CS-1996-19

Efficient Algorithms for Geometric Optimization

Pankaj K. Agarwal Micha Sharir

Department of Computer Science

Duke University

Durham, North Carolina 27708-0129

January 24, 1997

Efficient Algorithms for Geometric Optimization[*†]

Pankaj K. Agarwal [‡] Micha Sharir [§]

January 24, 1997

Abstract

We review the recent progress in the design of efficient algorithms for various problems in geometric optimization. We present several techniques used to attack these problems, such as parametric searching, geometric alternatives to parametric searching, prune-and-search techniques for linear programming and related problems, and LP-type problems and their efficient solution. We then describe a variety of applications of these and other techniques to numerous problems in geometric optimization, including facility location, proximity problems, statistical estimators and metrology, placement and intersection of polygons and polyhedra, and ray shooting and other query-type problems.

[*]Both authors are supported by a grant from the U.S.-Israeli Binational Science Foundation. Pankaj Agarwal has also been supported by National Science Foundation Grant CCR-93–01259, by an Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, and by an NYI award and matching funds from Xerox Corp. Micha Sharir has also been supported by NSF Grants CCR-94-24398 and CCR-93-11127, by a Max-Planck Research Award, and by a grant from the G.I.F., the German-Israeli Foundation for Scientific Research and Development.

[†]A preliminary version of this paper appeared as: P. K. Agarwal and M. Sharir, Algorithmic techniques for geometric optimization, in *Computer Science Today: Recent Trends and Developments, Lecture Notes in Computer Science*, vol. 1000 (J. van Leeuwen, ed.), 1995, pp. 234–253.

[‡]Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA.

[§]School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, ISRAEL; and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA.

1 Introduction

Combinatorial optimization deals with problems of maximizing or minimizing a function of one or more variables subject to a large number of inequality (and equality) constraints. Many problems can be formulated as combinatorial optimization problems, which has made this a very active area of research during the past half century. In many applications, the underlying optimization problem involves a constant number of variables and a large number of constraints that are are induced by a given collection of geometric objects; we refer to such problems as *geometric optimization* problems. In such cases one expects that faster and simpler algorithms can be developed by exploiting the geometric nature of the problem. Much work has been done on geometric optimization problems during the last twenty years. Many new elegant and sophisticated techniques have been developed and successfully applied to a variety of geometric optimization problems, and the aim of this paper is to survey the main techniques and applications of this kind.

This survey consist of two parts. The first part describes several general techniques that have led to efficient algorithms for a variety of geometric optimization problems, the most notable of which is linear programming. The second part lists many geometric applications of these techniques, and discusses some of them in more detail.

The first technique that we present is called *parametric searching*. Although restricted versions of parametric searching already existed earlier (see e.g. [97]), the full-scale technique was presented by Megiddo in the late 1970's and early 1980's [179, 180]. The technique was originally motivated by so-called *parametric optimization* problems in combinatorial optimization, and did not receive much attention by the computational geometry community until the late 1980's. In the last seven years, though, it has become one of the major techniques for solving geometric optimization problems efficiently. We outline the technique in detail in Section 2, first exemplifying it on the *slope-selection problem* [67], and then presenting various extensions of the technique.

Despite its power and versatility, parametric searching has certain drawbacks, which we discuss next. Consequently, there have been several recent attempts to replace parametric searching by alternative techniques, including *randomization* [14, 64, 170], *expander graphs* [27, 151, 153, 154], *geometric cuttings* [17, 43], and *matrix searching* [105, 107, 108, 106, 111]. We present these alternative techniques in Section 3.

Almost concurrently with the development of the parametric searching technique, Megiddo devised another ingenious technique for solving linear programming and several related optimization problems [181, 183]. This technique, now known as *decimation* or *prune-and-search*, was later refined and extended by Dyer [85], Clarkson [59], and others. The technique can be viewed as an optimized version of parametric searching, in which certain special properties of the problem allows one to improve further the efficiency of the algorithm. For example, this technique yields linear-time deterministic algorithms for *linear programming*

and for several related problems, including the *smallest-enclosing-ball* problem, when the dimension is fixed. (However, the dependence of the running time of these algorithms on the dimension is at least exponential.) We illustrate the technique in Section 4 by applying it to linear programming.

In the past decade, *randomized algorithms* have been developed for a variety of problems in computational geometry and in other fields; see, e.g., the books by Mulmuley [194] and by Motwani and Raghavan [193]. Clarkson [62] and Seidel [211] gave randomized algorithms for linear programming, whose expected time is linear in any fixed dimension, which are much simpler than their earlier deterministic counterparts. The dependence on the dimension of the running time of these algorithms is better (though still exponential). Actually, Clarkson's technique is rather general, and is also applicable to a variety of other geometric optimization problems. We describe this technique in Section 5.

Another significant progress on linear programming was made about four years ago, when new randomized algorithms for linear programming were obtained independently by Kalai [149], and by Matoušek et al. [178, 218] (these two algorithms are essentially dual versions of the same technique). The expected number of arithmetic operations performed by these algorithms is 'subexponential' in the input size, and is still linear in any fixed dimension, so they constitute an important step toward the still open goal of obtaining strongly polynomial algorithms for linear programming. (Recall that the polynomial-time algorithms by Khachiyan [155] and by Karmarkar [150] are not strongly polynomial, as the number of arithmetic operations performed by these algorithms depends on the size of the coefficients of the input constraints.) This new technique is presented in Section 6. The algorithm in [178, 218] is actually formulated in a general abstract framework, which fits not only linear programming but many other problems. Such "LP-type" problems are also reviewed in Section 6, including the connection, recently noted by Amenta [32, 33], between abstract linear programming and "Helly-type" theorems.

In the second part of this paper, we survey many geometric applications of the techniques described in the first part. These applications include problems involving *facility location* (e.g., finding p congruent disks of smallest possible radius whose union covers a given planar point set), *geometric proximity* (e.g., computing the diameter of a point set in three dimensions), *statistical estimators and metrology* (e.g., computing the smallest-width annulus that contains a given planar point set), *placement and intersection of polygons and polyhedra* (e.g., finding the largest similar copy of a convex polygon that fits inside a given polygonal environment), and *query-type problems* (e.g., the *ray-shooting* problem, in which we want to preprocess a given collection of objects, so that the first object hit by a query ray can then be determined efficiently).

Numerous non-geometric optimization problems have also benefited from the techniques presented here (see [21, 65, 105, 120, 196] for a sample of such applications), but we will concentrate only on geometric applications.

Although the common theme of most of the applications reviewed here is that they can be solved efficiently using parametric-searching, prune-and-search, LP-type, or related techniques, each of them requires a problem-specific, and often fairly sophisticated, approach. For example, the heart of a typical application of parametric searching is the design of efficient sequential and parallel algorithms for solving the appropriate problem-specific 'decision procedure' (see below for details). We will provide details of these solutions for some of the problems, but will have to suppress them for most of the applications.

PART I: TECHNIQUES

The first part of the survey describes several techniques commonly used in geometric optimization problems. We describe each technique and illustrate it by giving an example.

2 Parametric Searching

We begin by outlining the parametric-searching technique, and then illustrate the technique by giving an example where this technique is applied. Finally, we discuss various extensions of parametric searching.

2.1 Outline of the Technique

The parametric searching technique of Megiddo [179, 180] 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 $\mathcal{P}(\lambda)$ that depends on a real parameter λ, and is *monotone* in λ, meaning that if $\mathcal{P}(\lambda_0)$ is true for some λ_0, then $\mathcal{P}(\lambda)$ is true for all $\lambda < \lambda_0$. Our goal is to find λ^*, the maximum λ for which $\mathcal{P}(\lambda)$ is true, assuming such a maximum exists. Suppose further that $\mathcal{P}(\lambda)$ can be solved by a (sequential) algorithm A_s that takes λ and a set of data objects (independent of λ) as the input, and that, as a by-product, A_s can also determine whether the given λ is equal to, smaller than, or larger than the desired value $\lambda^\star$. Assume moreover that the control flow of A_s is governed by comparisons, each of which amounts to testing the sign of some low-degree polynomial in λ.

Megiddo's technique then runs A_s 'generically' at the unknown optimum λ^* and maintains an open interval I that is known to contain λ^*. Initially, I is the whole line. Whenever A_s reaches a branching point that depends on some comparison with an associated polynomial $p(\lambda)$, it computes all the roots $\lambda_1, \lambda_2, \ldots$ of p and computes $\mathcal{P}(\lambda_i)$ by running (the standard, non-generic version of) A_s with the value of $\lambda = \lambda_i$. If one of the λ_i is equal to $\lambda^\star$, we stop since we have found the value of $\lambda^\star$. Otherwise, we have determined the

open interval $(\lambda_i, \lambda_{i+1})$ that contains λ^*. Since the sign of $p(\lambda)$ remains the same for all $\lambda \in (\lambda_i, \lambda_{i+1})$, we can compute the sign of $p(\lambda^*)$ by evaluating, say, $p((\lambda_i + \lambda_{i+1})/2)$. The sign of $p(\lambda^*)$ also determines the outcome of the comparison at $\lambda^\star$. We now set I to be $I \cap (\lambda_i, \lambda_{i+1})$, and the execution of the generic A_s is resumed. As we proceed through this execution, each comparison that we resolve constrains the range where $\lambda^\star$ can lie even further; and we thus obtain a sequence of progressively smaller intervals, each known to contain $\lambda^\star$, until we either reach the end of A_s with a final interval I, or hit $\lambda^\star$ at one of the comparisons of A_s. In practically all applications, the generic algorithm will always make a comparison whose associated polynomial vanishes at $\lambda^\star$, which will then cause the overall algorithm to terminate with the desired value of $\lambda^\star$.

If A_s runs in time T_s and makes C_s comparisons, then, in general, the cost of the procedure just described is $O(C_s T_s)$, and is thus generally quadratic in the original complexity. To speed up the execution, Megiddo proposes to implement the generic algorithm by a parallel algorithm A_p (under Valiant's *comparison model* of computation [228]; see below). If A_p uses P processors and runs in T_p parallel steps, then each parallel step involves at most P *independent* comparisons; that is, we do not need to know the outcome of such a comparison to be able to execute other comparisons in the same 'batch'. We can then compute the $O(P)$ roots of all the polynomials associated with these comparisons, and perform a binary search to locate λ^* among them, using (the non-generic) A_s at each binary step. The cost of simulating a parallel step of A_p is thus $O(P + T_s \log P)$ (there is no need to sort the $O(P)$ roots; instead, one can use repeated median finding, whose overall cost is only $O(P)$), for a total running time of $O(PT_p + T_p T_s \log P)$. In most cases, the second term dominates the running time.

This technique can be generalized in many ways. For example, given a concave function $f(\lambda)$, we can compute the value λ^* of λ that maximizes $f(\lambda)$, provided we have sequential and parallel algorithms for the 'decision' problem (of comparing λ^* with a specific λ); these algorithms compute $f(\lambda)$ and its derivative, from which the relative location of the maximum of f is easy to determine. We can compute λ^* even if the decision algorithm A_s cannot distinguish between, say, $\lambda < \lambda^*$ and $\lambda = \lambda^*$; in this case we maintain a half-closed interval I containing λ^*, and the right endpoint of the final I gives the desired value λ^*. See [13, 68, 225, 226] for these and some other generalizations.

It is instructive to observe that parallelism is used here in a very weak sense, since the overall algorithm remains sequential and just simulates the parallel algorithm. The only feature that we require is that the parallel algorithm performs only a small number of batches of comparisons, and that the comparisons in each batch are independent of each other. We can therefore assume that the parallel algorithm runs in the parallel comparison model of Valiant [228], which measures parallelism only in terms of the comparisons being made, and ignores all other operations, such as bookkeeping, communication between the processors, and data allocation. Moreover, any portion of the algorithm that is independent of λ can be performed sequentially in any manner we like. These observations simplify the

technique considerably in many cases.

2.2 An example: The slope-selection problem

As an illustration, consider the *slope selection* problem, which we formulate in a dual setting, as follows: We are given a set L of n nonvertical lines in the plane and an integer $1 \leq k \leq \binom{n}{2}$, and we wish to find an intersection point between two lines of L that has the k-th smallest x-coordinate. (We assume, for simplicity, *general position* of the lines, so that no three lines are concurrent, and no two intersection points have the same x-coordinate.) We are thus seeking the k-th leftmost vertex of the *arrangement* $\mathcal{A}(L)$ of the lines in L; see [93, 216] for more details concerning arrangements. (The name of the problem comes from its primal setting, where we are given a set of n points and a parameter k as above, and wish to determine a segment connecting two input points that has the k-th smallest slope among all such segments.)

We define $\mathcal{P}(\lambda)$ to be true if the x-coordinates of at most k vertices of $\mathcal{A}(L)$ are smaller than or equal to λ. Obviously, $\mathcal{P}(\lambda)$ is monotone, and λ^*, the maximum value of λ for which $\mathcal{P}(\lambda)$ is true, is the x-coordinate of the desired vertex. After having computed λ^*, the actual vertex is rather easy to compute, and in fact the algorithm described below can compute the vertex without any additional work. In order to apply the parametric-searching technique, we need an algorithm that, given a vertical line $\ell : x = \lambda$, can compare λ with λ^*. If we denote by k_λ the number of vertices of $\mathcal{A}(L)$ that lie to the left of (or on) the line $x = \lambda$, then, clearly, we have $\lambda^* < \lambda$ (resp. $\lambda^* > \lambda$, $\lambda^* = \lambda$) if and only if $k_\lambda > k$ (resp. $k_\lambda < k$, $k_\lambda = k$).

Let $(\ell_1, \ell_2, \ldots, \ell_n)$ denote the sequence of lines in L sorted in the decreasing order of their slopes, and let $(\ell_{\pi(1)}, \ell_{\pi(2)}, \ldots, \ell_{\pi(n)})$ denote the sequence of these lines sorted by their intercepts with $x = \lambda$. An easy observation is that two lines ℓ_i, ℓ_j, with $i < j$, intersect to the left of $x = \lambda$ if and only if $\pi(i) > \pi(j)$. In other words, the number of intersection points to the left of $x = \lambda$ can be counted, in $O(n \log n)$ time, by counting the number of *inversions* in the permutation π, using a straightforward tree-insertion procedure [156]. Moreover, we can implement this inversion-counting procedure by a parallel sorting algorithm that takes $O(\log n)$ parallel steps and uses $O(n)$ processors (e.g., the one in [26]). Hence, we can count the number of inversions in $O(n \log n)$ time sequentially or in $O(\log n)$ parallel time using $O(n)$ processors. Plugging these algorithms into the parametric-searching paradigm, we obtain an $O(n \log^3 n)$-time algorithm for the slope-selection problem.

2.3 Improvements and extensions

Cole [66] observed that in certain applications of parametric searching, including the slope selection problem, the running time can be improved to $O((P + T_s)T_p)$, as follows. Consider a parallel step of the above generic algorithm. Suppose that, instead of invoking the decision

procedure $O(\log n)$ times in this step, we call it only $O(1)$ times, say, three times. This will determine the outcome of 7/8 of the comparisons, and will leave 1/8 of them unresolved (we assume here, for simplicity, that each comparison has only one critical value of λ where its outcome changes; this is the case in the slope-selection problem). Suppose further that each of the unresolved comparisons can influence only a constant (and small) number (say, two) of comparisons executed at the next parallel step. Then 3/4 of these comparisons can still be simulated generically with the currently available information. This leads to a modified scheme that mixes the parallel steps of the algorithm, since we now have to perform together new comparisons and yet unresolved old comparisons. Nevertheless, Cole shows that if carefully implemented (by assigning an appropriate time-dependent weight to each unresolved comparison, and by choosing the weighted median at each step of the binary search), the number of parallel steps of the algorithm increases only by an additive logarithmic term, which leads to the improvement stated above. An ideal setup for Cole's improvement is when the parallel algorithm is described as a circuit (or network), each of whose gates has a constant fan-out. Since sorting can be implemented efficiently by such a network [26], Cole's technique is applicable to problems whose decision procedure is based on sorting.

Cole's idea therefore improves the running time of the slope-selection algorithm to $O(n \log^2 n)$. (Note that the only step of the algorithm that depends on λ is the sorting that produces π. The subsequent inversion-counting step is independent of λ, and can be executed sequentially. In fact, this step can be totally suppressed, since it does not provide us with any additional information about λ.) Using additional machinery, Cole et al. [67] gave an optimal $O(n \log n)$-time solution. They observe that one can compare λ^* with a value λ that is 'far away' from λ^* in a faster manner, by counting inversions only approximately. This approximation is progressively refined as λ approaches λ^* in subsequent comparisons. Cole et al. show that the overall cost of $O(\log n)$ calls to the approximating decision procedure is only $O(n \log n)$, so this also bounds the running time of the whole algorithm. This technique was subsequently simplified in [43]. Chazelle et al. [52] have shown that the algorithm of [67] can be extended to compute, in $O(n \log n)$ time, the k-th leftmost vertex in an arrangement of n line segments.

The slope-selection problem is only one of many problems in geometric optimization that have been efficiently solved using parametric searching. See [5, 9, 13, 17, 51, 200] for a sample of other problems, many of which are described in Part II, that benefit from parametric searching.

