-90-02352. Work by Herbert Edelsbrunner was supported by NSF Grant CCR-89-21421. Work by Leonidas Guibas and Micha Sharir was supported by a grant from the U.S.–Israeli Binational Science Foundation. Work by Micha Sharir was also supported by ONR Grant N00014-90-J-1284, by NSF Grant CCR-89-01484, and by grants from the Fund for Basic Research administered by the Israeli Academy of Sciences, and the G.I.F., the German–Israeli Foundation for Scientific Research and Development.

technique to solve basic problems in computational geometry. Here is an overview of the problems considered in this paper.

Diameter of a Point Set in 3-Space. Compute the *diameter* of a set $\mathcal{P}$ of n points in 3-space, defined as the maximum distance between a pair of points in $\mathcal{P}$. This problem was solved by Clarkson and Shor [13] by a randomized algorithm with optimal expected running time $O(n \log n)$. However, the previously best-known deterministic algorithm for this problem runs in time $O(n^{4/3+\varepsilon})$ [2]. In this paper we give a deterministic solution that runs in time $O(n^{1+\varepsilon})$.[1]

Width of a Point Set in 3-Space. Given a point-set $\mathcal{P}$ as above, compute the *width* of $\mathcal{P}$, defined as the smallest distance between a pair of parallel planes that bound a slab containing $\mathcal{P}$. As far as we know, the best previous solution to this problem takes $O(n^2)$ time [18]. We present a (deterministic) algorithm that runs in time $O(n^{8/5+\varepsilon})$.

Closest Line Pair in 3-Space. Given a set $\mathcal{L}$ of n lines in 3-space, compute the closest pair in $\mathcal{L}$, with the distance between two lines l_1, l_2 defined as the minimum Euclidean distance $d(p, q)$ between a point $p \in l_1$ and a point $q \in l_2$. We present a (deterministic) algorithm that runs in time $O(n^{8/5+\varepsilon})$.

Although the application of Megiddo's parametric searching to the first three problems is rather natural, we are not aware of any previous attempt to apply the technique to these problems. The number of efficient geometric algorithms that make use of this technique is still rather small (see [1], [3]–[6], [15], and [23]–[28] for results of this kind); we believe that the potential of the technique in computational geometry is still quite far from being exhausted, as this paper well demonstrates.

The parametric-searching technique has been reviewed in many of the papers just cited. For the sake of completeness, we review it briefly here as well.

Suppose we have a problem $\mathcal{P}(d)$ that receives as input n data items and a real parameter d. We want to find a value d^* of d at which the output of $\mathcal{P}(d)$ satisfies certain properties. Suppose we have an efficient (serial) algorithm A_s for solving $\mathcal{P}(d)$ at any given d, and that, as a by-product, the algorithm A_s can also determine whether the given d is equal to, smaller than, or larger than the desired value d^*. Assume moreover that the flow of execution of A_s depends on comparisons, each of which is testing the sign of a low-degree polynomial in d and in the input items.

Megiddo's technique then runs the algorithm A_s "generically," without specifying the value of d, with the intention of simulating its execution at the unknown d^*. Each time a comparison is made, the few roots $d_1, d_2, \ldots$ of the associated

[1] The precise meaning of such a bound (similar bounds are given below) is that, given any fixed $\varepsilon > 0$, we can fine-tune the algorithm so that its running time is $O(n^{1+\varepsilon})$, with the constant of proportionality depending on ε.

polynomial are computed, and we run A_s at each of them, thereby determining the location of d^* among these roots, and thus also the sign of the polynomial at d^*, namely the outcome of the comparison at d^*. (Of course, if any of these roots turns out to be d^*, the algorithm stops right there.) Then execution of the generic A_s can be resumed. As we proceed through this execution, each comparison that we resolve further constrains the range where d^* can lie, and we thus obtain a sequence of progressively smaller intervals, each known to contain d^*, until we either hit d^* as one of the roots being tested, or A_s terminates with a final interval I. In our applications, though, the second possibility cannot arise. (Indeed, it follows that the outcome of A_s will be combinatorially the same when we run it on any $d \in I$. Since we seek the smallest d that meets the problem constraints, it follows that d^* must be left endpoint of I, so it would have been detected earlier as one of the tested roots.)

The cost of this implicit parametric searching is usually dominated by $C_s T_s$, where C_s is the maximum number of comparisons executed by A_s, and T_s is the maximum running time of A_s. Since this bound is generally too high, Megiddo suggests replacing the generic A_s by a parallel algorithm, A_p. If A_p uses P processors and runs in T_p parallel steps, then each such step involves at most P *independent* comparisons, that is, each can be carried out without having to know the outcome of the others. We can then compute the roots of all the P polynomials associated with these comparisons, and run a binary search to find the location of d^* among them (using, say, the serial algorithm A_s at each binary search step). This requires $O(P + T_s \log P)$ time per parallel step, for a total of $O(PT_p + T_s T_p \log P)$ time, which often results in a saving of nearly an order of magnitude in the running time. A further improvement by Cole [14] can in certain cases reduce the running time by another logarithmic factor. We remark that, since the parallel algorithm is simulated sequentially, we do not have to worry about processor allocation, interprocessor communication, and other issues that arise in real parallel algorithms. We therefore assume the weak parallel model of Valiant [29], where the complexity of the algorithm is measured only by the number of comparisons that it makes.

We show in subsequent sections how this paradigm can be applied to the problems under consideration. The diameter problem is studied in Section 2, the width problem in Section 3, and the closest-line-pair problem in Section 4.

2. Diameter in 3-Space

Let $\mathcal{P}$ be a given set of n points in 3-space. The *diameter* of $\mathcal{P}$ is defined as

$$diam(\mathcal{P}) = \max\{d(p, q): p, q \in \mathcal{P}\},$$

where $d(p, q)$ is the Euclidean distance between p and q.

A quadratic algorithm for computing $diam(\mathcal{P})$ is trivial. Faster algorithms have been obtained in [2]. Using randomization, Clarkson and Shor [13] have obtained

an algorithm whose expected running time (over the internal randomizations that it performs) is $O(n \log n)$. We obtain a deterministic algorithm that comes close—its running time is $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$.

Since we want to apply parametric searching, we first need to solve the "fixed-size" problem. That is, given d, determine whether $diam(\mathscr{P})$ is greater than, equal to, or less than d. This problem can be restated as follows. Let $B(p, d)$ denote a ball of radius d around the point p. Note that $diam(P) \leq d$ if any only if, for each $p \in \mathscr{P}$, the ball $B(p, d)$ contains $\mathscr{P}$. Hence $diam(\mathscr{P}) \leq d$ if and only if

$$\mathscr{P} \subseteq \bigcap_{p \in \mathscr{P}} B(p, d).$$

Ideally, we would like to calculate the intersection $K = \bigcap_{p \in \mathscr{P}} B(p, d)$, preprocess it for fast point location, and then test each $p \in \mathscr{P}$ for containment in K. Aiming toward this goal, we first note that, as shown in [13], the combinatorial complexity of K is linear in n. Unfortunately, we are not aware of any deterministic algorithm that constructs K in $O(n \log n)$ time. The only algorithm that achieves such a performance is the randomized algorithm given in [13]. We did not manage to derandomize this algorithm directly, and instead we give a somewhat different solution, based on sampling the input points (in a manner explained below). The disadvantage of the revised approach is that it is slightly less efficient—it runs in time $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$. The advantages are that this approach is deterministic and that it can be easily parallelized (in Valiant's model). This gives us all the ingredients needed to apply parametric searching.

Before we go any further, let us briefly recall some basic facts about the theory of range spaces—see [17] for details. A range space $(X, \mathscr{R})$ of VC-dimension d is a set system with the property that the largest subset Y of X to be shattered by $\mathscr{R}$ is of size d: we say that Y is shattered by $\mathscr{R}$ if each of the $2^{|Y|}$ subsets of Y can be expressed as $Y \cap R$ for some $R \in \mathscr{R}$. Assuming that X is finite, a subset N of X is called an *ε-net* for $(X, \mathscr{R})$ if every set of $\mathscr{R}$ of size greater than $\varepsilon |X|$ intersects N. The existence of ε-nets of size $O(\varepsilon^{-1} \log \varepsilon^{-1})$ is established in [17].

Returning to our problem, here is a detailed description of our solution. Let us first define a *spherical tetrahedron* as the convex hull $conv(\Delta \cup \{q\})$, where Δ is a *spherical triangle* on some sphere σ (a portion of σ bounded by three arcs of great circles) and q is a point inside the ball bounded by σ. Consider the range space $(X, \mathscr{R})$, where X is the set of balls $B(p, d)$, for $p \in \mathscr{P}$, and $\mathscr{R}$ is the family of subsets of X obtained by intersecting the interior of every possible spherical tetrahedron with the boundaries of the balls of X. This range space is easily seen to have finite VC-dimension. We fix some sufficiently large constant integer parameter r, and apply the recent algorithm of Matoušek [22] for computing a $(1/r)$-net N of size $O(r \log r)$. All this takes $O(n)$ time, with the constant of proportionality depending on r.

Next we construct the intersection E of the balls bounded by the spheres of N in constant time, using any brute-force method. (A simple method is to apply a standard "lifting" transformation, which maps the balls into half-spaces of 4-space, then compute the intersection M of these half-spaces, compute the intersection of

M with the paraboloid $x_4 = x_1^2 + x_2^2 + x_3^2$, and finally project the resulting intersection back into 3-space.) If the intersection E is empty or a singleton, then we stop right away—the diameter of $\mathcal{P}$ (even the diameter of the subset of $\mathcal{P}$ consisting of the centers of the spheres in N) is larger than d. Otherwise, we choose any point q in the interior of the convex set E, triangulate the boundary of E into a collection of spherical triangles by drawing great circular arcs between vertices, as needed, and connect q to each such spherical triangle, to obtain a decomposition of E into a collection of spherical tetrahedra; since the combinatorial complexity of E is $O(r \log r)$, it follows that the number of spherical tethedra in the decomposition is also $O(r \log r)$. Moreover, since N is a $(1/r)$-net, any spherical tetrahedron τ is crossed by at most n/r boundaries of the given balls $B(p, d)$. Any other ball $B(p, d)$ either fully contains τ or is disjoint from it. We also compute, for each point $p \in \mathcal{P}$, the tetrahedron τ containing it. This again can be done by brute force, testing each point against every tetrahedron. If any point of $\mathcal{P}$ falls out of E, again we can stop the algorithm, because the diameter then is clearly larger than d. Similarly, if some points of $\mathcal{P}$ fall inside a spherical tetrahedron τ and at least one ball $B(p, d)$ is disjoint from τ, then we also stop the algorithm because the desired diameter must be larger than d.