The parametric searching technique can be extended to higher dimensions in a natural manner. Suppose we have a d-variate (strictly) concave function $F(\lambda)$, where λ varies over $\mathbb{R}^d$. We wish to compute $\lambda^* \in \mathbb{R}^d$ at which $F(\lambda)$ attains its maximum value. Let A_s, A_p be, as above, sequential and parallel algorithms that can compute $F(\lambda_0)$ for any given λ_0. As above, we run A_p generically at λ^*. Each comparison involving λ now amounts to evaluating the sign of a d-variate polynomial $p(\lambda_1, \ldots, \lambda_d)$, and each parallel step requires

resolving P such independent comparisons at λ^*. Resolving a comparison is more difficult because $p(\lambda_1, \ldots, \lambda_d) = 0$ is now a $(d-1)$-dimensional variety. Cohen and Megiddo [65] described a recursive procedure to execute a parallel step for the case in which the polynomial corresponding to each of the comparisons is a linear function. The total running time in simulating A_p, using their procedure, is $2^{O(d^2)} T_s (T_p \log P)^d$; see also [21, 183, 196]. The running time was subsequently improved by Agarwal et al. [17] to $d^{O(d)} T_s (T_p \log P)^d$. Later Toledo [227] extended these techniques to comparisons involving nonlinear polynomials, using Collins's cylindrical algebraic decomposition [69]. The total running time of his procedure is $O(T_s (T_p \log n)^{2^d - 1})$. For the sake of completeness, we present these higher-dimensional extensions in an Appendix.

3 Alternative Approaches to Parametric Searching

Despite its power and versatility, the parametric searching technique has some shortcomings:

(i) Parametric searching requires the design of an efficient parallel algorithm for the generic version of the decision procedure. This is not always easy, even though it only needs to be done in the weak comparison model, and it often tends to make the overall solution quite complicated and impractical.

(ii) The generic algorithm requires exact computation of the roots of the polynomials $p(\lambda)$ whose signs determine the outcome of the comparisons made by the algorithm. In general, the roots of a polynomial cannot be computed exactly, therefore one has to rely either on numerical techniques to compute the roots of $p(\lambda)$ approximately, or on computational-algebra techniques to isolate the roots of $p(\lambda)$ and to determine the sign of $p(\lambda^*)$ without computing the roots explicitly. Both of these alternatives are rather expensive.

(iii) Finally, from an aesthetic point of view, the execution of an algorithm based on parametric searching may appear to be somewhat chaotic. Such an algorithm neither gives any insight to the problem, nor does its execution resemble any 'intuitive' flow of execution for solving the problem.

These shortcomings have led several researchers to look for alternative approaches to parametric searching for geometric optimization problems. Roughly speaking, parametric searching effectively conducts an implicit binary search over a set $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$ of 'critical values' of the parameter λ, to locate the optimum λ^* among them. (For example, in the slope-selection problem, the critical values are the $\Theta(n^2)$ x-coordinates of the vertices of the arrangement $\mathcal{A}(L)$.) The power of the technique stems from its ability to perform the binary search by generating only a small number of critical values during the search, without computing the entire Λ explicitly. In this section we describe some alternative ways of performing such a binary search, which also generates only a small set of critical values.

3.1 Randomization

Randomization is a natural approach to perform an implicit binary search over the critical values [170, 229, 235]. Suppose we know that λ^* lies in some interval $I = [\alpha, \beta]$. Suppose further that we can randomly choose an element $\lambda_0 \in I \cap \Lambda$, where each item is chosen with probability $1/|I \cap \Lambda|$. Then it follows that, by comparing λ^* with a few randomly chosen elements of $I \cap \Lambda$ (i.e., by executing the 'decision' algorithm at these values), we can shrink I to an interval I' that is guaranteed to contain λ^* and that is expected to contain significantly fewer critical values.

The difficult part is, of course, choosing a random element from $I \cap \Lambda$. In many cases, a procedure for computing $|I \cap \Lambda|$ can be converted into a procedure for generating a random element of $I \cap \Lambda$. For example, in the slope-selection problem, given a set L of n lines in the plane and a vertical strip $W = (l, r) \times \mathbb{R}$, an inversion-counting algorithm for computing the number of vertices of $\mathcal{A}(L)$ within W can be used to generate a multiset of q random vertices of $\mathcal{A}(L) \cap W$ in time $O(n \log n + q)$ [170]. Based on this observation, Matoušek [170] obtained the following simple slope-selection algorithm: Each step of the algorithm maintains a vertical strip $W(a, b) = \{(x, y) \mid a \leq x \leq b\}$ that is guaranteed to contain the k-th leftmost vertex; initially $a = -\infty$ and $b = +\infty$. Let m be the number of vertices of $\mathcal{A}(L)$ lying inside W. We repeat the following step until the algorithm terminates. If $m \leq n$, the k-th leftmost vertex of $\mathcal{A}(L)$ can be computed in $O(n \log n)$ by a sweep-line algorithm (through W). Otherwise, set k^* to be the number of vertices lying to the left of the line $x = a$. Let $j = \lfloor (k - k^*) \cdot n/m \rfloor$, $j_a = j - \lfloor 3\sqrt{n} \rfloor$, and $j_b = j + \lfloor 3\sqrt{n} \rfloor$. We choose n random vertices of $\mathcal{A}(L)$ lying inside $W(a, b)$. If the k-th leftmost vertex lies in $W(j_a, j_b)$ and the vertical strip $W(j_a, j_b)$ contains at most $cm/\sqrt{n}$ vertices, for some appropriate constant $c > 0$, we set $a = j_a$, $b = j_b$, and repeat this step. Otherwise, we discard the random sample of vertices, and draw a new sample. It can be shown, using Chernoff's bound [193], that the expected running time of the above algorithm is $O(n \log n)$. Shafer and Steiger [214] gave a slightly different $O(n \log n)$ expected-time algorithm for the slope-selection algorithm. They choose a random subset of $u = O(n \log n)$ vertices of $\mathcal{A}(L)$. Let $a_1, a_2, \ldots, a_u$ be the x-coordinates of these vertices. Using the algorithm by Cole et al. [67] for counting the number of inversions approximately, they determine in $O(n \log n)$ time the vertical strip $W(a_i, a_{i+1})$ that contains the k-th leftmost vertex of $\mathcal{A}(L)$. They prove that, with high probability, $W(a_i, a_{i+1})$ contains only $O(n)$ vertices of $\mathcal{A}(L)$, and therefore the desired vertex can be computed in an additional $O(n \log n)$ time by a sweep-line algorithm. See [77] for yet another randomized slope-selection algorithm. We will mention below a few more applications of this randomized approach.

3.2 Expanders and cuttings

In many cases, the above randomized approach can be derandomized, without affecting the asymptotic running time, using techniques based on *expanders* or on *geometric cuttings*. Expander graphs are special graphs that can be constructed deterministically (in a rather simple manner), have a linear number of edges, and share many properties with random graphs; see [29] for more details. For example, in the slope-selection problem, we can construct expander graphs whose vertices are the given lines and whose edges correspond to appropriate vertices of the arrangement (each edge corresponds to the intersection of the two lines that it connects). If we search among these vertices for the optimal λ^*, we obtain a slab containing λ^* and free of any of the expander-induced vertices. One can then show, using properties of expander graphs, that the number of vertices of $\mathcal{A}(L)$ within the slab is rather small, so that the search for λ^* within the slab can proceed in a more efficient manner. This is similar to the randomized solution in [170]. (The precise construction of the expander graph is somewhat involved, and is described in [154].)

Although expanders have been extensively used in many areas, including parallel computation, complexity theory, communication networks, their applications in computational geometry have been rather sparse so far. Ajtai and Megiddo [27] gave an efficient parallel linear-programming algorithm based on expanders, and later Katz [151] and Katz and Sharir [153, 154] applied expanders to solve several geometric optimization problems, including the application to the slope-selection problem, as just mentioned.

Chazelle et al. [51] developed an $O(n \log^2 n)$-time deterministic algorithm for the slope-selection problem, using *cuttings*.[1] The bound was subsequently improved by Brönnimann and Chazelle [43] to $O(n \log n)$. For the sake of completeness, we give a brief sketch of the algorithm by Chazelle et al. [51]. The algorithm works in $O(\log n)$ stages. In the beginning of the j-th stage, we have a vertical strip W_j, which contains λ^*, and a triangulation T_j of W_j. For each triangle $\triangle \in T_j$, we store the vertices of $\triangle$, the subset $L_\triangle$ of lines in L that intersect the interior of $\triangle$, and the intersection points of $L_\triangle$ with the edges of $\triangle$. We refer to the vertices of T_j and the intersection points of $L_\triangle$ with the edges of $\triangle$, over all $\triangle \in T_j$ as *critical points*. The algorithm maintains the following two invariants:

(C1) The total number of critical points is at most $c_1 n$, for some constant $c_1 > 0$.

(C2) For every triangle $\triangle \in T_j$, $|L_\triangle| \leq n/c_2^j$ lines of L, where $c_2 \geq 2$ is a constant.

By (C1) and (C2), W_j contains $O(n)$ vertices of $\mathcal{A}(L)$ for $j > \log n$, so we can find λ^* by a sweep-line algorithm. We set W_1 to be a vertical strip containing all the vertices of $\mathcal{A}(L)$, and T_1 consists of a single unbounded triangle, namely W_1 itself. Suppose we are

[1] A $(1/r)$-*cutting* for a set H of n hyperplanes in $\mathbb{R}^d$ is a partition Ξ of $\mathbb{R}^d$ into $O(r^d)$ simplices with pairwise disjoint interiors so that the interior of each simplex intersects at most n/r hyperplanes of H. For any given r, a $1/r$-cutting of size $O(r^d)$ can be computed in time $O(nr^{d-1})$ [49].

in the beginning of the j-th stage. For every triangle $\triangle$ with $|L_\triangle| > n/c_2^j$, we compute a $(1/4c)$-cutting $\Xi_\triangle$ of $L_\triangle$, clip each triangle $\tau \in \Xi_\triangle$ within $\triangle$, re-triangulate $\tau \cap \triangle$, and compute the critical points for the new triangulation of W_j. If the total number of critical points after this refinement is at most $c_1 n$, we move to the next stage. Otherwise, we shrink the strip W_j as follows. We choose a critical point with the median x-coordinate, say $x = \lambda_m$, and, using the decision algorithm described in Section 2.2, determine whether λ^* is greater than, smaller than, or equal to λ_m. If $\lambda^* = \lambda_m$, then we stop, otherwise we shrink W_j to either $(l, \lambda_m) \times \mathbb{R}$ or $(\lambda_m, r) \times \mathbb{R}$, depending on whether λ^* is smaller or larger than λ_m. In any case, the new strip contains only half of the critical points. After repeating this procedure for a constant number of times, we can ensure that the number of critical points in the current strip is at most $c_1 n/4$. We set W_{j+1} to this strip, clip T_j' within W_{j+1}, re-triangulate every clipped triangle, and merge two triangles if the their union is a triangle intersecting at most n/c_2^j lines of L. The total number of critical points after these steps can be proved to be at most $c_1 n$. As shown in [51], each stage takes $O(n \log n)$ time, so the overall running time is $O(n \log^2 n)$. Using the same idea as in [67] (of counting the number of inversions approximately), Brönnimann and Chazelle [43] managed to improve the running time to $O(n \log n)$.

3.3 Matrix searching

An entirely different alternative to parametric searching was proposed by Frederickson and Johnson [105, 107, 108], which is based on searching in *sorted matrices*. It is applicable in cases where the set Λ of candidate critical values for the optimum parameter λ^* can be stored in an $n \times n$ matrix A, each of whose rows and columns is sorted. The size of the matrix is too large for an explicit binary search through its elements, so an implicit search is called for. Here we assume that each entry of the matrix A can be computed in $O(1)$ time. We give a brief sketch of this matrix-searching technique.

Let us assume that $n = 2^k$ for some $k \geq 0$. The algorithm works in two phases. The first phase, which consists of k steps, maintains a collection of disjoint submatrices of A so that λ^* is guaranteed to be an element of one of these submatrices. In the beginning of the i-th step, the algorithm has at most $B_i = 2^{i+2} - 1$ matrices, each of size $2^{k-i+1} \times 2^{k-i+1}$. The i-th step splits every such matrix into four square submatrices, each of size $2^{k-i} \times 2^{k-i}$, and discards some of these submatrices, so as to be left with only B_{i+1} matrices. After the k-th step, we are left with $O(n)$ singleton matrices, so we can perform a binary search on these $O(n)$ critical values to obtain λ^*.

The only nontrivial step in the above algorithm is determining which of the submatrices should be discarded in each step of the first phase, so that at most B_{i+1} matrices are left after the i-th step. After splitting each submatrix, we construct two sets U and V: U is the set of the smallest (i.e., upper-leftmost) element of each submatrix, and V is the set of largest (i.e., bottom-rightmost) element of each submatrix. We choose the median elements

λ_U and λ_V of U and V, respectively. We run the decision algorithm at λ_U and λ_V, to compare them with λ^*. If any of them is equal to λ^*, we are done. Otherwise, there are four cases to consider:

(i) If $\lambda_U < \lambda^*$, we discard all those matrices whose largest elements are smaller than λ_U.

(ii) If $\lambda_U > \lambda^*$, we discard all those matrices whose smallest elements are larger than λ_U; at least half of the matrices are discarded in this case.

(iii) If $\lambda_V < \lambda^*$, we discard all those matrices whose largest elements are smaller than λ_V; at least half of the matrices are discarded in this case.

(iv) If $\lambda_V > \lambda^*$, we discard all those matrices whose smallest elements are larger than λ_V.

It can be shown that this prune step retains at most B_{i+1} submatrices [105, 107, 108], as desired. In conclusion, one can find the optimum λ^* by executing only $O(\log n)$ calls to the decision procedure, so the running time of this matrix-searching technique is $O(\log n)$ times the cost of the decision procedure. The technique, when applicable, is both efficient and simple compared to the standard parametric searching.

Aggarwal et al. [22, 23, 24, 25] studied a different matrix-searching technique for optimization problems. They gave a linear-time algorithm for computing the minimum (or maximum) element of every row of a *totally monotone* matrix; a matrix $A = \{a_{i,j}\}$ is called totally monotone if $a_{i_1,j_1} < a_{i_1,j_2}$ implies that $a_{i_2,j_1} < a_{i_2,j_2}$, for any $1 \leq i_1 < i_2 \leq m, 1 \leq j_1 < j_2 \leq n$. Totally monotone matrices arise in many geometric, as well as nongeometric, optimization problems. For example, the farthest neighbors of all vertices of a convex polygon and the geodesic diameter of a simple polygon can be computed in linear time, using such matrices [23, 129].

4 Prune-and-Search Technique and Linear Programming

Like parametric searching, the *prune-and-search* (or *decimation*) technique also performs an implicit binary search over the finite set of candidate values for λ^*, but, while doing so, it also tries to eliminate input objects that are guaranteed not to affect the value of λ^*. Each phase of the technique eliminates a constant fraction of the remaining objects. Therefore, after a logarithmic number of steps, the problem size becomes a constant, and the problem can be solved in a final, brute-force step. Because of the 'decimation' of input objects, the overall cost of the resulting algorithm remains proportional to the cost of the first pruning phase. The prune-and-search technique was originally introduced by Megiddo [181, 183], in developing a linear-time algorithm for linear programming with n constraints in 2- and 3-dimensions. Later he extended the approach to obtain an $O(2^{2^d} n)$-time algorithm for linear programming in $\mathbb{R}^d$. Since then the prune-and-search technique has been applied to

many other geometric optimization problems. We illustrate the technique by describing Megiddo's two-dimensional linear-programming algorithm.

We are given a set $H = \{h_1, \ldots, h_n\}$ of n halfplanes and a vector c, and we wish to minimize cx over the *feasible region* $K = \bigcap_{i=1}^{n} h_i$. Without loss of generality, assume that $c = (0, 1)$ (i.e., we seek the lowest point of K). Let L denote the set of lines bounding the halfplanes of H, and let L^+ (resp. L^-) denote the subset of lines $\ell_i \in L$ whose associated halfplane h_i lies below (resp. above) ℓ_i. The algorithm pairs up the lines of L into disjoint pairs (ℓ_1, ℓ_2), (ℓ_3, ℓ_4), ..., so that either both the lines in a pair belong to L^+, or both belong to L^-. The algorithm computes the intersection points of the lines in each pair, and chooses the median, x_m, of their x-coordinates. Let x^* denote the x-coordinate of the optimal (i.e., lowest) point in K (if such a point exists). The algorithm then uses a linear-time decision procedure (whose details are omitted here, though some of them are discussed below), that determines whether $x_m = x^*$, $x_m < x^*$, or $x_m > x^*$. If $x_m = x^*$ we stop, since we have found the optimum. Suppose that $x_m < x^*$. If (ℓ, ℓ') is a pair of lines both of which belong to L^- and whose intersection point lies to the left of x_m, then we can discard the line with the smaller slope from any further consideration, because that line is known to pass below the optimal point of K. All other cases can be treated in a fully symmetric manner, so we have managed to discard about $n/4$ lines. We have thus computed, in $O(n)$ time, a subset $H' \subseteq H$ of about $3n/4$ constraints such that the optimal point of $K' = \bigcap_{h \in H'} h$ is the same as that of K. We now apply the whole procedure once again to H', and keep repeating this (for $O(\log n)$ stages) until either the number of remaining lines falls below some small constant, in which case we solve the problem by brute force (in constant time), or the algorithm has hit x^* 'accidentally', in which case it stops right away. (We omit here the description of the linear-time decision procedure, and of handling cases in which K is empty or unbounded; see [93, 181, 183] for details.) It is now easy to see that the overall running time of the algorithm is $O(n)$.