We are now left with $O(r \log r)$ subproblems, each corresponding to some spherical tetrahedron τ, and involving $n_\tau \leq n/r$ balls and m_τ points of $\mathcal{P}$; the subproblem at τ is to determine whether all these points lie in the intersection of all these balls. Note that $\sum_\tau m_\tau = n$. Each of these subproblems is solved recursively in the same manner described above. When the size of each subproblem becomes a sufficiently small constant, we solve it by brute force, as described above for the sample N. The algorithm terminates in one of two ways. Either it successfully asserts that at each subproblem all points lie in the intersection of all balls, or else it is stopped prematurely in one of the cases described above. In the former case we conclude that the diameter of $\mathcal{P}$ is smaller than or equal to d, whereas in the latter case the diameter must be larger than d. By fine-tuning the algorithm we can also disambiguate between the cases $diam(\mathcal{P}) < d$ and $diam(\mathcal{P}) = d$ (in the second case some point must lie on the boundary of some ball).

The running time of the algorithm is easy to bound by noticing that the recursion has only $O(\log_r n)$ stages, and that the cost of each stage is linear in the total size of all subproblems at that stage. The overall number of points in the subproblems at stage j always remains at most n, while the total number of balls in these subproblems is bounded by $(c \log r)^j n$ for some absolute constant c. From this it easily follows that the total running time is at most $O(n^{1+\varepsilon})$, where $\varepsilon = O(\log \log r / \log r)$. In other words, for any $\varepsilon > 0$ we can choose $r = r(\varepsilon)$ sufficiently large so that the running time of the algorithm is $O(n^{1+\varepsilon})$, where the constant of proportionality depends on r, and thus on ε.

To apply parametric seaching we still need to design a parallel version of this algorithm (in Valiant's model). Fortunately, this is fairly easy to do. A close inspection of the algorithms of [22] reveals that they are all parallelizable, when r is a constant—the main algorithm for constructing ε-approximation runs in a logarithmic number of stages, and in each stage it examines in parallel many subsets of X of relatively small size. As we move from stage to stage the size of

the sets in question keeps growing and then shrinks back. As described in [22], the sizes grow at a geometric rate, so the maximum size (in the middle stage) can be rather high, but it is not hard to see that the algorithm also works properly with a much smaller growth rate followed by a similarly slow rate of decrease. Since there are only logarithmically many stages, this implies that the algorithms of [22] can be parallelized to run in polylogarithmic time, using a linear number of processors. See [12] for details. The construction and decomposition of the intersection E takes constant time, and the distribution of the balls and points among the subproblems is also easy to do in parallel—simply test each ball and each point against every tetrahedron. All this implies that the entire algorithm can be executed in parallel in a polylogarithmic number of parallel stages, using $O(n^{1+\varepsilon})$ processors (where the dependence on ε is as in the serial algorithm).

Plugging all this into the machinery of parametric searching, we readily obtain our first result:

Theorem 2.1. *Given a set $\mathcal{P}$ of n points in 3-space, the diameter of $\mathcal{P}$ can be computed deterministically in time $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$.*

Remarks. The main reason why the performance of our algorithm is worse than the expected performance $O(n \log n)$ of the randomized algorithm of [13] is that we were unable to design a more efficient deterministic algorithm for constructing the intersection of n congruent balls in 3-space. Such an intersection is constructed in [13] by a randomized incremental algorithm. It is an interesting challenge to find a technique to derandomize their algorithm (in the spirit of the recent technique of [8]), or to find any other deterministic method for this problem. (For the diameter problem, even if we had such an algorithm, there would still remain the issue of designing an efficient parallel version of it, and even then we would still lose a couple of logarithmic factors in the application of parametric searching. Still, the intersection problem is an intriguing challenge, which we pose as an open problem of independent interest.)

3. Width in 3-Space

In this section we consider the problem of computing the *width* of a point set in 3-space. Given a set $\mathcal{P}$ of n points in 3-space, its width is the smallest distance between a pair of parallel planes with the property that all points of $\mathcal{P}$ are contained in the closed slab bounded by the planes. In two dimensions the width can be trivially computed in time $O(n \log n)$, or even in linear time if the convex hull of $\mathcal{P}$ is given. However, in three dimensions the problem becomes harder, and the best-known algorithms for computing width take $O(n^2)$ time [18]. In this section we present a deterministic algorithm with running time $O(n^{8/5+\varepsilon})$ for any $\varepsilon > 0$.

It clearly suffices to compute the width of the convex hull of $\mathcal{P}$, so we may assume, without loss of generality, that the points of $\mathcal{P}$ are in convex position,

and that the structure of their convex hull, denoted $conv(\mathcal{P})$, is known (it takes only $O(n \log n)$ time to compute the hull). It is easy to verify that any two planes defining the width are such that either one touches $conv(\mathcal{P})$ at a face and one at a vertex, or each of the planes touches $conv(\mathcal{P})$ at an edge.

The first case is easy to handle. For each face F of $K \equiv conv(\mathcal{P})$ let $\mathbf{n}_F$ denote the outward unit normal to F. We can draw on the unit sphere S^2 a spherical map $\mathcal{M}$, known as the *Gaussian diagram* or the *normal diagram* of K, whose vertices are the vectors $\mathbf{n}_F$, whose edges are great circular arcs, each being the locus of the outward normal directions of all planes supporting K at a fixed edge, and whose faces are regions, each being the locus of the outward normal directions to all planes supporting K at a fixed vertex. Given K, it is easy to construct $\mathcal{M}$ and to preprocess it for fast point location in overall $O(n)$ time [16], [19]. Then, for each face F of K, we take the vector $-\mathbf{n}_F$ and locate it in $\mathcal{M}$, thereby obtaining the vertex of K "antipodal to F," namely, the vertex supported by a plane with outward normal direction opposite of that of F. The distance between this supporting plane and the plane containing F gives us a candidate for the width of $\mathcal{P}$, and the smallest such distance, W_0, over all faces F of K, gives us the best candidate for the width attained between a plane supporting a face and a plane supporting a vertex of K. So far, the procedure takes only $O(n \log n)$ time.

The difficult case, however, is to find the smallest distance between a pair of parallel planes supporting K at a pair of antipodal edges, because in the worst case there could be $\Theta(n^2)$ such pairs, and of course we do not want to inspect all of those. To handle this case we apply parametric searching. The corresponding "fixed-size decision problem" is: Given a real parameter $W > 0$, determine whether the width of $\mathcal{P}$ is equal to, or greater than, or smaller than W. If we can obtain efficient serial and parallel algorithms for this decision problem, then application of the parametric searching paradigm will give us an efficient algorithm for computing the width of $\mathcal{P}$.

In what follows we assume that the given W is less than or equal to W_0, for otherwise it is clear that the width of $\mathcal{P}$ is less than W. Let $\mathcal{M}'$ denote the spherical map $\mathcal{M}$ reflected through the origin, and consider the superposition of $\mathcal{M}$ and $\mathcal{M}'$. By symmetry, it suffices to superimpose the top parts of $\mathcal{M}$ and $\mathcal{M}'$, i.e., their portions within the hemisphere $z \geq 0$. Each intersection between an edge of $\mathcal{M}$ and an edge of $\mathcal{M}'$ gives us a direction $\mathbf{n}$ for which there exist two parallel planes orthogonal to $\mathbf{n}$ and supporting K at a pair of antipodal edges. If there is any such intersection for which the distance between the corresponding planes is less than W, then the width of $\mathcal{P}$ is clearly less than W; if there is a pair of planes with distance W and no pair with distance less than W, then the width is equal to W; otherwise the width is greater than W.

We solve the decision problem in two stages. In the first stage we compute, for each edge e of $\mathcal{M}'$, the subset of all edges of $\mathcal{M}$ that intersect it, represented compactly as the disjoint union of a small number of "canonical" subsets, using standard range-searching techniques. This produces a collection of subproblems, each involving a subset $\mathcal{E}$ of the edges of $\mathcal{M}$ and a subset $\mathcal{E}'$ of the edges of $\mathcal{M}'$, with the additional property that each edge of $\mathcal{E}'$ intersects all edges of $\mathcal{E}$. We next map the edges of $\mathcal{E}$ into certain surfaces in 4-space and the edges of $\mathcal{E}'$ into points

in 4-space, which reduces the subproblem to another in which we have to determine whether any of the points lies above any of the surfaces. This latter problem is then solved using the point-location technique of [10].

In more detail we proceed as follows. We centrally project (the top portions of) $\mathcal{M}$ and $\mathcal{M}'$ from the origin onto the plane $z = 1$. In this manner each edge of $\mathcal{M}$ or of $\mathcal{M}'$ is mapped either to a straight line segment or to a straight ray. The first stage is then easy to accomplish: we process the collection of segments and rays obtained from the projection of the edges of $\mathcal{M}$ and of $\mathcal{M}'$ into a *hereditary segment tree* structure, by the technique given in [11]. This produces a collection of pairs $(\mathcal{E}, \mathcal{E}')$, where $\mathcal{E}$ is a canonical subset of the edges of $\mathcal{M}$ represented at some node v of the structure, and $\mathcal{E}'$ is a subset of the edges of $\mathcal{M}'$ corresponding to segments that are stored at v and intersect all the segments that correspond to the members of $\mathcal{E}$. We refer the reader to [11] for more details, but remark that the total number of pairs $(\mathcal{E}, \mathcal{E}')$ that this procedure generates is $O(n \log n)$, and that their overall size is $O(n \log^2 n)$.