Remark. It is instructive to compare this method to the parametric searching technique, in the context of two-dimensional linear programming. In both approaches, the decision procedure aims to compare some given x_0 with the optimal value x^*. The way this is done is by computing the maximum and minimum values of the intercepts of the lines in L^- and in L^+, respectively, with the line $x = x_0$. A trivial method for computing those maximum and minimum in parallel is in a binary-tree manner, computing the maximum or minimum of pairs of lines, then of pairs of pairs, and so on. Both techniques begin by implementing generically the first parallel step of this decision procedure. The improved performance of the prune-and-search algorithm stems from the realization that (a) there is no need to perform the full binary search over the critical values of the comparisons in that stage—a single binary search step suffices (this is similar to Cole's enhancement of parametric searching, mentioned above), and (b) this single comparison allows us to discard a quarter of the given lines, so there is no point in continuing the generic simulation, and it is better to start the whole algorithm from scratch, with the surviving lines. From this

point of view, the prune-and-search technique can be regarded as an optimized variant of parametric searching.

This technique can be extended to higher dimensions, although it becomes more complicated, and requires recursive invocations of the algorithm on subproblems in lower dimensions. It yields a deterministic algorithm for linear programming that runs in $O(C_d n)$ time, where C_d is a constant depending on d. One of the difficult steps in higher dimensions is to develop a procedure that can discard a fraction of the input constraints from further consideration by invoking the $(d-1)$-dimensional linear-programming algorithm a constant number of times; the value of C_d depends on this constant. The original approach by Megiddo [183] gives $C_d = 2^{2^d}$, which was improved by Clarkson [59] and Dyer [86] to 3^{d^2}. Their procedure can be simplified and improved using geometric cuttings as follows (see [17, 90]). Let H be the set of hyperplanes bounding the input constraints. Choose r to be a constant, and compute a $1/r$-cutting Ξ. By invoking the $(d-1)$-dimensional linear-programming algorithm recursively $O(r^d)$ times (at most three times for each hyperplane h supporting a facet of a simplex in Ξ: on h itself, and on two parallel hyperplanes parallel to h, one on each side and lying very close to h), one can determine the simplex Δ of Ξ that contains x^*. The constraints whose bounding hyperplanes do not intersect Δ can be discarded because they do not determine the optimum value. We solve the problem recursively with the remaining n/r constraints. Dyer and Frieze [90] (see also Agarwal et al. [17]) have shown that the number of calls to the recursive algorithm can be reduced to $O(dr)$. This yields an $d^{O(d)}n$-time algorithm for linear programming in $\mathbb{R}^d$. An entirely different algorithm with a similar running time was given by Chazelle and Matoušek [65]. It is an open problem whether faster deterministic algorithms can be developed. Although no progress has been made on this front, there have been significant developments on randomized algorithms for linear programming, which we will discuss in the next two sections.

Recently there has been a considerable interest in parallelizing Megiddo's prune-and-search algorithm. Deng [76] gave an $O(\log n)$-time and $O(n)$-work algorithm for two-dimensional linear programming, under the CRCW model of computation. His algorithm, however, does not extend to higher dimensions. Alon and Megiddo [28] gave a randomized algorithm under the CRCW model of computation that runs, with high probability, in $O(1)$ time using $O(n)$ processors. Ajtai and Megiddo [27] gave an $O((\log\log n)^d)$-time deterministic algorithm using $O(n)$ processors under Valiant's model of computation. Goodrich [117] and Sen [213] gave an $O((\log\log n)^{d+2})$-time, $O(n)$-work algorithm under the CRCW model; see also [88].

5 Randomized Algorithms for Linear Programming

Random sampling has become one of the most powerful and versatile techniques in computational geometry, so it is no surprise that this technique has also been successful in solving

many geometric optimization problems. See the book by Mulmuley [194] and the survey papers by Clarkson [60] and Seidel [212] for applications of the random-sampling technique in computational geometry. In this section, we describe a randomized algorithm for linear programming by Clarkson [62], based on random sampling, which is actually quite general and can be applied to any geometric set-cover and related problems [6, 45]. Other randomized algorithms for linear-programming, which run in expected linear time for any fixed dimension, are proposed by Dyer and Frieze [90], Seidel [211], Kalai [149], and Matoušek et al. [178].

Let H be the set of constraints. We assign a weight $\mu(h) \in \mathbb{Z}$ to each constraint; initially $\mu(h) = 1$ for all $h \in H$. For a subset $A \subseteq H$, let $\mu(A) = \sum_{h \in A} \mu(h)$. The algorithm works in rounds, each of which consists of the following steps. Set $r = 6d^2$. If $|H| \leq 6d^2$, we compute the optimal solution using the simplex algorithm. Otherwise, choose a random sample $R \subset H$ such that $\mu(R) = r$. (We can regard H as a multiset in which each constraint appears $\mu(h)$ times, and we choose a multiset $R \in \binom{H}{r}$ of r constraints.) We compute the optimal solution x_R for R and the subset $V \subset H \setminus R$ of constraints that x_R *violates* (that is, the subset of constraints that do not contain x_R). If $V = \emptyset$, the algorithm returns x_R. If $\mu(V) \leq 3\mu(H)/d$, we double the weight of each constraint in V; in any case, we repeat the sampling procedure. See Figure 1 for a pseudocode of the algorithm.

```
function procedure RANDOM_lp (H)        /* H: n constraints in R^d
    if n ≤ 6d^2 then                      /* μ_h = 1 for all h ∈ H
        return SIMPLEX(H)                       /* returns v(H)
    else
        r = 6d^2
        repeat
            choose random R ∈ (H choose r)
            x_R := SIMPLEX(R)
            V := {h ∈ H | x_R violates h}
            if μ(V) ≤ μ(H)/3d then
                for all h ∈ V do μ_h := 2μ_h
        until V = ∅
        return x_R
```

Figure 1: Clarkson's randomized LP algorithm

Let B be the set of d constraints whose boundaries are incident to the optimal solution. A round is called *successful* if $\mu(V) \leq 3\mu(H)/d$. Using the fact that R is a random subset, one can argue that each round is successful with probability at least $1/2$. Every successful round increases $\mu(H)$ by a factor of at most $(1 + 1/3d)$, so the total weight $\mu(H)$ after kd successful rounds is at most $n(1 + 1/3d)^{kd} < ne^{k/3}$. On the other hand, each successful iteration doubles the weight of at least one constraint in B (it is easily verified that V must

contain such a constraint), which implies that after kd iterations $\mu(H) \geq \mu(B) \geq 2^k$. Hence, after kd successful rounds, $2^k \leq \mu(H) \leq ne^{k/3}$. This implies that the above algorithm terminates in at most $3d \ln n$ successful rounds. Since each round takes $O(d^d)$ time to compute x_R and $O(dn)$ time to compute V, the expected running time of the algorithm is $O((d^2n + d^{d+1}) \log n)$. By combining this algorithm with a randomized recursive algorithm, Clarkson improved the expected running time to $O(d^2n) + (d)^{d/2+O(1)} \log n$.

6 Abstract Linear Programming

In this section we present an abstract framework that captures linear programming, as well as many other geometric optimization problems, including computing smallest enclosing balls (or ellipsoids) of finite point sets in $\mathbb{R}^d$, computing largest balls (ellipsoids) circumscribed in convex polytopes in $\mathbb{R}^d$, computing the distance between polytopes in d-space, general convex programming, and many other problems. Sharir and Welzl [218] and Matoušek et al. [178] (see also Kalai [149]) presented a randomized algorithm for optimization problems in this framework, whose expected running time is linear in terms of the number of constraints whenever the combinatorial dimension d (whose precise definition, in this abstract framework, will be given below) is fixed. More importantly, the running time is 'subexponential' in d for many of the LP-type problems, including linear programming. This is the first subexponential 'combinatorial' bound for linear programming (a bound that counts the number of arithmetic operations and is independent of the bit complexity of the input), and is a first step toward the major open problem of obtaining a strongly polynomial algorithm for linear programming. The papers by Gärtner and Welzl [110] and by Goldwasser [112] also survey the known results on LP-type problems.

6.1 An abstract framework

Let us consider optimization problems specified by a pair (H, w), where H is a finite set, and $w : 2^H \to \mathcal{W}$ is a function into a linearly ordered set $(\mathcal{W}, \leq)$; we assume that $\mathcal{W}$ has a minimum value $-\infty$. The elements of H are called *constraints*, and for $G \subseteq H$, $w(G)$ is called the *value of* G. Intuitively, $w(G)$ denotes the smallest value attainable by a certain objective function while satisfying all the constraints of G. The goal is to compute a minimal subset B_H of H with $w(B_H) = w(H)$ (from which, in general, the value of H is easy to determine), assuming the availability of three basic operations, which we specify below.

Such a minimization problem is called *LP-type* if the following two axioms are satisfied:

Axiom 1. (*Monotonicity*) For any F, G with $F \subseteq G \subseteq H$, we have
$$w(F) \leq w(G).$$

Axiom 2. (*Locality*) For any $F \subseteq G \subseteq H$ with $-\infty < w(F) = w(G)$ and any $h \in H$,

$$w(G) < w(G \cup \{h\}) \Rightarrow w(F) < w(F \cup \{h\}).$$

Linear programming is easily shown to be an LP-type problem, if we set $w(G)$ to be the vertex of the feasible region that minimizes the objective function and that is coordinate-wise lexicographically smallest (this definition is important to satisfy Axiom 2), and if we extend the definition of $w(G)$ in an appropriate manner to handle empty or unbounded feasible regions.

A *basis* $B \subseteq H$ is a set of constraints satisfying $-\infty < w(B)$, and $w(B') < w(B)$ for all proper subsets B' of B. For $G \subseteq H$, with $-\infty < w(G)$, a *basis of* G is a minimal subset B of G with $w(B) = w(G)$. (For linear programming, a basis of G is a minimal set of halfspace constraints in G such that the minimal vertex of their intersection is the minimal vertex of G.) A constraint h is *violated by* G if $w(G) < w(G \cup \{h\})$, and it is *extreme in* G if $w(G - \{h\}) < w(G)$. The *combinatorial dimension of* (H, w), denoted as $\dim(H, w)$, is the maximum cardinality of any basis. We call an LP-type problem *basis regular* if for any basis with $|B| = \dim(H, w)$ and for any constraint h, every basis of $B \cup \{h\}$ has exactly $\dim(H, w)$ elements. (Clearly, linear programming is basis-regular, where the dimension of every basis is d.)

We assume that the following primitive operations are available.

(*Violation test*) h `is violated by` B: for a constraint h and a basis B, tests whether h is violated by B.

(*Basis computation*) `basis`(B, h): for a constraint h and a basis B, computes a basis of $B \cup \{h\}$.

(*Initial basis*) `initial`(H): An initial basis B_0 with exactly $\dim(H, w)$ elements is available.

For linear programming, the first operation can be performed in $O(d)$ time, by substituting the coordinates of the vertex $w(B)$ into the equation of the hyperplane defining h. The second operation can be regarded as a dual version of the pivot step in the simplex algorithm, and can be implemented in $O(d^2)$ time. The third operation is also easy to implement.

We are now in position to describe the algorithm. Using the initial-basis primitive, we compute a basis B_0 and call `SUBEX_lp`(H, B_0), where `SUBEX_lp` is the recursive algorithm, given in Figure 2, for computing a basis B_H of H.

A simple inductive argument shows the expected number of primitive operations performed by the algorithm is $O(2^\delta n)$, where $n = |H|$ and $\delta = \dim(H, w)$ is the combinatorial dimension. However, using a more involved analysis, which can be found in [178], one can show that basis-regular LP-type problems can be solved with an expected number of

```
function procedure SUBEX_lp(H, C);      /* H: set of n constraints in R^d;
    if H = C then                                    /* C ⊆ H: a basis;
        return C                                   /* returns a basis of H.
    else
        choose a random h ∈ H \ C;
        B := SUBEX_lp(H \ {h}, C);
        if h is violated by B then
            return SUBEX_lp(H, basis(B, h))
        else
            return B;
```

Figure 2: A randomized algorithm for LP-type problems.

at most $e^{2\sqrt{\delta \ln((n-\delta)/\sqrt{\delta})+O(\sqrt{\delta}+\ln n)}}$ violation tests and basis computations. This is the 'subexponential' bound that we alluded to.

Matoušek [173] has given examples of abstract LP-type problems of combinatorial dimension d with $2d$ constraints, for which the above algorithm requires $\Omega(e^{\sqrt{2d}}/\sqrt[4]{d})$ primitive operations. Here is an example of such a problem. Let A be a lower-triangular $d \times d$ $\{0,1\}$-matrix, with all diagonal entries being 0, i.e., $a_{i,j} \in \{0,1\}$ for $1 \le j < i \le d$ and $a_{ij} = 0$ for $1 \le i \le j \le d$. Let $x_1, \ldots, x_d$ denote variables over $\mathbb{Z}_2$, and suppose that all additions and multiplications are performed modulo 2. We define a set of $2d$ constraints $H(A) = \{h_i^c \mid 1 \le i \le d,\ c \in \{0,1\}\}$, where

$$h_i^c :\ x_i \ge \sum_{j=1}^{i-1} a_{ij} x_j + c\,.$$

That is, $x_i = 1$ if the right-hand side of the constraint is 1 modulo 2 and $x_i \in \{0,1\}$ if the right-hand side is 0 modulo 2. For a subset $G \subseteq H$, we define $w(G)$ to be the lexicographically smallest point of $\bigcap_{h \in G} h$. It can be shown that the above example is an instance of a basis-regular LP-type problem, with combinatorial dimension d. Matoušek showed that if A is chosen randomly (i.e., each entry a_{ij}, for $1 \le j < i \le d$, is chosen independently, with $\Pr[a_{ij} = 0] = \Pr[a_{ij} = 1] = 1/2$) and the initial basis is also chosen randomly, then the expected number of primitive operations performed by `SUBEX_lp` is $\Omega(e^{\sqrt{2d}}/\sqrt[4]{d})$.

6.2 Linear programming

We are given a set H of n halfspaces in $\mathbb{R}^d$. We assume that the objective vector is $c = (1, 0, 0, \ldots, 0)$, and the goal is to minimize cx over all points in the common intersection

$\bigcap_{h \in H} h$. For a subset $G \subseteq H$, define $w(G)$ to be the lexicographically smallest point (vertex) of the intersection of halfspaces in G. (As noted, some care is needed to handle unbounded or empty feasible regions; we omit here details concerning this issue.)

As noted above, linear programming is a basis-regular LP-type problem, with combinatorial dimension d, and each violation test or basis computation can be implemented in time $O(d)$ or $O(d^2)$, respectively. In summary, we obtain a randomized algorithm for linear programming, which performs $e^{2\sqrt{d \ln(n/\sqrt{d})} + O(\sqrt{d} + \ln n)}$ expected number of arithmetic operations. Using `SUBEX_lp` instead of the simplex algorithm for solving the small-size problems in the `RANDOM_lp` algorithm (given in Figure 1), the expected number of arithmetic operations can be reduced to $O(d^2 n) + e^{O(\sqrt{d \log d})}$.

In view of Matoušek's lower bound, one should aim to exploit additional properties of linear programming to obtain a better bound on the performance of the algorithm for linear programming; this is still a major open problem.

6.3 Extensions

Recently, Chazelle and Matoušek [54] gave a deterministic algorithm for solving LP-type problems in time $O(\delta^{O(\delta)} n)$, provided an additional axiom holds (together with an additional computational assumption). Still, these extra requirements are satisfied in many natural LP-type problems. Matoušek [175] has investigated the problem of finding the best solution, for an abstract LP-type problem, that satisfies all but k of the given constraints. He proved that the number of bases that violate at most k constraints in a non-degenerate instance of an LP-type problem is $O((k+1)^\delta)$, where δ is the combinatorial dimension of the problem, and that they can be computed in time $O(n(k+1)^\delta)$. In some cases the running time can be improved using appropriate data structures; see [175] for details.

Amenta [32] considers the following extension of the abstract framework: Suppose we are given a family of LP-type problems (H, w_λ), monotonically parameterized by a real parameter λ; the underlying ordered value set $\mathcal{W}$ has a maximum element $+\infty$ representing *infeasibility*. The goal is to find the smallest λ for which (H, w_λ) is feasible, i.e. $w_\lambda(H) < +\infty$. See [32, 33] for more details and related work.

6.4 Abstract linear programming and Helly-type theorems

In this subsection we describe an interesting connection between Helly-type theorems and LP-type problems, as originally noted by Amenta [32].