Next consider a fixed pair $(\mathcal{E}, \mathcal{E}')$ of subsets of edges produced by the first stage. We know that each edge of $\mathcal{E}$ intersects every edge of $\mathcal{E}'$. The subproblem induced by this pair is to determine whether there is a pair of edges, $e \in \mathcal{E}$, $e' \in \mathcal{E}'$, so that the (unique) pair of parallel planes passing through the corresponding pair of edges, s, s', of K are at distance $\leq W$. This subproblem is not affected by assuming that s and s' are full lines in 3-space. Moreover, we can also ignore the constraints that limit the amount by which these planes can rotate about s or s' (originally, these planes could rotate only between the two faces incident to s or to s'); this is an easy consequence of the property that all pairs of edges in $\mathcal{E}$ and $\mathcal{E}'$ intersect, which ensures that no spurious pairs of planes will arise if the constraints are dropped. Finally, it is easy to check that each line corresponds to an edge of $\mathcal{E}$ lies above all the lines that correspond to the edges of $\mathcal{E}'$.

With all these relaxations, we can parametrize each edge e of $\mathcal{E}$ (or of $\mathcal{E}'$) by four real parameters, namely those that define the line in 3-space containing the corresponding edge of K. We choose these parameters (x_1, x_2, x_3, x_4) in such a way that the last parameter x_4 determines the distance by which the line has to be translated in the positive z-direction from some restricted position determined by the first three parameters. This allows us to map the edges of $\mathcal{E}$ into points in 4-space. Each edge $e \in \mathcal{E}'$, corresponding to an edge s of K, is mapped into a surface σ_e in 4-space which is the locus of all points w corresponding to lines in 3-space which lie above the line l containing s and whose distance from l is exactly W. Removal of σ_e partitions 4-space into two subsets, one, denoted σ_e^-, consisting of points for which the corresponding lines either lie below l, or lie above and the interline distance is $< W$ and the other, denoted σ_e^+, consisting of points whose corresponding lines lie above l and whose distance from l is $> W$. Our choice of the parameter x_4 implies that σ_e is *monotone* in the $x_1x_2x_3$-direction (meaning that it is the graph of a function $x_4 = f(x_1, x_2, x_3)$) and that σ_e^+ is the half-space that lies above σ_e (in the x_4-sense). (The definition and properties of σ_e have to be slightly restated for lines whose xy-projections are parallel to that of l, but this will not concern us in the algorithm that follows.) Symmetrically, we can map the edges of $\mathcal{E}'$ into points in 4-space, and the edges of $\mathcal{E}$ into surfaces, defined in a

symmetric manner, with the direction of the z-axis reversed. It is also easy to verify that the surfaces σ_e (in either case) are all algebraic of small constant degree, assuming appropriate representation of lines in space as points in 4-space.

We thus obtain a collection of $a = |\mathscr{E}'|$ surfaces and $b = |\mathscr{E}|$ points in 4-space, and our goal is to determine the relative position of the points with respect to the surfaces. More precisely, we wish to determine whether any point lies in the union of the regions σ_e^-, over all $e \in \mathscr{E}$; instead, we use the contrapositive variant: determine whether all the points lie in the region above (or on) the *upper envelope* of the given surfaces. For this we preprocess the arrangement of the surfaces σ_e into a data structure that supports fast point-location queries. We follow the partitioning technique of [10], but adjust it to the particular structure that arises in our problem.

We construct a partition of 4-space as follows. We fix some sufficiently large constant parameter r, and choose a sample R of $O(r \log r)$ of the given surfaces. We now form the arrangement of the surfaces in R and triangulate the cell of the arrangement that lies above the upper envelope of R (this is where we deviate from the technique of [10], where the entire arrangement is triangulated; actually, to be precise, we should speak of a stratification and not a triangulation, but this is a technicality which is of no importance here, because r is a constant.) To do so, we fix a surface $\sigma \in R$, and intersect it with all the other surfaces of R to obtain a system of two-dimensional surfaces within σ. We now triangulate the three-dimensional arrangement that these surfaces induce within σ. The vertical decomposition technique of [10], in three dimensions, yields a triangulation into $O(r^{3+\varepsilon})$ cells, each having constant-description complexity. We now erect, for each resulting cell c, a semi-infinite vertical cylinder c^+ bounded from below by c, and discard those cylinders that are crossed by another surface of R. The surviving cylinders, over all surfaces $\sigma \in R$, give us the desired triangulation of the uppermost cell of the arrangement of R; the number of cylinders is clearly $O(r^{4+\varepsilon})$. We refer the reader to [10] for more details and for comparison between our technique and the more general technique given there, which produces $O(r^{5+\varepsilon})$ cells (for the entire arrangement).

Next, we partition the given points among the resulting cylinders (halting the algorithm immediately if any point lies outside all cylinders), and recurse in each cylinder with its associated set of points and the set of surfaces that cross it. Let us look at this operation more closely. We choose R to be a $(1/r)$-net, of size $O(r \log r)$, of the set of given surfaces. (The relevant range space with respect to which the net is constructed consists of ranges each being the subset of surfaces intersecting a cylindrical cell of the form obtained in the vertical decomposition procedure described above.) This can be done in linear time by random sampling or, in a deterministic fashion, by using the recent technique of [22]. It follows that the triangulation of the uppermost cell consists of $O(r^{4+\varepsilon})$ cylindrical cells, each being crossed by at most a/r surfaces.