Let $\mathbf{K}$ be an infinite collection of sets in $\mathbb{R}^d$, and let t be an integer. We say that $\mathbf{K}$ satisfies a *Helly-type* theorem, with *Helly number* t, if the following holds: If $\mathcal{K}$ is a finite subcollection of $\mathbf{K}$ with the property that every subcollection of t elements of $\mathcal{K}$ has a

nonempty intersection, then $\bigcap \mathcal{K} \neq \emptyset$. (The best known example of a Helly-type theorem is Helly's theorem itself [123], which applies for the collection $\mathbf{K}$ of all convex sets in $\mathbb{R}^d$, with the Helly number $d+1$; see [70] for an excellent survey on this topic.) Suppose further that we are given a collection $\mathcal{K}(\lambda)$, consisting of n sets $K_1(\lambda), \ldots, K_n(\lambda)$ that are parametrized by some real parameter λ, with the property that $K_i(\lambda) \subseteq K_i(\lambda')$, for $i = 1, \ldots, n$ and for $\lambda \leq \lambda'$, and that, for any fixed λ, the family $\{K_1(\lambda), \ldots, K_n(\lambda)\}$ admits a Helly-type theorem, with a fixed Helly number t. Our goal is to compute the smallest λ for which $\bigcap_{i=1}^{n} K_i(\lambda) \neq \emptyset$, assuming that such a minimum exists. Amenta proved that this problem can be transformed to an LP-type problem, whose combinatorial dimension is at most t.

As an illustration, consider the smallest-enclosing-ball problem. Let $P = \{p_1, \ldots, p_n\}$ be the given set of n points in $\mathbb{R}^d$, and let $K_i(\lambda)$ be the ball of radius λ centered at p_i, for $i = 1, \ldots, n$. Since the K_i's are convex, the collection in question has Helly number $d+1$. It is easily seen that the minimal λ for which the $K_i(\lambda)$'s have nonempty intersection is the radius of the smallest enclosing ball of P. This shows that the smallest-enclosing-ball problem s LP-type, and can thus be solved in $O(n)$ randomized expected time in any fixed dimension. See below for more details.

There are several other examples where Helly-type theorems can be turned into LP-type problems. They include (i) computing a line transversal to a family of translates of a convex object in the plane, (ii) computing a smallest homothet of a given convex set that intersects (or contains, or is contained in) every member in a given collection of n convex sets in $\mathbb{R}^d$, and (iii) computing a line transversal to certain families of convex objects in 3-space. We refer the reader to [32, 33] for more details and for additional examples.

PART II: APPLICATIONS

In the first part of the paper we focused on general techniques for solving geometric optimization problems. In this second part, we list numerous problems in geometric optimization that can be attacked using some of the techniques reviewed above. For the sake of completeness, we will also review variants of these problems for which the above techniques are not applicable.

7 Facility-Location Problems

A typical *facility-location* problem is defined as follows: Given a set $D = \{d_1, \ldots, d_n\}$ of n *demand points* in $\mathbb{R}^d$, a parameter p, and a distance function δ, we wish to find a set S of p *supply objects* (points, lines, segments, etc.) so that the maximum distance between a

demand point and its nearest supply object is minimized. That is, we minimize, over all possible appropriate sets S, the following objective function

$$c(D, S) = \max_{1 \le i \le n} \min_{s \in S} \delta(d_i, s).$$

Instead of minimizing the above quantity, one can choose other objective functions, such as

$$c'(D, S) \quad = \quad \sum_{i=1}^{n} \min_{s \in S} \delta(d_i, s).$$

In some applications, a weight w_i is assigned to each point $d_i \in D$, and the distance from d_i to a point $x \in \mathbb{R}^2$ is defined as $w_i \delta(d_i, x)$. The book by Drezner [83] describes many other variants of the facility-location problem.

The set $S = \{s_1, \ldots, s_p\}$ of supply objects partitions D into p *clusters*, $D_1, \ldots, D_p$, so that s_i is the nearest supply object to all points in D_i. Therefore a facility-location problem can also be regarded as a clustering problem. These facility-location (or clustering) problems arise in many areas, including operations research, pattern matching, data compression, and data mining. A useful extension of the facility-location problem, which has been widely studied, is the *capacitated facility-location* problem, in which we have an additional constraint that the size of each cluster should be at most c for some parameter $c \ge n/p$. If p is considered as part of the input, most facility-location problems are NP-hard, even in the plane or even when only an approximate solution is being sought [101, 113, 159, 186, 187, 167]. Although many of these problems can be solved in polynomial time for a fixed value of p, some of them still remain intractable. In this section we review efficient algorithms for a few specific facility-location problems, to which the techniques introduced in Part I can be applied; in these applications, p is usually a small constant.

7.1 Euclidean p-center

Given a set D of n demand points in $\mathbb{R}^d$, we wish to find a set S of p supply *points* so that the maximum Euclidean distance between a demand point and its nearest neighbor in S is minimized. This problem can be solved efficiently, when p is small, using the parametric searching technique. The decision problem in this case is to determine, for a given radius r, whether D can be covered by the union of p balls of radius r. In some applications, S is required to be a subset of D, in which case the problem is referred to as the *discrete p-center* problem.

General results. A naive procedure for the p-center problem runs in time $O(n^{dp+2})$, observing that the critical radius r^* is determined by at most $d+1$ points, which also determine one of the balls; similarly, there are $O(n^{d(p-1)})$ choices for the other $p-1$ balls,

and it takes $O(n)$ time to verify whether a specific choice of balls covers D. For the planar case, Drezner [79] gave an improved $O(n^{2p+1})$-time algorithm, which was subsequently improved by Hwang et al. [142] to $n^{O(\sqrt{p})}$. Hwang et al. [141] have given another $n^{O(\sqrt{p})}$-time algorithm for computing a discrete p-center. Therefore, for a fixed value of p, the Euclidean p-center (and also the Euclidean discrete p-center) problem can be solved in polynomial time in any fixed dimension. However, either of these problems is NP-complete for $d \geq 2$, if p is part of the input [104, 187]. This has led researchers to develop efficient algorithms for approximate solutions and for small values of p and d.

Approximation algorithms. Let r^* be the minimum value of r for which p disks of radius r cover D. The greedy algorithm described in Figure 3, originally proposed by Gonzalez [113] and by Hochbaum and Shmoys [132, 133], computes in $O(np)$ time a set S of p points so that $c(D, S) \leq 2r^*$.

```
function procedure GREEDY_COVER (D, p);      /* D: set of n points in R^d;
    for i = 1 to n do
        MAX_DIST(i) = ∞;
    for i = 1 to p do
        s_i = d_j  s.t.  MAX_DIST(j) = max_{1≤l≤n} MAX_DIST(l);
        for j = 1 to n do
            MAX_DIST(j) = min{MAX_DIST(j), δ(s_i, d_j)};
    return {s_1, ..., s_p};
```

Figure 3: Greedy algorithm for approximate p-center.

This algorithm works equally well for any metric and for the weighted case [89]. Note that it also provides an approximate solution to the discrete p-center problem. The running time was improved to $O(n \log p)$ by Feder and Green [101]. They also showed that computing a set S of p supply points such that $c(D, S) \leq 1.822r^*$ under the Euclidean distance function, or $c(D, S) < 2r^*$ under the L_∞-metric, is NP-Hard. See [114, 159] for other approximation algorithms.

Another way of seeking an approximation is to find a small number of balls of a fixed radius, say r, that cover all demand points. Computing k^*, the minimum number of balls of radius r that cover D, is also NP-complete [104]. A greedy algorithm can construct $k^* \log n$ balls of radius r that cover D. Hochbaum and Maass gave a polynomial-time algorithm to compute a cover of size $(1 + \varepsilon)k^*$, for any $\varepsilon > 0$ [130]; see also [45, 101, 114]. No constant-factor approximation algorithm is known for the capacitated covering problem, with unit-radius disks, that is, the problem of partitioning a given point set S in the plane into the minimum number of clusters, each of which consists of at most c points and can be covered by a disk of radius r. Nevertheless, the greedy algorithm can be modified to obtain

an $O(\log n)$-factor approximation for this problem [36].

The general results reviewed so far do not make use of parametric searching: since there are only $O(n^{d+1})$ candidate values for the optimum radius r^*, one can simply enumerate all these values and run a standard binary search among them. The improvement that one can gain from parametric searching is significant only when p is relatively small, which is what we are going to discuss next.

Euclidean 1-center. The 1-center problem is to compute the smallest ball enclosing D. The decision procedure for the 1-center problem is thus to determine whether D can be covered by a ball of radius r. For $d = 2$, the decision problem can be solved in $O(\log n)$ parallel steps using $O(n)$ processors, e.g., be testing whether the intersection of the disks of radius r centered at the points of D is nonempty. This yields an $O(n \log^3 n)$-time algorithm for the planar Euclidean 1-center problem. Using the prune-and-search paradigm, one can, however, solve the 1-center problem in linear time [86], and this approach extends to higher dimensions, where, for any fixed d, the running time is $d^{O(d)}n$ [17, 54, 90]. Megiddo [185, 189] extends this approach to obtain a linear-time algorithm for the weighted 1-center problem. Dynamic data structures for maintaining the smallest enclosing ball of a set of points, as points are being inserted and deleted, are given in [11, 37]. See [78, 82, 84, 102, 182] for other variants of the 1-center problem. A natural extension of the 1-center problem is to find a disk of the smallest radius that contains k of the n input points. The best known deterministic algorithm runs in time $O(n \log n + nk \log k)$ using $O(n + k^2 \log k)$ space [100, 72] (see also [96]), and the best known randomized algorithm runs in $O(n \log n + nk)$ expected time using $O(nk)$ space, or in $O(n \log n + nk \log k)$ expected time using $O(n)$ space [174]. Matoušek [175] also showed that the smallest disk covering all but k points can be computed in time[2] $O(n \log n + k^3 n^{\varepsilon})$.

The smallest-enclosing-ball problem is an LP-type problem, with combinatorial dimension $d + 1$ [218, 232]. Indeed, the constraints are the given points, and the function w maps each subset G to the radius of the smallest ball containing G. Monotonicity of w is trivial, and locality follows easily from the uniqueness of the smallest enclosing ball of a given set of points in general position. The combinatorial dimension is $d + 1$ because at most $d + 1$ points are needed to determine the smallest enclosing ball. This problem is, however, not basis-regular (the smallest enclosing ball may be determined by any number, between 2 and $d + 1$, of points), and a naive implementation of the basis-changing operation may be quite costly (in d). Nevertheless, Gärtner [109] showed that this operation can be performed in this case using expected $e^{O(\sqrt{d})}$ arithmetic operations. Hence, the expected running time of the algorithm is $O(d^2 n) + e^{O(\sqrt{d \log d})}$.

[2] In this paper, the meaning of complexity bounds that depend on an arbitrary parameter $\varepsilon > 0$, like the one stated here, is that given any $\varepsilon > 0$, we can fine-tune the algorithm so that its complexity satisfies the stated bound. In these bounds the constant of proportionality usually depends on ε, and tends to infinity when ε tends to zero.

There are several extensions of the smallest-enclosing-ball problem. They include: (i) computing the smallest enclosing ellipsoid of a point set [54, 87, 201, 232], (ii) computing the largest ellipsoid (or ball) inscribed inside a convex polytope in $\mathbb{R}^d$ [109], (iii) computing a smallest ball that intersects (or contains) a given set of convex objects in $\mathbb{R}^d$ (see [185], and (iv) computing a smallest area annulus containing a given planar point set. All these problems are known to be LP-type, and thus can be solved using the algorithm described in Section 6. However, not all of them run in subexponential expected time because they are not basis regular. Linear-time algorithms, based on prune-and-search technique, have also been developed for many of these problems in two dimensions [40, 41, 42, 145].

Euclidean 2-center. In this problem we want to cover a set D of n points in $\mathbb{R}^d$ by two balls of smallest possible common radius. There is a trivial $O(n^{d+1})$-time algorithm for the 2-center problem in $\mathbb{R}^d$, because the 'clusters' D_1 and D_2 in an optimal solution can be separated by a hyperplane [80]. Faster algorithms have been developed for the planar case using parametric searching. Agarwal and Sharir [13] gave an $O(n^2 \log n)$-time algorithm for determining whether D can be covered by two disks of radius r. Their algorithm proceeds as follows: There are $O(n^2)$ distinct subsets of D that can be covered by a disk of radius r, and these subsets can be computed in $O(n^2 \log n)$ time, by processing the arrangement of the n disks of radius r, centered at the points of D. For each such subset D_1, the algorithm checks whether $D \setminus D_1$ can be covered by another disk of radius r. Using a dynamic data structure, the total time spent is shown to be $O(n^2 \log n)$. Plugging this algorithm into the parametric searching machinery, one obtains an $O(n^2 \log^3 n)$-time algorithm for the Euclidean 2-center problem. Matoušek [170] gave a simpler randomized $O(n^2 \log^2 n)$ expected-time algorithm by replacing parametric searching with randomization. The running time of the decision algorithm was improved by Hershberger [126] to $O(n^2)$, which has been utilized in the best near-quadratic solution, by Jaromczyk and Kowaluk [146], which runs in $O(n^2 \log n)$ time; see also [147].

A major progress on this problem was recently made by Sharir [215], who gave an $O(n \log^9 n)$-time algorithm, by combining the parametric searching technique with several additional techniques, including a variant of the matrix searching algorithm of Frederickson and Johnson [108]. Eppstein [99] has simplified Sharir's algorithm, using randomization and better data structures, and obtained an improved solution, whose expected running time is $O(n \log^2 n)$.

Recently Agarwal et al. [19] have developed an $O(n^{4/3} \log^5 n)$-time algorithm for the discrete 2-center problem.

Rectilinear p-center. In this problem the metric is the L_∞-distance, so the decision problem is now to cover the given set D by a set of p axis-parallel cubes, each of length $2r$. The problem is NP-Hard if p is part of the input and $d \geq 2$, or if d is part of the input and

$p \geq 3$ [104, 186]. Ko et al. [159] showed that computing an S with $c(D, S) < 2r^*$ is also NP-Hard.

The rectilinear 1-center problem is trivially solved in linear time, and a polynomial-time algorithm for the rectilinear 2-center problem, even if d is unbounded, is given in [186]. A linear-time algorithm for the planar rectilinear 2-center problem is given by Drezner [81] (see also [157]); Ko and Lee [158] gave an $O(n \log n)$-time algorithm for the weighted case. Recently, Sharir and Welzl [219] have developed a linear-time algorithm for the rectilinear 3-center problem, by showing that it is an LP-type problem (as is the rectilinear 2-center problem). They have also obtained an $O(n \log n)$-time algorithm for computing a rectilinear 4-center (and have shown that this algorithm is worst-case optimal), and an $O(n \log^5 n)$-time algorithm for computing a rectilinear 5-center. The algorithms for the 4-center and 5-center employ the Frederickson-Johnson matrix searching technique. See [152, 219] for additional related results.

7.2 Euclidean p-line-center

Let D be a set of n points in $\mathbb{R}^d$ and δ be the Euclidean distance function. We wish to compute the smallest real value w^* so that D can be covered by the union of p strips of width w^*. Megiddo and Tamir showed that the problem of determining whether $w^* = 0$ (i.e, D can be covered by p lines) is NP-Hard [188], which not only proves that the p-line-center is NP-Complete, but also proves that approximating w^* within a constant factor is NP-Complete. Approximation algorithms for this problem are given in [121].

The 1-line center is the classical *width* problem. For $d = 2$, an $O(n \log n)$-time algorithm was given by Houle and Toussaint [138]. A matching lower bound was proved by Lee and Wu [163]. They also gave an $O(n^2 \log n)$-time algorithm for the weighted case, which was improved to $O(n \log n)$ in [137].

For the 2-line-center problem in the plane, Agarwal and Sharir [13] (see also [12]) gave an $O(n^2 \log^5 n)$-time algorithm, using parametric searching. This algorithm is very similar to their 2-center algorithm, i.e., the decision algorithm finds all subsets of S that can be covered by a strip of width w and for each such subset S_1, it determines whether $S \setminus S_1$ can be covered by another strip of width w. The heart of this decision procedure is an efficient algorithm for the following off-line width problem: given a sequence $\Sigma = (\sigma_1, \ldots, \sigma_n)$ of insertions and deletions of points in a set D and a real number w, is there an i such that after performing the first i updates, the width of the current point set is at most w? A solution to this off-line width problem, that runs in $O(n^2 \log^3 n)$ time, is given in [12]. The running time for the optimization problem was improved to $O(n^2 \log^4 n)$ by Katz and Sharir [154] and by Glozman et al. [111], using expander graphs and the Frederickson-Johnson matrix searching technique, respectively. The best known algorithm, by Jaromczyk and Kowaluk [148], runs in $O(n^2 \log^2 n)$ time. It is an open problem whether a near-linear (or just subquadratic)

time algorithm exists for computing a 2-line center.

7.3 Euclidean p-median

Let D be a set of n points in $\mathbb{R}^d$. We wish to compute a set S of p supply points so that the sum of distances from each demand point to its nearest supply point is minimized (i.e., we want to minimize the objective function $c'(D, S)$). This problem can be solved in polynomial time for $d = 1$ (for $d = 1$ and $p = 1$ the solution is the median of the given points, whence the problem derives its name), and it is NP-Hard for $d \geq 2$ [187]. The special case of $d = 2, p = 1$ is the classical *Fermant-Weber problem*, and it goes back to the 17th century. It is known that the solution for the Fermant-Weber problem is unique and algebraic. Several numerical approaches have been proposed to compute an approximate solution. See [48, 233] for the history of the problem and for the known algorithms, and [197] for some heuristics for the p-median problem that work well for a set of random points.

7.4 Segment-center

Given a segment e, we wish to find a translated and rotated copy of e so that the maximum distance from each point of the given set D of demand points to this copy is minimized. This problem was originally considered by Imai et al. [143], who had given an $O(n^4 \log n)$-time algorithm. An improved solution, based on parametric searching, with $O(n^2 \alpha(n) \log^3 n)$ running time, was later obtained in [8] (here $\alpha(n)$ denotes the extremely slowly growing inverse of Ackermann's function). The decision problem in this case is to determine whether there exists a translated and rotated copy of the 'hippodrome' $H = e \oplus B_r$, the Minkowski sum of the segment e with a disk of radius r, which fully contains D. Since H is convex, this is equivalent to H containing $P = \text{conv}(D)$. Hence the decision procedure is actually: Given a convex polygon P and the hippodrome H, does H contain a translated and rotated copy of P? see Figure 4. Note that placements of P can be specified in terms of three parameters, two for the translation and one for the rotation. Let $FP \subseteq \mathbb{R}^3$ denote the set of placements of P at which P lies inside H. Using Davenport–Schinzel sequences [216], Agarwal and Sharir showed that the complexity of FP is $O(n^2 2^{\alpha(n)})$, and that it can be computed in time $O(n^2 2^{\alpha(n)} \log n)$. By exploiting various geometric and combinatorial properties of FP and using some elegant results from combinatorial geometry, Efrat and Sharir [95] showed that the complexity of FP is only $O(n \log n)$, and that one can determine in time $O(n^{1+\varepsilon})$ whether $FP \neq \emptyset$. Plugging this into the parametric searching technique, one obtains an $O(n^{1+\varepsilon})$-time solution to the segment-center problem.

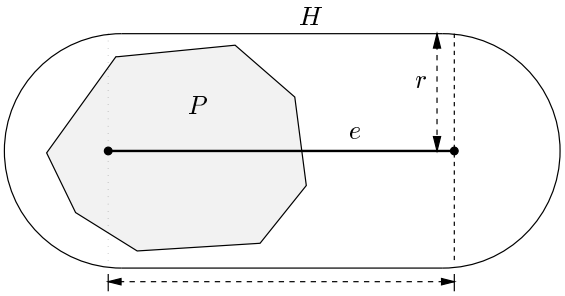

Figure 4: The segment-center problem

7.5 Other facility-location problems

Besides the problems discussed above, several other variants of the facility-location problem have been studied. For example, Hershberger [125] described an $O(n^2/\log\log n)$-time algorithm for partitioning a given set S of n points into two subsets so that the sum of their diameters is minimized. If we want to minimize the maximum of the two diameters, the running time can be improved to $O(n\log n)$ [127]. Glozman et al. [111] have studied problems of covering S by several different kinds of shapes. Maass [167] showed that the problem of covering S with the minimum number of unit-width annuli is NP-Hard even for $d = 1$ (a unit-width annulus in 1-dimension is a union of two unit-length intervals), and Hochbaum and Maass [131] gave an approximation algorithm for covering points with annuli.

8 Proximity Problems

8.1 Diameter in 3-space

Given a set S of n points in $\mathbb{R}^3$, we wish to compute the *diameter* of S, that is, the maximum distance between any two points of S. The decision procedure here is to determine, for a given radius r, whether the intersection of the balls of radius r centered at the points of S contains S. The intersection of congruent balls in $\mathbb{R}^3$ has linear complexity [118, 124], therefore it is natural to ask whether the intersection of n congruent balls can be computed in $O(n\log n)$ time. (Checking whether all points of S lie in the intersection can then be performed in additional $O(n\log n)$ time, using straightforward point-location techniques.) Clarkson and Shor [64] gave a very simple $O(n\log n)$ expected-time randomized algorithm (which is worst-case optimal) for computing the intersection, and then used a randomized prune-and-search algorithm, summarized in Figure 5, to compute the diameter of S.

```
function procedure DIAMETER (S);
    choose a random point p ∈ S;
    q = a farthest neighbor of p;
    compute I = ⋂_{p'∈S} B(p', δ(p,q))
    S_1 = S \ I
    if S_1 = ∅
        then return d(p,q)
        else return DIAMETER (S_1)
```

Figure 5: A randomized algorithm for computing the diameter in 3D.

The correctness of the above algorithm is easy to check. The only nontrivial step in the above algorithm is computing I and S_1. If δ is the Euclidean metric, I can be computed in $O(|S| \log |S|)$ expected time, using the ball-intersection algorithm. S_1 can then be computed in additional $O(|S| \log |S|)$ time, using any optimal planar point-location algorithm (see, e.g., [208]). Hence, each recursive step of the algorithm takes $O(|S| \log |S|)$ expected time. Since p is chosen randomly, $|S_1| \leq 2|S|/3$ with high probability, which implies that the expected running time of the overall algorithm is $O(n \log n)$.

It was a challenging open problem whether an $O(n \log n)$-time deterministic algorithm can be developed for computing the intersection of n congruent balls in 3-space. This has been answered in the affirmative by Amato et al. [31], following a series of near-linear time but weaker deterministic algorithms [51, 177, 202]. Amato et al. derandomized the Clarkson-Shor algorithm, using several sophisticated techniques.[3] Their algorithm yields an $O(n \log^3 n)$-time algorithm for computing the diameter. Obtaining an optimal $O(n \log n)$-time deterministic algorithm for computing the diameter in 3-space still remains elusive.

8.2 Closest line pair

Given a set L of n lines in $\mathbb{R}^3$, we wish to compute a closest pair of lines in L. Let $d(L, L')$ denote the Euclidean distance between the closest pair of lines in $L \times L'$, for two disjoint sets L, L' of lines. Two algorithms for this problem, both based on parametric searching, were given independently by Chazelle et al. [51] and by Pellegrini [199]; both algorithms run in $O(n^{8/5+\varepsilon})$ time. Using Plücker coordinates [53, 220] and range-searching data structures, the algorithms construct, in $O(n^{8/5+\varepsilon})$ time, a family of pairs $\{(L_1, L'_1), \ldots, (L_k, L'_k)\}$, so that every line in L_i lies below (in the z-direction) all the lines of L'_i and $d(L, L') = \min_{1 \leq i \leq k} d(L_i, L'_i)$. Hence, it suffices to compute a closest pair in $L_i \times L'_i$, for each $i \leq k$, which can be done using parametric searching. The decision procedure is: For a given real number r, determine whether $d(L_i, L'_i) \leq r$, for each $i \leq k$. Since lines in 3-space have

[3] An earlier attempt by Brönnimann et al. [44] to derandomize Clarkson-Shor algorithm had an error.

four degrees of freedom, each of these subproblems can be transformed to the following point-location problem in $\mathbb{R}^4$: Given a set S of n points in $\mathbb{R}^4$ (representing the lines in L_i) and a set ? of m surfaces, each being the graph of an algebraic trivariate function of constant degree (each surface is the locus of lines in $\mathbb{R}^3$ that pass above a line of L_i' at distance r from it), determine whether every point in S lies below all the surfaces of ? . It is shown in [51] that this point-location problem can be solved in time $O(n^{4/5+\varepsilon}m^{4/5+\varepsilon})$, which implies an $O(n^{8/5+\varepsilon})$-time algorithm for computing $d(L, L')$. Agarwal and Sharir [14] have shown that $d(L_i, L_i')$ can be computed in $O(n^{3/4+\varepsilon}m^{3/4+\varepsilon})$ expected time, by replacing parametric searching with randomization and by exploiting certain geometric properties that the surfaces in ? possess. Roughly speaking, this is accomplished by generalizing the Clarkson-Shor algorithm for computing the diameter, described in Figure 5. However, this algorithm does not improve the running time for computing $d(L, L')$, because we still need $O(n^{8/5+\varepsilon})$ time for constructing the pairs (L_i, L_i').

If we are interested in computing a pair of lines with the minimum vertical distance, the running time can be improved to $O(n^{4/3+\varepsilon})$ [199].

8.3 Distance between polytopes

We wish to compute the Euclidean distance $d(\mathcal{P}_1, \mathcal{P}_2)$ between two given convex polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ in $\mathbb{R}^d$. If the polytopes intersect, then this distance is 0. If they do not intersect, then this distance equals the maximum distance between two parallel hyperplanes separating the polytopes; such a pair of hyperplanes is unique, and they are orthogonal to the segment connecting two points $a \in \mathcal{P}_1$ and $b \in \mathcal{P}_2$ with $d(a, b) = d(\mathcal{P}_1, \mathcal{P}_2)$. It is shown by Gärtner [109] that this problem is LP-type, with combinatorial dimension at most $d + 2$ (or $d + 1$, if the polytopes do not intersect). It is also shown there that the primitive operations can be performed with expected $e^{O(\sqrt{d})}$ arithmetic operations. Hence, the problem can be solved by the general LP-type algorithm, whose expected number of arithmetic operations is $O(d^2 n) + e^{O(\sqrt{d \log d})}$, where n is the total number of facets in $\mathcal{P}_1$ and $\mathcal{P}_2$. For $d = 2$, the maximum and the minimum distance between two convex polygons can be computed in $O(\log n)$ time, assuming that the vertices of each P_i are stored in an array, sorted in a clockwise order [92].

8.4 Selecting distances

Let S be a set of n points in the plane, and let $1 \leq k \leq \binom{n}{2}$ be an integer. We wish to compute the k-th smallest distance between a pair of points of S. This can be done using parametric searching. The decision problem is to compute, for a given real r, the sum $\sum_{p \in S} |D_r(p) \cap (S - \{p\})|$, where $D_r(p)$ is the closed disk of radius r centered at p. (This sum is twice the number of pairs of points of S at distance $\leq r$.) Agarwal et al. [5] gave an $O(n^{4/3} \log^{4/3} n)$ expected-time randomized algorithm for the decision problem,

using the random-sampling technique of [64], which yields an $O(n^{4/3}\log^{8/3} n)$ expected-time algorithm for the distance-selection problem. Goodrich [115] derandomized this algorithm, at a cost of an additional polylogarithmic factor in the running time. Katz and Sharir [153] obtained an expander-based $O(n^{4/3}\log^{3+\varepsilon} n)$-time (deterministic) algorithm for this problem. See also [207].

8.5 Shape matching

Let P and Q be two polygons with m and n edges, respectively. The problem is to measure the 'resemblance' between P and Q, that is, to determine how well can a copy of P fit Q, if we allow P to translate or to both translate and rotate. The *Hausdorff distance* is one of the common ways of measuring resemblance between two (fixed) sets P and Q [139]; it is defined as

$$H(P,Q) = \max\left\{\max_{a\in P}\min_{b\in Q} d(a,b),\ \max_{a\in Q}\min_{b\in P} d(a,b)\right\},$$

where $d(\cdot,\cdot)$ is the Euclidean distance.

If we allow P to translate only, then we want to compute $\min_v H(P+v,Q)$. The problem has been solved by Agarwal et al. [18], using parametric searching, in $O((mn)^2\log^3(mn))$ time, which is significantly faster than the previously best known algorithm by Alt et al. [30]. If P and Q are finite sets of points, a more efficient solution, not based on parametric searching, is proposed by Huttenlocher et al. [140]. Their solution, however, does not apply to the case of polygons. If we measure distance by the L_∞-metric, faster algorithms, based on parametric searching, are developed in [55, 57].

If we allow P to translate and rotate, then computing the minimum Hausdorff distance becomes significantly harder. Chew et al. [56] have given an $O(m^2n^2\log^3 mn)$-time algorithm when both P and Q are finite point sets, and an $O(m^3n^2\log^3 mn)$-time algorithm when P and Q are polygons.

Another way of measuring the resemblance between two polygons P and Q is by computing the area of their intersection (or, rather, of their symmetric difference). Suppose we wish to minimize the area of the symmetric difference between P and Q, under translation of P. For this case, de Berg et al. [73] gave an $O(n\log n)$-time algorithm, using the prune-and-search paradigm. Their algorithm extends to higher dimensions at a polylogarithmic cost, using parametric searching.

8.6 Surface simplification

A generic *surface-simplification* problem is defined as follows: Given a polyhedral object P in $\mathbb{R}^3$ and an error parameter $\varepsilon > 0$, compute a polyhedral approximation Π of P with the minimum number of vertices, so that the maximum distance between P and Π is at most

ε. There are several ways of defining the maximum distance between P and Π, depending on the application. We will refer to an object that lies within ε distance from P as an *ε-approximation* of P. Surface simplification is a central problem in graphics, geographic information systems, scientific computing, and visualization.

One way of solving the problem is to run a binary search on the number of vertices of the approximating surface. We then need to solve the decision problem of determining whether there exists an ε-approximation with at most k vertices, for some given k. Unfortunately, this problem is NP-Hard [20], so one seeks efficient techniques for computing an ε-approximation of size (number of vertices) close to k_{OPT}, where k_{OPT} is the minimum size of an ε-approximation. Although several ad-hoc algorithms have been developed for computing an ε-approximation [74, 75, 135, 136, 168], none of them guarantees any reasonable bound on the size of the output, and many of them do not even ensure that the maximum distance between the input and the output surface is indeed at most ε. There has been some recent progress on developing polynomial-time approximation algorithm for computing ε-approximations in some special cases.

The simplest, but nevertheless an interesting, special case is when P is a convex polytope (containing the origin). In this case we wish to compute another convex polytope Q with the minimum number of vertices so that $(1-\varepsilon)P \subseteq Q \subseteq (1+\varepsilon)P$ (or so that $P \subseteq Q \subseteq (1+\varepsilon)P$). We can thus pose a more general problem: Given two convex polytopes $P_1 \subseteq P_2$ in $\mathbb{R}^3$, compute a convex polytope Q with the minimum number of vertices such that $P_1 \subseteq Q \subseteq P_2$. Das and Joesph [71] have attempted to prove that this problem is NP-Hard, but their proof contains an error, and it still remains an open problem. Mitchell and Suri [192] have shown that there exists a nested polytope Q with at most $3k_{\mathrm{OPT}}$ vertices, whose vertices are a subset of the vertices of P_2. The problem can now be formulated as a hitting-set problem, and, using a greedy approach, they presented an $O(n^3)$-time algorithm for computing a nested polytope with $O(k_{\mathrm{OPT}} \log n)$ vertices. Clarkson [61] showed that the randomized technique described in Section 5 can compute a nested polytope with $O(k_{\mathrm{OPT}} \log k_{\mathrm{OPT}})$ vertices in $O(n \log^c n)$ expected-time, for some constant $c > 0$. Brönnimann and Goodrich [45] extended Clarkson's algorithm to obtain a polynomial-time, deterministic algorithm that constructs a nested polytope with $O(k_{\mathrm{OPT}})$ vertices.

A widely-studied special case of surface simplification, motivated by applications in geographic information systems and scientific computing, is when P is a polyhedral terrain (i.e., the graph of a continuous piecewise-linear bivariate function). In most of the applications, P is represented as a finite set of n points, sampled from the input surface, and the goal is to compute a polyhedral terrain Q with the minimum number of vertices, such that the vertical distance between any point of P and Q is at most ε. Agarwal and Suri [20] showed that this problem is NP-Hard. They also gave a polynomial-time algorithm for computing an ε-approximation of size $O(k_{\mathrm{OPT}} \log k_{\mathrm{OPT}})$, by reducing the problem to a geometric set-cover problem, but the running time of their algorithm is $O(n^8)$, which is rather high. Agarwal and Desiken [6] have shown that Clarkson's randomized algorithm can

be extended to compute a polyhedral terrain of size $O(k_{\mathrm{OPT}}^2 \log^2 k_{\mathrm{OPT}})$ in expected time $O(n^{2+\delta} + k_{\mathrm{OPT}}^3 \log^3 k_{\mathrm{OPT}})$. The survey paper by Heckbert and Garland [122] summarizes most of the known results on terrain simplification.

A dual version of the problem of computing an ε-approximation is: Given a polyhedral surface P and an integer k, compute an approximating surface Q that has at most k vertices, whose distance from P is the smallest possible. Very little is known about this problem, except in the plane. Goodrich [116] showed that, given a set S of n points in the plane, an x-monotone polygonal chain Q with at most k vertices that minimizes the maximum vertical distance between Q and the points of S can be computed in time $O(n \log n)$. His algorithm is based on the parametric-searching technique, and uses Cole's improvement of parametric searching. (See [116] for other related work on this problem.) If the vertices of Q are required to be a subset of S, the best known algorithm is by Varadarajan [229]; it is based on parametric searching, and its running time is $O(n^{4/3+\varepsilon})$.