Thus, we obtain a collection of $O(r^{4+\varepsilon})$ subproblems, where the subproblem corresponding to the ith cylinder involves at most a/r surfaces and b_i points (those lying in the cell), and the goal is to determine whether all these points lie above all these surfaces. We solve each subproblem recursively using the same partition-

ing technique. The recursion bottoms out when the number a_i of surfaces is greater than the fourth power of the number b_i of points. In this case we flip the roles of $\mathscr{E}$ and $\mathscr{E}'$, turning the given surfaces into points and points into surfaces, and then we apply the following technique.

For each group of b_i surfaces (formerly points), we apply the cell decomposition technique described above. This time, however, we conceptually reverse the x_4-axis, so that we now compute and triangulate the cell below the lower envelope. We locate the a_i points (formerly surfaces) inside each cell of the triangulation, and proceed recursively from there (not flipping points and surfaces anymore). Adapting the analysis of [10], it easily follows that the resulting procedure requires $O(b_i^{4+\varepsilon})$ preprocessing time and storage, for any $\varepsilon > 0$, and supports point-location queries (each determining whether a given query point lies below all surfaces of b_i) in $O(\log b_i)$ time per query. We thus obtain an algorithm whose running time is $O(b_i^{4+\varepsilon} + a_i \log b_i)$, which, by assumption, is only $O(a_i^{1+\varepsilon})$.

The running time of the algorithm is thus easily seen to be governed by the recurrence

$$T(a, b) \leq \sum_{i=1}^{O(r^{4+\varepsilon})} T\left(\frac{a}{r}, b_i\right) + O(a + b) \qquad \text{if} \quad a < b^4,$$

$$T(a, b) = O(a^{1+\varepsilon}) \qquad \text{if} \quad a \geq b^4.$$

The solution of this recurrence is

$$T(a, b) = O(a^{4/5+\varepsilon} b^{4/5+\varepsilon} + a^{1+\varepsilon} + b^{1+\varepsilon}).$$

We now apply this procedure to each pair $(\mathscr{E}, \mathscr{E}')$ obtained in the first stage of the algorithm. Since the total number of pairs is $O(n \log n)$ and their overall size is $O(n \log^2 n)$, it is not difficult to show that the overall cost of these applications is $O(n^{8/5+\varepsilon})$ for a slightly larger, but still arbitrarily small, ε. Hence, it can be determined whether the width of $\mathscr{P}$ is greater than, or equal to, or smaller than a given parameter W, in deterministic time $O(n^{8/5+\varepsilon})$ for any $\varepsilon > 0$.

To apply parametric searching we also need to derive a parallel version of the algorithm (in Valiant's model). Note that the first stage does not invoke parametric searching and thus need not be parallelized. The second stage, which depends on the parameter W, is easy to parallelize. As in the preceding section, the construction of the $(1/r)$-nets can be done in parallel in polylogarithmic time per net. Since r is a constant, the partitioning of 4-space described above can be constructed in constant parallel time, and the subproblems that are generated can then be solved in parallel. This requires $O(n^{8/5+\varepsilon})$ processors and polylogarithmic parallel time. We omit the more refined details of this parallelization, and, plugging all this into the parametric search paradigm, we conclude, in summary:

Theorem 3.1. *Given a set $\mathscr{P}$ of n points in 3-space, the width of $\mathscr{P}$ can be computed (deterministically) in time $O(n^{8/5+\varepsilon})$ for any $\varepsilon > 0$.*

4. Closest Line Pair in 3-Space

In this section we consider the following problem. Let $\mathscr{L}$ be a given set of n lines in 3-space. The distance $d(l_1, l_2)$ between two lines is defined as $\min_{p \in l_1, q \in l_2} d(p, q)$, where $d(p, q)$ is the Euclidean distance between the points p and q. The *closest-line-pair* problem calls for computing

$$\min_{l_1, l_2 \in \mathscr{L}} d(l_1, l_2).$$

This problem arises in many applications, for example in proximity detection between airplane flight paths. In this section we provide an algorithm, based on parametric searching, that computes the closest line pair in time $O(n^{8/5+\varepsilon})$ for any $\varepsilon > 0$.

As above, the first task is to solve the fixed-size problem: Given a set $\mathscr{L}$ of n lines in space and a parameter $d > 0$, determine whether there is a pair of lines at distance less that d apart, or, if not, whether there is a pair at distance exactly d apart.

This problem is solved using a technique very similar to that of the preceding section. We first split $\mathscr{L}$ into two subsets $\mathscr{L}_1, \mathscr{L}_2$ of roughly equal size. We test recursively whether any pair of lines in $\mathscr{L}_1$ lie at distance $< d$ (or $= d$) apart, and similarly for $\mathscr{L}_2$, and are left with a "bichromatic" version of the problem, namely to determine whether there is a pair of lines $l_1 \in \mathscr{L}_1, l_2 \in \mathscr{L}_2$ at distance $< d$ or $= d$ apart. For technical reasons, similar to those in the preceding section, we only study the restricted version of the problem, in which we also assume that all lines in $\mathscr{L}_1$ lie below all lines in $\mathscr{L}_2$. To ensure that this can be done with no loss of generality, we proceed as follows.