9 Statistical Estimators and Related Problems

9.1 Plane fitting

Given a set S of n points in $\mathbb{R}^3$, we wish to fit a plane h through S so that the maximum distance between h and the points of S is minimized. This is the same problem as computing the *width* of S (the smallest distance between a pair of parallel supporting planes of S), which is considerably harder than the two-dimensional variant mentioned in Section 7.2. Houle and Toussaint [138] gave an $O(n^2)$-time algorithm for computing the width in $\mathbb{R}^3$. This can be improved using parametric searching. The decision procedure is to determine, for a given distance w, whether the convex hull of S has two 'antipodal' edges, such that the two parallel planes containing these edges are supporting planes of S and lie at distance $\leq w$. (One also needs to consider pairs of parallel planes, one containing a facet of $\mathrm{conv}(S)$ and the other passing through a vertex. However, it is easy to test all these pairs in $O(n \log n)$ time.) The major technical issue here is to avoid having to test quadratically many pairs of antipodal edges, which may exist in the worst case. Chazelle et al. [51] gave an algorithm that is based on parametric searching and runs in time $O(n^{8/5+\varepsilon})$ (see [2] for an improved bound). They reduced the width problem to the problem of computing a closest pair between two sets L, L' of lines in $\mathbb{R}^3$ (each line containing an edge of the convex hull of S), such that each line in L lies below all the lines of L'. The fact that this latter problem now has an improved $O(n^{3/2+\varepsilon})$ expected-time solution [14] implies that the width can be computed in expected time $O(n^{3/2+\varepsilon})$. See [160, 176, 221, 222, 230] for other results on hyperplane fitting.

9.2 Circle fitting

Given a set S of n points in the plane, we wish to fit a circle C through S so that the maximum distance between the points of S and C is minimized. This is equivalent to finding an annulus of minimum width that contains S. Ebara et al. [91] observed that the center of a minimum-width annulus is either a vertex of the closest-point Voronoi diagram of S, or a vertex of the farthest-point Voronoi diagram, or an intersection point of a pair of edges of the two diagrams. Based on this observation, they obtained a quadratic-time algorithm. Using parametric searching, Agarwal et al. [18] have shown that the center of the minimum-width annulus can be found without checking all of the $O(n^2)$ candidate intersection points explicitly; their algorithm runs in $O(n^{8/5+\varepsilon})$ time; see also [2] for an improved solution. Using randomization and an improved analysis, the expected running time has been improved to $O(n^{3/2+\varepsilon})$ by Agarwal and Sharir [14]. Finding an annulus of minimum area that contains S is a simpler problem, since it can be formulated as an instance of linear programming in $\mathbb{R}^4$, and can thus be solved in $O(n)$ time [183]. In certain metrology applications [134, 206, 231], one wants to fit a circle C through S so that the sum of distances between C and the points of S is minimized. No algorithm is known for computing an exact solution, though several numerical techniques have been proposed; see [38, 161, 224]. See [162, 223] for other variants of the circle-fitting problem and for some special cases.

9.3 Cylinder fitting

Given a set S of n points in $\mathbb{R}^3$, we wish to find a cylinder of smallest radius that contains S. Using parametric searching, the decision problem in this case can be rephrased as: Given a set $\mathcal{B}$ of n balls of a fixed radius r in $\mathbb{R}^3$, determine whether there exists a line that intersects all the balls of $\mathcal{B}$ (the balls are centered at the points of S and the line is the symmetry axis of a cylinder of radius r that contains S). Agarwal and Matoušek [10] showed that finding such a line can be reduced to computing the convex hull of a set of n points in $\mathbb{R}^9$, which, combined with parametric searching, leads to an $O(n^4 \log^{O(1)} n)$-time algorithm for finding a smallest cylinder enclosing S; see e.g. [209]. The bound has recently been improved by Agarwal et al. [4] to $O(n^{3+\varepsilon})$, by showing that the combinatorial complexity of the space of all lines that intersect all the balls of $\mathcal{B}$ is $O(n^{3+\varepsilon})$, and by designing a different algorithm, also based on parametric searching, whose decision procedure calculates this space of lines and determines whether it is nonempty. Faster algorithms have been developed for some special cases [103, 209]. Agarwal et al. [4] also gave an $O(n/\delta^2)$-time algorithm to compute a cylinder of radius $\leq (1+\delta)r^*$ containing all the points of S, where r^* is the radius of the smallest cylinder enclosing S.

Note that this problem is different from those considered in the two previous subsections. The problem analogous to those studied above would be to find a cylindrical shell (a region

enclosed between two concentric cylinders) of smallest width (difference between the radii of the cylinders), which contains a given point set S. This problem is considerably harder, and no solution for it that improves upon the naive brute-force technique is known.

9.4 Center points

Given a set S of n points in the plane, we wish to compute a *center point* $\sigma \in \mathbb{R}^2$, such that any halfplane containing σ also contains at least $\lfloor n/3 \rfloor$ points of S. It is a known consequence of Helly's Theorem that σ always exists [93]. In a dual setting, let L be the set of lines dual to the points in S, and let K_1, K_2 be the convex hulls of the $\lfloor n/3 \rfloor$ and $\lfloor 2n/3 \rfloor$ levels of the arrangement $\mathcal{A}(L)$, respectively.[4] The dual of a center point of S is a line separating K_1 and K_2. This implies that the set of center points is a convex polygon with at most $2n$ edges.

Cole et al. [68] gave an $O(n \log^3 n)$-time algorithm for computing a center point, using multi-dimensional parametric searching. Using the prune-and-search paradigm, Matoušek [169] obtained an $O(n \log^3 n)$-time algorithm for computing K_1 and K_2, which in turn yields the set of all center points. Recently, Jadhav and Mukhopadhyay [144] gave a linear-time algorithm for computing a center point, using a direct and elegant technique.

Near-quadratic algorithms for computing a center point in three dimensions were developed in [68, 195]. Clarkson et al. [63] gave an efficient algorithm for computing an approximate center point in $\mathbb{R}^d$.

9.5 Ham-sandwich cuts

Let $S_1, \ldots, S_d$ be d (finite) point sets in $\mathbb{R}^d$. A *ham-sandwich cut* is a hyperplane h such that each of the two open halfspaces bounded by h contains at most $\lfloor |S_i|/2 \rfloor$ points of S_i, for each $1 \leq i \leq d$. The ham-sandwich theorem (see, e.g., [93]) guarantees the existence of such a cut. For $d = 2$, there is always a ham-sandwich cut whose dual is an intersection point of the median levels of $\mathcal{A}(L_1)$ and $\mathcal{A}(L_2)$, where L_i is the set of lines dual to the points in S_i, for $i = 1, 2$. It can be shown that the number of intersection points between the median levels of $\mathcal{A}(L_1)$ and of $\mathcal{A}(L_2)$ is always odd.

Several prune-and-search algorithms have been proposed for computing a ham-sandwich cut in the plane. Megiddo [184] gave a linear-time algorithm for the special case where S_1 and S_2 are linearly separable. Modifying this algorithm, Edelsbrunner and Waupotitsch [94] gave an $O(n \log n)$-time algorithm when S_1 and S_2 are not necessarily linearly separable. A linear-time, recursive algorithm for this general case is given by Lo and Steiger [166].

[4] The *level* of a point p with respect to $\mathcal{A}(L)$ is the number of lines lying strictly below p. The *k-level* of $\mathcal{A}(L)$ is the closure of the set of edges of $\mathcal{A}(L)$ whose level is k (the level is fixed over an edge of $\mathcal{A}(L)$); each k-level is an x-monotone connected polygonal chain.

It works as follows. At each level of recursion, the algorithm maintains two sets of lines, R and B, and two integers p, q, such that any intersection point between K_R^p, the p-level of $\mathcal{A}(R)$, and K_B^q, the q-level of $\mathcal{A}(B)$, is dual to a ham-sandwich cut of the original sets; moreover, the number of such intersections is guaranteed to be odd. The goal is to compute an intersection point of K_R^p and K_B^q. Initially R and B are the sets of lines dual to S_1 and S_2, and $p = \lfloor |S_1|/2 \rfloor$, $q = \lfloor |S_2|/2 \rfloor$. Let r be a sufficiently large constant. One then computes a $(1/r)$-cutting Ξ of $R \cup B$. At least one of the triangles of Ξ contains an odd number of intersection points of the levels. By computing the intersection points of the edges of Ξ with $R \cup B$, such a triangle Δ can be found in linear time. Let $R_\Delta \subseteq R$ and $B_\Delta \subseteq B$ be the subsets of lines in R and B, respectively, that intersect Δ, and let p' (resp. q') be the number of lines in R (resp. B) that lie below Δ. We then solve the problem recursively for R_Δ and B_Δ with the respective levels $p^* = p - p'$ and $q^* = q - q'$. It easily follows that the p^*-level of $\mathcal{A}(R_\Delta)$ and the q^*-level of $\mathcal{A}(B_\Delta)$ intersect at an odd number of points. Since $|R_\Delta| + |B_\Delta| = O(n/r)$, the total running time of the algorithm is $O(n)$. Lo et al. [165] extend this approach to $\mathbb{R}^3$, and obtain an $O(n^{3/2})$-time algorithm for computing ham-sandwich cuts in three dimensions.

10 Placement and Intersection

10.1 Intersection of polyhedra

Given a set $\mathcal{P} = \{P_1, \ldots, P_m\}$ of m convex polyhedra in $\mathbb{R}^d$, with a total of n facets, is their common intersection $I = \bigcap_{i=1}^m P_i$ nonempty? Of course, this is an instance of linear programming in $\mathbb{R}^d$ with n constraints, but the goal is to obtain faster algorithms that depend on m more significantly than they depend on n. Reichling [203] presented an $O(m \log^2 n)$-time prune-and-search algorithm for $d = 2$. His algorithm maintains a vertical strip W bounded by two vertical lines b_l, b_r, such that $I \subseteq W$. Let k be the total number of vertices of all the P_i's lying inside W. If $k \leq m \log n$, the algorithm explicitly computes $I \cap W$ in $O(m \log^2 n)$ time. Otherwise, it finds a vertical line ℓ inside W such that both W^+ and W^- contain at least $k/4$ vertices, where W^+ (resp. W^-) is the portion of W lying to the right (resp. to the left) of ℓ. By running a binary search on each P_i, one can determine whether ℓ intersects P_i and, if so, obtain the top and bottom edges of P_i intersecting ℓ. This allows us to compute the intersection $I \cap \ell$ as the intersection of m intervals, in $O(m)$ time. If this intersection is nonempty, we stop, since we have found a point in I. Otherwise, if one of the polygons of $\mathcal{P}$ lies fully to the right (resp. to the left) of ℓ, then I cannot lie in W^- (resp. in W^+); if one polygon lies fully in W^- and another lies fully in W^+, then clearly $I = \emptyset$. Finally, if ℓ intersects all the P_i's, but their intersection along ℓ is empty, then, following the same technique as in Megiddo's two-dimensional linear-programming algorithm [181], one can determine, in additional $O(m)$ time, which of W^+, W^- can be asserted not to contain I. Hence, if the algorithm has not stopped, it needs to recurse

in only one of the slabs W^+, W^-. Since the algorithm prunes a fraction of the vertices in each stage, it terminates after $O(\log n)$ stages, from which the asserted running time follows easily. Reichling [204] and Eppstein [98] extended this approach to $d = 3$, but their approaches do not extend to higher dimensions. However, if we have a comparison-based data structure that can determine in $O(\log n)$ time whether a query point lies in a specified P_i, then, using multi-dimensional parametric searching, we can determine in $O(m \log^{O(1)} n)$ time whether $I \neq \emptyset$.

10.2 Polygon placement

Let P be a convex m-gon, and let Q be a closed planar polygonal environment with n edges. We wish to compute the largest similar copy of P (under translation, rotation, and scaling) that can be placed inside Q. Using generalized Delauney triangulation induced by P within Q, Chew and Kedem [58] obtained an $O(m^4 n^2 2^{\alpha(n)} \log n)$-time algorithm. Faster algorithms have been developed using parametric searching [3, 217]. The decision problem in this case is: Given a convex polygon B with m edges (a scaled copy of P) and a planar polygonal environment Q with n edges, can B be placed inside Q (allowing translation and rotation)? Each placement of B can be represented as a point in $\mathbb{R}^3$, using two coordinates for translation and one for rotation. Let FP denote the resulting three-dimensional space of all free placements of B inside Q. Leven and Sharir [164] have shown that the complexity of FP is $O(mn\lambda_6(mn))$, where $\lambda_s(n)$ is the maximum length of a Davenport–Schinzel sequence of order s composed of n symbols [216] (it is almost linear in n for any fixed s). Sharir and Toledo [217] gave an $O(m^2 n \lambda_6(mn) \log mn)$-time algorithm to determine whether $FP \neq \emptyset$ — they first compute a superset of the vertices of FP, in $O(mn\lambda_6(mn) \log mn)$ time, and then spend $O(m \log n)$ time for each of these vertices to determine whether the corresponding placement of B is free, using a standard triangle range-searching data structure. Recently, Agarwal et al. [3] gave an $O(mn\lambda_6(mn) \log mn)$ expected-time randomized algorithm to compute FP. Plugging these algorithms into the parametric searching machinery, one can obtain an $O(m^2 n \lambda_6(mn) \log^3 mn \log\log mn)$-time deterministic algorithm, or an $O(mn\lambda_6(mn) \log^4 mn)$ expected-time randomized algorithm, for computing a largest similar placement of P inside Q.

Faster algorithms are known for computing a largest placement of P inside Q in some special cases. If both P and Q are convex, then a largest similar copy of P inside Q can be computed in time $O(mn^2 \log n)$ [1]; if P is not allowed to rotate, then the running time is $O(m + n \log^2 n)$ [225].

The *biggest-stick* problem is another interesting special case of the largest-placement problem; here Q is a simple polygon and P is a line segment. In this case, we are interested in finding the longest segment that can be placed inside Q. This problem can be solved using a divide-and-conquer algorithm, developed in [18], and later refined in [2, 14]. It proceeds as follows: Partition Q into two simple polygons Q_1, Q_2 by a diagonal ℓ so that

each of Q_1 and Q_2 has at most $2n/3$ vertices. Recursively compute the longest segment that can be placed in each Q_i, and then determine the longest segment that can be placed in Q and that intersects the diagonal ℓ. The decision step for this subproblem is to determine whether there exists a placement of a line segment of length w that lies inside Q and crosses ℓ. Agarwal et al. [18] have shown that this problem can be reduced to that in which we are given a set S of points and a set ? of algebraic surfaces in $\mathbb{R}^4$, where each surface is the graph of a trivariate function, and we wish to determine whether every point of S lies below all the surfaces of ? . Agarwal and Sharir [14] gave a randomized algorithm with $O(n^{3/2+\varepsilon})$ expected running time for this point-location problem. Using randomization, instead of parametric searching, they obtained an $O(n^{3/2+\varepsilon})$ expected-time procedure for the overall merge step (finding the biggest stick that crosses ℓ). The total running time of the algorithm is therefore also $O(n^{3/2+\varepsilon})$. Finding a longest segment inside Q whose endpoints are vertices of Q is a simpler problem, and can be solved by a linear-time algorithm due to Hershberger and Suri [129].

10.3 Collision detection

Let P and Q be two (possibly nonconvex) polyhedra in $\mathbb{R}^3$. P is assumed to be fixed and Q to move along a given trajectory π. The goal is to determine the first position on π, if any, at which Q intersects P. This problem can be solved using parametric searching. Suppose, for example, that Q is only allowed to translate along a line. Then the decision problem is to determine whether Q intersects P as it translates along a segment e in $\mathbb{R}^3$. Let $Q_e = Q \oplus e$ be the Minkowski sum of Q and e. Then Q intersects P as it translates along e if and only if Q_e intersects P. This intersection problem can be solved in $O(n^{8/5+\varepsilon})$ time, using simplex range-searching data structures [198, 210]. Plugging this into the parametric searching machinery, we can compute, in $O(n^{8/5+\varepsilon})$ time, the first intersection of Q with P as Q moves along a line.

If Q rotates around a fixed axis ℓ, then the decision problem is to determine whether Q intersects P as it rotates by a given angle θ from its initial position. In this case, each edge of Q sweeps a section of a hyperboloid. Schömer and Thiel [210] have shown that, using a standard linearization technique (as described in [10, 234]), the intersection-detection problem can be formulated as an instance of simplex range searching in $\mathbb{R}^5$, and can be solved in time $O(n^{8/5+\varepsilon})$. Plugging this algorithm into the parametric searching technique, we can also compute the first intersection in time $O(n^{8/5+\varepsilon})$.

Gupta et al. [119] have studied various collision-detection problems for a set of moving points in the plane. For example, they give an $O(n^{5/3}\log^{6/5} n)$-time algorithm for determining whether a collision occurs in a set of points, each moving in the plane along a line with constant velocity.

11 Query-Type Problems

Parametric searching has also been successfully applied in designing efficient data structures for a number of query-type problems. In this section we discuss a few of these problems, including ray shooting and linear optimization queries.

11.1 Ray shooting

The general ray-shooting problem can be defined as follows. Preprocess a given set S of objects in $\mathbb{R}^d$ (usually $d = 2$ or 3), so that the first object hit by a query ray can be computed efficiently. The ray-shooting problem arises in computer graphics, visualization, and in many other geometric problems [9, 10, 11, 200]. The connection between ray shooting and parametric searching was observed by Agarwal and Matoušek [9]. Here the decision problem is to determine, for a specified point σ on the query ray ρ, whether the initial segment $s\sigma$ of ρ intersects any object in S (where s is the origin of ρ). Hence, we need to execute generically, in the parametric-searching style, an appropriate intersection-detection query procedure on the segment $s\sigma$, where σ is the (unknown) first intersection point of ρ and the objects in S. Based on this technique, several efficient ray-shooting data structures have been developed [2, 9, 15, 16]. We illustrate this technique by giving a simple example.