We project all lines of $\mathscr{L}$ onto the xy-plane, orient the projections from left to right, and partition $\mathscr{L}$ so that all lines in $\mathscr{L}_1$ have projections with slope smaller than those of the projections of the lines in $\mathscr{L}_2$. Then, as shown in [9], a line $l_1 \in \mathscr{L}_1$ lies below a line $l_2 \in \mathscr{L}_2$ if and only if the Plücker point image of l_1 lies (in 5-space) on a given side of the Plücker hyperplane image of l_2. Excluding special cases which we can handle directly, we can readily find a direction in 5-space, which by convention we call vertical, corresponding to that side. Thus we map the lines of $\mathscr{L}_1$ to Plücker points and the lines of $\mathscr{L}_2$ to Plücker hyperplanes. We next choose a sufficiently large constant parameter r, construct a $(1/r)$-net R of size $O(r \log r)$ in the collection of hyperplanes (with respect to ranges, each being the set of hyperplanes crossing some simplex), form the arrangement of R, extract from it the zone of the (quadric) Plücker surface (on which all our a points lie), and triangulate the zone cells into simplices. By the recent results of [7], the number of resulting simplices is $O(r^4 \log^5 r)$. Each simplex s is crossed by at most b/r hyperplanes, where $b = n/2$ is the size of $\mathscr{L}_2$, and all other hyperplanes pass fully above or fully below it. We thus obtain, for each simplex s, three subproblems, involving, respectively,

(i) the points in s and the hyperplanes passing above s,
(ii) the points in s and the hyperplanes passing below s, and
(iii) the points in s and the hyperplanes crossing s.

The first two subproblems are solved directly, using the technique to be described in a moment, because in each of them we have, when passing back to 3-space, two sets of lines, with all lines in one set lying above all lines in the other. The third subproblem is solved recursively, for each of the simplices.

We thus have reduced the problem to the case where we have two sets of lines, $\mathscr{L}'$, $\mathscr{L}''$, consisting of, say, p lines and q lines, respectively, so that each line in $\mathscr{L}'$ passes below every line of $\mathscr{L}''$, and our goal is to determine whether there is any pair of lines, $l' \in \mathscr{L}'$, $l'' \in \mathscr{L}''$, at distance $< d$ (or $= d$) apart. However, this is exactly the subproblem that we ended up with in the preceding section, so we can solve it in time $O(p^{4/5+\varepsilon}q^{4/5+\varepsilon} + p^{1+\varepsilon} + q^{1+\varepsilon})$ for any $\varepsilon > 0$. Since in the preceding partitioning we have only a constant number of such subproblems (of types (i) or (ii)), it follows that the cost of all these nonrecursive subproblems is $O(a^{4/5+\varepsilon}b^{4/5+\varepsilon} + a^{1+\varepsilon} + b^{1+\varepsilon})$, for any $\varepsilon > 0$, where $a = |\mathscr{L}_1|$ and $b = |\mathscr{L}_2|$.

We derive that the overall time $T(a, b)$ for solving the bichromatic problem involving two sets of lines with a and b lines, respectively, satisfies the recurrence

$$T(a, b) = O(a^{4/5+\varepsilon}b^{4/5+\varepsilon} + a^{1+\varepsilon} + b^{1+\varepsilon}) + \sum_{i=1}^{O(r^4 \log^5 r)} T(a_i, b_i),$$

where, for each i, we have $b_i \leq b/r$ and $\sum_i a_i = a$. It then easily follows that

$$T(a, b) = O(a^{4/5+\varepsilon}b^{4/5+\varepsilon} + a^{1+\varepsilon} + b^{1+\varepsilon})$$

for any $\varepsilon > 0$. Hence, the time $T'(n)$ required for solving the original problem involving a single set of n lines satisfies the recurrence

$$T'(n) = 2T'\left(\frac{n}{2}\right) + O(n^{8/5+\varepsilon})$$

whose solution is $T'(n) = O(n^{8/5+\varepsilon})$ for any $\varepsilon > 0$. We can thus conclude that the fixed-size problem can be solved (deterministically) in time $O(n^{8/5+\varepsilon})$ for any $\varepsilon > 0$, and can also be solved by a parallel algorithm (in Valiant's model) which takes polylogarithmic parallel time and uses $O(n^{8/5+\varepsilon})$ processors. Plugging all these into the machinery of parametric searching, we obtain, as in the previous section:

Theorem 4.1. *Given a set $\mathscr{L}$ of n lines in 3-space, the closest pair in $\mathscr{L}$ (under the Euclidean distance between lines in space) can be found (deterministically) in time $O(n^{8/5+\varepsilon})$ for any $\varepsilon > 0$.*

5. Discussion

In this paper we have investigated applications of parametric searching to geometric problems. We have applied the technique to several basic problems in

three-dimensional computational geometry, namely the problems of computing the diameter and width of a point set in 3-space and of computing the closest line pair in a set of lines in 3-space. In addition to the parametric-searching technique itself, our solutions also make use of a battery of recent space-partitioning techniques for range searching and point location, and of techniques for manipulating arrangements of lines in 3-space.