Let S be a set of n lines in the plane, and let S^* be the set of points dual to the lines of S. A segment e intersects a line of S if and only if the double-wedge e^* dual to e contains a point of S^*. Hence, a segment intersection-detection query for S can be answered by preprocessing S^* into a triangle (or a wedge) range-searching structure; see, e.g., [171, 172]. Roughly speaking, we construct a *partition tree* T on S^* as follows. We fix a sufficiently large constant r. If $|S^*| \leq 2r$, T consists of a single node storing S^*. Otherwise, using a result of Matoušek [171], we construct, in $O(n)$ time, a family of pairs $\Pi = \{(S_1^*, \Delta_1), \ldots, (S_u^*, \Delta_u)\}$ such that (i) $S_1^*, \ldots, S_u^*$ form a partition of S^*, (ii) $n/r \leq |S_i^*| \leq 2n/r$ for each i, (iii) each Δ_i is a triangle containing S_i^*, and (iv) every line intersects at most $c\sqrt{r}$ triangles Δ_i of Π, for some absolute constant c (independent of r). We recursively construct a partition tree T_i on each S_i^*, and attach it as the i-th subtree of T. The root of T_i stores the simplex Δ_i. The total size of T is linear, and the time spent in constructing T is $O(n \log n)$.

Let e be a query segment, and let e^* be its dual double wedge. To determine whether e intersects any line of S (that is, whether e^* contains any point of S^*), we traverse T in a top-down fashion, starting from the root. Let v be a node visited by the algorithm. If v is a leaf, we explicitly check whether any point of S_v^* lies in the double wedge e^*. Suppose then that v is an internal node. If $\Delta_v \subseteq e^*$, then clearly $e^* \cap S \neq \emptyset$, and we stop. If $\Delta_v \cap e^* = \emptyset$, then we stop processing v and do not visit any of its children. If ∂e^* intersects Δ_v, we recursively visit all the children of v. Let $Q(n_v)$ denote the number of nodes in the subtree rooted at v visited by the query procedure (n_v is the size of S_v^*). By construction, a line intersects at most $c\sqrt{r}$ triangles of Π_v (the partition constructed at v), so ∂e^* intersects at

most $2c\sqrt{r}$ triangles of Π_v. Hence, we obtain the following recurrence:

$$Q(n_v) \leq 2c\sqrt{r}Q(2n_v/r) + O(r).$$

The solution of the above recurrence is $O(n^{1/2+\varepsilon})$, for any $\varepsilon > 0$, provided r is chosen sufficiently large (as a function of ε). Since the height of T is $O(\log n)$, we can answer a query in $O(\log n)$ parallel time, using $O(n^{1/2+\varepsilon})$ processors, by visiting the nodes of the same level in parallel.

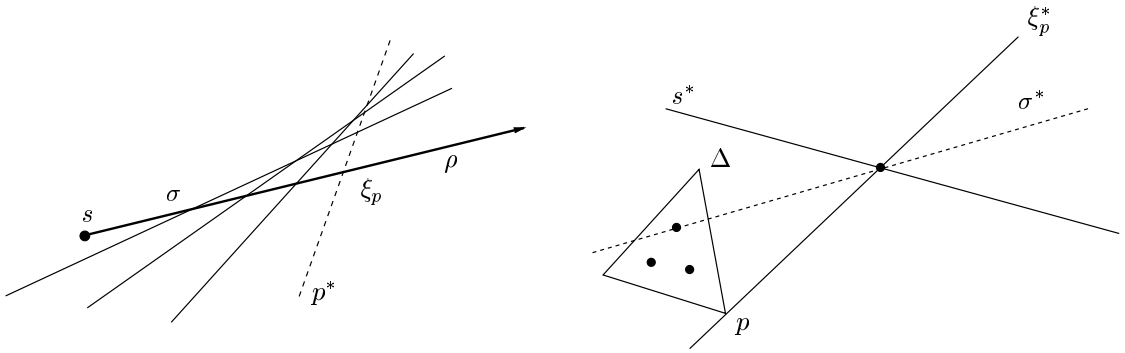

Figure 6: A ray-shooting query

Returning to the task of answering a ray-shooting query, let ρ be a query ray with origin s, and let σ be the (unknown) first intersection point of ρ with a line of S. We compute σ by running the parallel version of the segment intersection-detection procedure generically, in the parametric searching style, on the segment $\gamma = s\sigma$. At each node v that the query procedure visits, it tests whether $\Delta_v \subseteq \gamma^*$ or $\Delta_v \cap \partial\gamma^* \neq \emptyset$. Since σ is the only indeterminant in these tests, the tests reduce to determining, for each vertex p of Δ_v, whether p lies above, below, or on the line σ^* dual to σ; see Figure 6. Let ξ_p be the intersection point of the dual line p^* and the line containing ρ; and set $\gamma_p = s\xi_p$. By determining whether the segment γ_p intersects a line of S, we can determine whether p lies above or below σ^*. Hence, using this parametric searching approach, a ray-shooting query can be answered in $O(n^{1/2+\varepsilon})$ time.

Several other ray-shooting data structures based on this technique have been developed in [9, 10, 11, 200].

11.2 Linear-optimization queries

We wish to preprocess a set H of halfspaces in $\mathbb{R}^d$ into a linear-size data structure so that, for a linear objective function c, we can efficiently compute the vertex of $\bigcap H$ that minimizes c. Using multi-dimensional parametric searching and data structures for answering halfspace-emptiness queries, Matoušek [172] presented an efficient algorithm for answering linear-

optimization queries. A slightly faster randomized algorithm has recently been proposed by Chan [47]. Linear-optimization queries can be used to answer many other queries. For example, using Matoušek's technique and a dynamic data structure for halfspace range searching, the 1-center of a set S of points in $\mathbb{R}^d$ can be maintained dynamically, as points are inserted into or deleted from S. See [7, 11, 172] for additional applications of multi-dimensional parametric searching for query-type problems.

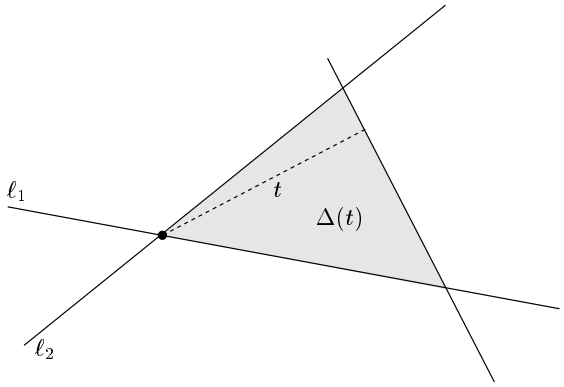

Figure 7: An extremal-placement query

11.3 Extremal placement queries

Let S be a set of n points in $\mathbb{R}^d$. We wish to preprocess S into a data structure so that queries of the following form can be answered efficiently: Let $\Delta(t)$, for $t \in \mathbb{R}$, be a family of simplices such that $\Delta(t_1) \subseteq \Delta(t_2)$, for any $t_1 \leq t_2$, and such that for any t, $\Delta(t)$ can be computed in $O(1)$ time. The goal is to compute the largest value $t = t_{\max}$ for which the interior of $\Delta(t_{\max})$ does not contain any point of S. For example, let Δ be a fixed simplex and let $\Delta(t) = t\Delta$ be the simplex obtained by scaling Δ by a factor of t. We thus want to find the largest dilated copy of Δ that does not contain any point of S. As another example, let ℓ_1, ℓ_2 be two lines, and let m be a constant. Define $\Delta(t)$ to be the triangle formed by ℓ_1, ℓ_2, and the line with slope m and at distance t from the intersection point $\ell_1 \cap \ell_2$; see Figure 7. These problems arise in many applications, including hidden surface removal [200] and Euclidean shortest paths [190].

By preprocessing S into a simplex range-searching data structure, we can determine whether $\Delta(t_0) \cap S = \emptyset$, for any given t_0, and then plug this query procedure into the parametric searching machinery, thereby obtaining the desired $t_{\max}$. The best-known data structure for simplex range searching can answer a query in time $O(m/n^{1/d} \log^{d+1} n)$, using $O(m)$ space, so an extremal placement query can be answered in time $O(m/n^{1/d} \log^{2(d+1)} n)$.

12 Discussion

In this survey we have reviewed several techniques for geometric optimization, and discussed many geometric problems that benefit from these techniques. There are of course quite a few non-geometric parametric optimization problems that can also be solved efficiently using these techniques. For example, parametric searching has been applied to develop efficient algorithm for the following problems: (i) let G be a directed graph, in which the weight of each edge e is a d-variate linear function $w_e(\mathbf{x})$, and let s and t be two vertices in G, find the point $\mathbf{x} \in \mathbb{R}^d$, so that the maximum flow from s to t is maximized over all points $\mathbf{x} \in \mathbb{R}^d$ [65]; (ii) compute the minimum *edit distance* between two given sequences, where the cost of performing an insertion, deletion, and substitution is a univariate linear function [120].

There are several other geometric optimization problems that are not discussed here, and we conclude by mentioning two classes of them. The first class is the *optimal motion-planning* problem, where we are given a moving robot B, an environment with obstacles, and two placements of the robot, and we want to compute an "optimal" collision-free path for B between the given placements. The cost of a path depends on B and on the application. In the simplest case, B is point robot, O is a set of polygonal obstacles in the plane or polyhedral obstacles in 3-space, and the cost of a path is its Euclidean length. We then face the Euclidean shortest-path problem, which has been studied intensively in the past decade; see [46, 128, 205]. The problem becomes much harder if B is not a point, because even the notion of optimality is not well defined. See [191] for an excellent survey on this topic.

The second class of problems that we want to mention can be called *geometric graph* problems. Given a set S of points in $\mathbb{R}^d$, we can define a weighted complete graph induced by S, where the weight of an edge (p, q) is the distance between p and q under some suitable metric (two of the most commonly used metrics are the Euclidean and the rectilinear metrics). We can now pose many optimization problems on this graph, including the Euclidean travelling salesperson, Euclidean matching, Euclidean (rectilinear) Steiner trees, and minimum weight triangulation. Although all these problems can be solved using techniques known for general graphs, the hope is that better and/or simpler algorithms can be developed by exploiting the geometry of the problem. There have been several significant developments on geometric graph problems over the last few years, of which the most exciting is an $n^{O(1/\varepsilon)}$-time $(1+\varepsilon)$-approximation algorithm for the Euclidean travelling salesperson problem [35]. We refer the reader to [39] for a survey on approximation algorithms for such geometric optimization problems.

References

[1] P. K. Agarwal, N. Amenta, and M. Sharir, Placement of one convex polygon inside another, Tech. Report CS-1995-29, Duke University, 1995.

[2] P. K. Agarwal, B. Aronov, and M. Sharir, Computing envelopes in four dimensions with applications, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 348–358.

[3] P. K. Agarwal, B. Aronov, and M. Sharir, Motion planning for a convex polygon in a polygonal environment, manuscript, 1996.

[4] P. K. Agarwal, B. Aronov, and M. Sharir, Line transversals of balls and smallest enclosing cylinders in three dimensions, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997.

[5] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri, Selecting distances in the plane, *Algorithmica*, 9 (1993), 495–514.

[6] P. K. Agarwal and P. K. Desikan, An approximation algorithm for terrain simplification, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997.

[7] P. K. Agarwal, A. Efrat, and M. Sharir, Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 39–50.

[8] P. K. Agarwal, A. Efrat, M. Sharir, and S. Toledo, Computing a segment center for a planar point set, *J. Algorithms*, 15 (1993), 314–323.

[9] P. K. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.*, 22 (1993), 794–806.

[10] P. K. Agarwal and J. Matoušek, On range searching with semialgebraic sets, *Discrete Comput. Geom.*, 11 (1994), 393–418.

[11] P. K. Agarwal and J. Matoušek, Dynamic half-space range reporting and its applications, *Algorithmica*, 13 (1995), 325–345.

[12] P. K. Agarwal and M. Sharir, Off-line dynamic maintenance of the width of a planar point set, *Comput. Geom. Theory Appl.*, 1 (1991), 65–78.

[13] P. K. Agarwal and M. Sharir, Planar geometric location problems, *Algorithmica*, 11 (1994), 185–195.

[14] P. K. Agarwal and M. Sharir, Efficient randomized algorithms for some geometric optimization problems, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 326–335.

[15] P. K. Agarwal and M. Sharir, Ray shooting amidst convex polygons in 2D, *J. Algorithms*, 21 (1996), 508–519.

[16] P. K. Agarwal and M. Sharir, Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions, *SIAM J. Comput.*, 25 (1996), 100–116.

[17] P. K. Agarwal, M. Sharir, and S. Toledo, An efficient multi-dimensional searching technique and its applications, Tech. Report CS-1993-20, Dept. Comp. Sci., Duke University, 1993.

[18] P. K. Agarwal, M. Sharir, and S. Toledo, Applications of parametric searching in geometric optimization, *J. Algorithms*, 17 (1994), 292–318.

[19] P. K. Agarwal, M. Sharir, and E. Welzl, The discrete 2-center problem, manuscript, 1996.

[20] P. K. Agarwal and S. Suri, Surface approximation and geometric partitions, *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, 1994, pp. 24–33.

[21] R. Agarwala and D. Fernández-Baca, Weighted multidimensional search and its applications to convex optimization, *SIAM J. Comput.*, 25 (1996), 83–99.

[22] A. Aggarwal and M. M. Klawe, Applications of generalized matrix searching to geometric algorithms, *Discrete Appl. Math.*, 27 (1987), 3–23.

[23] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber, Geometric applications of a matrix-searching algorithm, *Algorithmica*, 2 (1987), 195–208.

[24] A. Aggarwal, D. Kravets, J. K. Park, and S. Sen, Parallel searching in generalized Monge arrays with applications, *Proc. 2nd ACM Sympos. Parallel Algorithms Architect.*, 1990, pp. 259–268.

[25] A. Aggarwal and J. Park, Notes on searching in multidimensional monotone arrays, *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, 1988, pp. 497–512.

[26] M. Ajtai, J. Komlós, and E. Szemerédi, Sorting in $c \log n$ parallel steps, *Combinatorica*, 3 (1983), 1–19.

[27] M. Ajtai and N. Megiddo, A deterministic $poly(\log \log n)$-time n-processor algorithm for linear programming in fixed dimensions, *SIAM J. Comput.*, 25 (1996), 1171–1195.

[28] N. Alon and N. Megiddo, Parallel linear programming in fixed dimension almost surely in constant time, *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.*, 1990, pp. 574–582.

[29] N. Alon and J. Spencer, *The Probabilistic Method*, J. Wiley and Sons, New York, NY, 1993.

[30] H. Alt, B. Behrends, and J. Blömer, Approximate matching of polygonal shapes, *Ann. Math. Artif. Intell.*, 13 (1995), 251–266.

[31] N. M. Amato, M. T. Goodrich, and E. A. Ramos, Parallel algorithms for higher-dimensional convex hulls, *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, 1994, pp. 683–694.

[32] N. Amenta, Bounded boxes, Hausdorff distance, and a new proof of an interesting Helly theorem, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 340–347.

[33] N. Amenta, Helly-type theorems and generalized linear programming, *Discrete Comput. Geom.*, 12 (1994), 241–261.

[34] D. S. Arnon, G. E. Collins, and S. McCallum, Cylindrical algebraic decomposition I: The basic algorithm, *SIAM J. Comput.*, 13 (1984), 865–877.

[35] S. Arora, Polynomial time approximation schemes for Euclidean TSP and other geometric problems, *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci*, 1996, pp. 2–11.

[36] J. Bar-Ilan, G. Kortsarz, and D. Peleg, How to allocate network centers, *J. Algorithms*, 15 (1993), 385–415.

[37] R. Bar-Yehuda, A. Efrat, and A. Itai, A simple algorithm for maintaining the center of a planar point-set, *Proc. 5th Canad. Conf. Comput. Geom.*, 1993, pp. 252–257.

[38] M. Berman, Large sample bias in least squares estimators of a circular arc center and its radius, *Comput. Vision, Graphics, and Image Process*, 45 (1989), 126–128.

[39] M. Bern and D. Eppstein, Approximation algorithms for geometric problems, in: *Approximation Problems for NP-Hard Problems* (D. S. Hochbaum, ed.), PWS Publishing Company, Boston, MA, 1996, pp. 296–345.

[40] B. Bhattacharya, J. Czyzowicz, P. Egyed, G. Toussaint, I. Stojmenović, and J. Urrutia, Computing shortest transversals of sets, *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, 1991, pp. 71–80.

[41] B. Bhattacharya and G. Toussaint, Computing shortest transversals, *Computing*, 46 (1991), 93–119.

[42] B. K. Bhattacharya, S. Jadhav, A. Mukhopadhyay, and J.-M. Robert, Optimal algorithms for some smallest intersection radius problems, *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, 1991, pp. 81–88.

[43] H. Brönnimann and B. Chazelle, Optimal slope selection via cuttings, *Proc. 6th Canad. Conf. Comput. Geom.*, 1994, pp. 99–103.

[44] H. Brönnimann, B. Chazelle, and J. Matoušek, Product range spaces, sensitive sampling, and derandomization, *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.*, 1993, pp. 400–409.