Although our solutions constitute significant improvements over previous ones (in the diameter problem, over previous deterministic solutions), they are probably not optimal, and their improvements remain open problems. The most intriguing such problem, from a theoretical point of view, is to obtain a deterministic algorithm for the three-dimensional diameter problem which runs in $O(n \log n)$ time.

Acknowledgments

The research reported in this paper was done when Bernard Chazelle, Herbert Edelsbrunner, and Micha Sharir visited Leo Guibas at the DEC Systems Research Center in Palo Alto, CA. The authors would like to thank the Systems Research Center for its hospitality and support of the research. The authors also wish to thank Pankaj Agarwal and Jirka Matoušek for helpful discussions which led to improvements of some of the results.

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, pp. 321–331. Also in *Algorithmica* **9** (1993), 495–514.
2. P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minium spanning trees and bichromatic closest pairs, *Discrete Comput. Geom.* **6** (1991), 407–422.
3. P. K. Agarwal, A. Efrat, M. Sharir, and S. Toledo, Computing a segment center for a planar point set, *J. Algorithms*, to appear.
4. P. K. Agarwal and J. Matoušek, Ray shooting and parametric search, *Proc. 24th Ann. ACM Symp. on Theory of Computing*, 1992, pp. 517–526.
5. P. K. Agarwal and M. Sharir, Planar geometric location problems. *Proc. 2nd ACM Symp. on Discrete Algorithms*, 1991, pp. 449–458. (Also to appear in *Algorithmica*.)
6. P. K. Agarwal, M. Sharir, and S. Toledo, New applications of parametric searching in computational geometry, *Proc. 3rd ACM–SIAM Symp. on Discrete Algorithms*, 1992, pp. 72–82.
7. B. Aronov, M. Pellegrini, and M. Sharir, On the zone of a surface in a hyperplane arrangement, *Discrete Comput. Geom.* **9** (1993), 177–186.
8. B. Chazelle, An optimal convex hull algorithm and new results on cuttings, *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, 1991, pp. 29–38.
9. B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Lines in space—combinatorics, algorithms and applications, *Proc. 20th Ann. ACM Symp. on Theory of Computing*, 1989, pp. 382–393.
10. B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, A singly exponential stratification scheme for real semi-algebraic varieties and its applications, *Proc. 16th ICALP*, July 1989, pp. 179–193, LNCS 372, Springer-Verlag, Berlin. Also in *Theoret. Comput. Sci.* **84** (1991), 77–105.
11. B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, *Algorithmica*, to appear.
12. B. Chazelle, and J. Matoušek, On linear-time deterministic algorithms for optimization problems in fixed dimension, *Proc. 4th Ann. ACM–SIAM Symposium on Discrete Algorithms*, 1993.

13. K. Clarkson and P. Shor, Applications of random sampling to computational geometry, *II*, *Discrete Comput. Geom.* **4** (1989), 387–421.
14. R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, *J. Assoc. Comput. Mach.* **34** (1987), 200–208.
15. R. Cole, M. Sharir, and C. Yap, On k-hulls and related problems, *SIAM J. Comput.* **16** (1987), 61–77.
16. H. Edelsbrunner, L. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.* **15** (1986), pp. 317–340.
17. D. Haussler and E. Welzl, ε-nets and simplex range queries, *Discrete Comput. Geom.* **2** (1987), 127–151.
18. M. Houle and G. Toussaint, Computing the width of a set, *Proc. 1st ACM Symp. on Computational Geometry*, 1985, pp. 1–7.
19. D. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Comput.* **12** (1983), 28–35.
20. J. Matoušek, Efficient partition trees, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 1–9.
21. J. Matoušek, Randomized optimal algorithm for slope selection, *Inform. Process. Lett.* **39** (1991), 183–187.
22. J. Matoušek, Approximations and optimal geometric divide-and-conquer, *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, pp. 1–10.
23. J. Matoušek, Computing the center of planar point sets, in *Discrete and Computational Geometry DIMACS* (J. E. Goodman, R. Pollack, and W. Steiger, eds.), American Mathematical Society, Providence, RI, 1991, pp. 221–230.
24. J. Matoušek, Linear optimization queries, *J. Algorithms*, to appear.
25. N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. Assoc. Comput. Mech.* **30** (1983), 852–865.
26. N. Naor and M. Sharir, Computing a point in the center of a point set in three dimensions. *Proc. 2nd Canadian Conference on Computational Geometry*, 1990, pp. 10–13.
27. A. Stein and M. Werman, Finding the repeated median regression line, *Proc. 3rd ACM–SIAM Symp. on Discrete Algorithms*, 1992, pp. 409–413.
28. S. Toledo, Extremal polygon placement problems, *Proc. 7th ACM Symp. on Computational Geometry*, (1991), pp. 176–185.
29. L. Valiant, Parallelism in comparison problems, *SIAM J. Comput.* **4** (1975), 348–355.

Received January 3, 1992, and in revised form December 1, 1992.