[45] H. Brönnimann and M. T. Goodrich, Almost optimal set covers in finite VC-dimension, *Discrete Comput. Geom.*, 14 (1995), 263–279.

[46] J. Canny and J. H. Reif, New lower bound techniques for robot motion planning problems, *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, 1987, pp. 49–60.

[47] T. M. Chan, Fixed-dimensional linear programming queries made easy, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 284–290.

[48] R. Chandrasekaran and A. Tamir, Algebraic optimization: the Fermat-Weber location problem, *Math. Program.*, 46 (1990), 219–224.

[49] B. Chazelle, Cutting hyperplanes for divide-and-conquer, *Discrete Comput. Geom.*, 9 (1993), 145–158.

[50] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, A singly-exponential stratification scheme for real semi-algebraic varieties and its applications, *Theoret. Comput. Sci.*, 84 (1991), 77–105.

[51] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Diameter, width, closest line pair and parametric searching, *Discrete Comput. Geom.*, 10 (1993), 183–196.

[52] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, *Algorithmica*, 11 (1994), 116–132.

[53] B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi, Lines in space: Combinatorics and algorithms, *Algorithmica*, 15 (1996), 428–447.

[54] B. Chazelle and J. Matoušek, On linear-time deterministic algorithms for optimization proble ms in fixed dimension, *J. Algorithms*, 21 (1996), 579–597.

[55] L. P. Chew, D. Dor, A. Efrat, and K. Kedem, Geometric pattern matching in d-dimensional space, *Proc. 2nd Annu. European Sympos. Algorithms*, *Lecture Notes in Computer Science*, Vol. 979, Springer-Verlag, 1995, pp. 264–279.

[56] L. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets, Geometric pattern matching under Euclidean motion, *Proc. 5th Canad. Conf. Comput. Geom.*, 1993, pp. 151–156.

[57] L. P. Chew and K. Kedem, Improvements on geometric pattern matching problems, *Proc. 3rd Scand. Workshop Algorithm Theory*, *Lecture Notes in Computer Science*, Vol. 621, Springer-Verlag, 1992, pp. 318–325.

[58] L. P. Chew and K. Kedem, A convex polygon among polygonal obstacles: Placement and high-clearance motion, *Comput. Geom. Theory Appl.*, 3 (1993), 59–89.

[59] K. L. Clarkson, Linear programming in $O(n3^{d^2})$ time, *Inform. Process. Lett.*, 22 (1986), 21–24.

[60] K. L. Clarkson, Randomized geometric algorithms, in: *Computing in Euclidean Geometry* (D.-Z. Du and F. K. Hwang, eds.), World Scientific, Singapore, 1992, pp. 117–162.

[61] K. L. Clarkson, Algorithms for polytope covering and approximation, *Proc. 3rd Workshop Algorithms Data Struct.*, *Lecture Notes in Computer Science*, Vol. 709, Springer-Verlag, 1993, pp. 246–252.

[62] K. L. Clarkson, Las Vegas algorithms for linear and integer programming, *J. ACM*, 42 (1995), 488–499.

[63] K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S.-H. Teng, Approximating center points with iterated Radon points, *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993, pp. 91–98.

[64] K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.*, 4 (1989), 387–421.

[65] E. Cohen and N. Megiddo, Maximizing concave functions in fixed dimension, in: *Complexity in Numeric Computation* (P. Pardalos, ed.), World Scientific, Singapore, 1993.

[66] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, *J. ACM*, 34 (1987), 200–208.

[67] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi, An optimal-time algorithm for slope selection, *SIAM J. Comput.*, 18 (1989), 792–810.

[68] R. Cole, M. Sharir, and C. K. Yap, On k-hulls and related problems, *SIAM J. Comput.*, 16 (1987), 61–77.

[69] G. E. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, *Proc. 2nd GI Conference on Automata Theory and Formal Languages*, *Lecture Notes in Computer Science*, Vol. 33, Springer-Verlag, 1975, pp. 134–183.

[70] L. Danzer, B. Grünbaum, and V. Klee, Helly's theorem and its relatives, in: *Convexity*, *Proc. Symp. Pure Math.*, Vol. 7, Amer. Math. Soc., Providence, 1963, pp. 101–180.

[71] G. Das and D. Joseph, The complexity of minimum convex nested polyhedra, *Proc. 2nd Canad. Conf. Comput. Geom.*, 1990, pp. 296–301.

[72] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid, Static and dynamic algorithms for k-point clustering problems, *J. Algorithms*, 19 (1995), 474–503.

[73] M. de Berg, O. Devillers, M. van Kreveld, O. Schwarzkopf, and M. Teillaud, Computing the maximum overlap of two convex polygons under translation, *Proc. 7th Annu. Internat. Sympos. Algorithms Comput.*, 1996.

[74] L. De Floriani, A graph based approach to object feature recognition, *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, 1987, pp. 100–109.

[75] M. DeHaemer and M. Zyda, Simplification of objects rendered by polygonal approximations, *Computers and Graphics*, 15 (1992), 175–184.

[76] X. Deng, An optimal parallel algorithm for linear programming in the plane, *Inform. Process. Lett.*, 35 (1990), 213–217.

[77] M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu, A randomized algorithm for slope selection, *Internat. J. Comput. Geom. Appl.*, 2 (1992), 1–27.

[78] Z. Drezner, On a modified 1-center problem, *Manage. Sci.*, 27 (1981), 838–851.

[79] Z. Drezner, The p-centre problems — Heuristic and optimal algorithms, *J. Oper. Res. Soc.*, 35 (1984), 741–748.

[80] Z. Drezner, The planar two-center and two-median problem, *Transp. Sci.*, 18 (1984), 351–361.

[81] Z. Drezner, On the rectangular p-center problem, *Naval Res. Logist. Q.*, 34 (1987), 229–234.

[82] Z. Drezner, Conditional p-centre problems, *Transp. Sci.*, 23 (1989), 51–53.

[83] Z. Drezner, ed., *Facility Location*, Springer-Verlag, New York, 1995.

[84] Z. Drezner, A. Mehrez, and G. O. Wesolowsky, The facility location problems with limited distances, *Transp. Sci.*, 25 (1992), 183–187.

[85] M. E. Dyer, Linear time algorithms for two- and three-variable linear programs, *SIAM J. Comput.*, 13 (1984), 31–45.

[86] M. E. Dyer, On a multidimensional search technique and its application to the Euclidean one-centre problem, *SIAM J. Comput.*, 15 (1986), 725–738.

[87] M. E. Dyer, A class of convex programs with applications to computational geometry, *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 1992, pp. 9–15.

[88] M. E. Dyer, A parallel algorithm for linear programming in fixed dimension, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 345–349.

[89] M. E. Dyer and A. M. Frieze, A simple heuristic for the p-centre problem, *Oper. Res. Lett.*, 3 (1985), 285–288.

[90] M. E. Dyer and A. M. Frieze, A randomized algorithm for fixed-dimension linear programming, *Math. Program.*, 44 (1989), 203–212.

[91] H. Ebara, N. Fukuyama, H. Nakano, and Y. Nakanishi, Roundness algorithms using the Voronoi diagrams, *Abstracts 1st Canad. Conf. Comput. Geom.*, 1989, p. 41.

[92] H. Edelsbrunner, Computing the extreme distances between two convex polygons, *J. Algorithms*, 6 (1985), 213–224.

[93] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[94] H. Edelsbrunner and R. Waupotitsch, Computing a ham-sandwich cut in two dimensions, *J. Symbolic Comput.*, 2 (1986), 171–178.

[95] A. Efrat and M. Sharir, A near-linear algorithm for the planar segment center problem, *Discrete Comput. Geom.*, 16 (1996), in press.

[96] A. Efrat, M. Sharir, and A. Ziv, Computing the smallest k-enclosing circle and related problems, *Comput. Geom. Theory Appl.*, 4 (1994), 119–136.

[97] M. Eisner and D. Severance, Mathematical techniques for efficient record segmentation in large shared databases, *J. ACM*, 23 (1976), 619–635.

[98] D. Eppstein, Dynamic three-dimensional linear programming, *ORSA J. Comput.*, 4 (1992), 360–368.

[99] D. Eppstein, Faster construction of planar two-centers, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997.

[100] D. Eppstein and J. Erickson, Iterated nearest neighbors and finding minimal polytopes, *Discrete Comput. Geom.*, 11 (1994), 321–350.

[101] T. Feder and D. H. Greene, Optimal algorithms for approximate clustering, *Proc. 20th Annu. ACM Sympos. Theory Comput.*, 1988, pp. 434–444.

[102] F. Follert, E. Schömer, and J. Sellen, Subquadratic algorithms for the weighted maximin facility location problem, *Proc. 7th Canad. Conf. Comput. Geom.*, 1995, pp. 1–6.

[103] F. Follert, E. Schömer, J. Sellen, M. Smid, and C. Thiel, Computing a largest empty anchored cylinder, and related problems, *Proc. 15th Conf. Foundations of Software Technology and Theoretical Comput. Sci., Lecture Notes in Computer Science*, Vol. 1026, Springer-Verlag, 1995, pp. 428–442.

[104] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, Optimal packing and covering in the plane are NP-complete, *Inform. Process. Lett.*, 12 (1981), 133–137.

[105] G. N. Frederickson, Optimal algorithms for tree partitioning, *Proc. 2nd ACM-SIAM Symp. Discr. Algo.*, 1991, pp. 168–177.

[106] G. N. Frederickson and D. B. Johnson, The complexity of selection and ranking in $X+Y$ and matrices with sorted rows and columns, *J. Comput. Syst. Sci.*, 24 (1982), 197–208.

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

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

[109] B. Gärtner, A subexponential algorithm for abstract optimization problems, *SIAM J. Comput.*, 24 (1995), 1018–1035.

[110] B. Gärtner and E. Welzl, Linear programming — Randomized and abstract frameworks, *Proc. 13th Sympos. Theoret. Aspects Comput. Sci.*, *Lecture Notes in Computer Science*, Vol. 1046, Springer-Verlag, 1996, pp. 669–687.

[111] A. Glozman, K. Kedem, and G. Shpitalnik, On some geometric selection and optimization problems via sorted matrices, *Proc. 4th Workshop Algorithms Data Struct.*, *Lecture Notes in Computer Science*, Vol. 955, Springer-Verlag, 1995, pp. 26–37.

[112] M. Goldwasser, A survey of linear programming in randomized subexponential time, *ACM-SIGACT News*, 26 (1995), 96–104.

[113] T. Gonzalez, Clustering to minimize the maximum intercluster distance, *Theoret. Comput. Sci.*, 38 (1985), 293–306.

[114] T. Gonzalez, Covering a set of points in multidimensional space, *Inform. Process. Lett.*, 40 (1991), 181–188.

[115] M. T. Goodrich, Geometric partitioning made easier, even in parallel, *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993, pp. 73–82.

[116] M. T. Goodrich, Efficient piecewise-linear function approximation using the uniform metric, *Discrete Comput. Geom.*, 14 (1995), 445–462.

[117] M. T. Goodrich, Fixed-dimensional parallel linear programming via relative epsilon-approximations, *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, 1996, pp. 132–141.

[118] B. Grünbaum, A proof of Vázsonyi's conjecture, *Bull. Research Council Israel, Section A*, 6 (1956), 77–78.

[119] P. Gupta, R. Janardan, and M. Smid, Fast algorithms for collision and proximity problems involving moving geometric objects, Report MPI-I-94-113, Max-Planck-Institut Inform., Saarbrücken, Germany, 1994.

[120] D. Gusfield, K. Balasubramanian, and D. Naor, Parametric optimization of sequence alignment, *Algorithmica*, 12 (1994), 312–326.

[121] R. Hassin and N. Megiddo, Approximation algorithms for hitting objects by straight lines, *Discrete Appl. Math.*, 30 (1991), 29–42.

[122] P. S. Heckbert and M. Garland, Fast polygonal approximation of terrains and height fields, Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[123] E. Helly, Über Systeme von abgeschlossenen Mengen mit gemeinschaftlichen Punkten, *Monaths. Math. und Physik*, 37 (1930), 281–302.

[124] A. Heppes, Beweis einer Vermutung von A. Vázsonyi, *Acta Math. Acad. Sci. Hungar.*, 7 (1956), 463–466.

[125] J. Hershberger, Minimizing the sum of diameters efficiently, *Comput. Geom. Theory Appl.*, 2 (1992), 111–118.

[126] J. Hershberger, A faster algorithm for the two-center decision problem, *Inform. Process. Lett.*, 47 (1993), 23–29.

[127] J. Hershberger and S. Suri, Finding tailored partitions, *J. Algorithms*, 12 (1991), 431–463.

[128] J. Hershberger and S. Suri, Efficient computation of Euclidean shortest paths in the plane, *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.*, 1993, pp. 508–517.

[129] J. Hershberger and S. Suri, Matrix searching with the shortest path metric, *Proc. 25th Annu. ACM Sympos. Theory Comput.*, 1993, pp. 485–494.

[130] D. S. Hochbaum and W. Maass, Approximation schemes for covering and packing problems in image processing and VLSI, *J. ACM*, 31 (1984), 130–136.

[131] D. S. Hochbaum and W. Maass, Fast approximation algorithms for a nonconvex covering problem, *J. Algorithms*, 8 (1987), 305–323.

[132] D. S. Hochbaum and D. Shmoys, A best possible heuristic for the k-center problem, *Math. Oper. Res.*, 10 (1985), 180–184.

[133] D. S. Hochbaum and D. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *J. ACM*, 33 (1986), 533–550.

[134] R. Hocken, J. Raja, and U. Babu, Sampling issues in coordinate metrology, *Manufacturing Review*, 6 (1993), 282–294.

[135] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, Piecewise smooth surface reconstruction, *Proc. SIGGRAPH 94*, 1994, pp. 295–302.

[136] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, Mesh optimization, *Proc. SIGGRAPH 93*, 1993, pp. 19–26.

[137] M. E. Houle, H. Imai, K. Imai, and J.-M. Robert, Weighted orthogonal linear L_∞-approximation and applications, *Proc. 1st Workshop Algorithms Data Struct.*, *Lecture Notes in Computer Science*, Vol. 382, Springer-Verlag, 1989, pp. 183–191.

[138] M. E. Houle and G. T. Toussaint, Computing the width of a set, *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-10 (1988), 761–765.

[139] D. P. Huttenlocher and K. Kedem, Computing the minimum Hausdorff distance for point sets under translation, *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, 1990, pp. 340–349.

[140] D. P. Huttenlocher, K. Kedem, and M. Sharir, The upper envelope of Voronoi surfaces and its applications, *Discrete Comput. Geom.*, 9 (1993), 267–291.

[141] R. Z. Hwang, R. C. Chang, and R. C. T. Lee, The generalized searching over separators strategy to solve some NP-Hard problems in subexponential time, *Algorithmica*, 9 (1993), 398–423.

[142] R. Z. Hwang, R. C. T. Lee, and R. C. Chang, The slab dividing approach to solve the Euclidean p-center problem, *Algorithmica*, 9 (1993), 1–22.

[143] H. Imai, D. Lee, and C. Yang, 1-segment center covering problems, *ORSA J. Comput.*, 4 (1992), 426–434.

[144] S. Jadhav and A. Mukhopadhyay, Computing a centerpoint of a finite planar set of points in linear time, *Discrete Comput. Geom.*, 12 (1994), 291–312.

[145] S. Jadhav, A. Mukhopadhyay, and B. Bhattacharya, An optimal algorithm for the intersection radius of a set of convex polygons, *J. Algorithms*, 20 (1996), 244–267.

[146] J. W. Jaromczyk and M. Kowaluk, An efficient algorithm for the Euclidean two-center problem, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 303–311.

[147] J. W. Jaromczyk and M. Kowaluk, A geometric proof of the combinatorial bounds for the number of optimal solutions to the 2-center Euclidean problem, *Proc. 7th Canad. Conf. Comput. Geom.*, 1995, pp. 19–24.

[148] J. W. Jaromczyk and M. Kowaluk, The two-line center problem from a polar view: A new algorithm and data structure, *Proc. 4th Workshop Algorithms Data Struct.*, *Lecture Notes in Computer Science*, Vol. 955, Springer-Verlag, 1995, pp. 13–25.

[149] G. Kalai, A subexponential randomized simplex algorithm, *Proc. 24th Annu. ACM Sympos. Theory Comput.*, 1992, pp. 475–482.

[150] N. Karmarkar, A new polynomial-time algorithm for linear programming, *Combinatorica*, 4 (1984), 373–395.

[151] M. J. Katz, Improved algorithms in geometric optimization via expanders, *Proc. 3rd Israel Symposium on Theory of Computing and Systems*, 1995, pp. 78–87.

[152] M. J. Katz and F. Nielsen, On piercing sets of objects, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 113–121.

[153] M. J. Katz and M. Sharir, An expander-based approach to geometric optimization, *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993, pp. 198–207.

[154] M. J. Katz and M. Sharir, Optimal slope selection via expanders, *Inform. Process. Lett.*, 47 (1993), 115–122.

[155] L. G. Khachiyan, Polynomial algorithm in linear programming, *U.S.S.R. Comput. Math. and Math. Phys.*, 20 (1980), 53–72.

[156] D. E. Knuth, *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[157] M. T. Ko and Y. T. Ching, Linear time algorithms for the weighted tailored 2-partition problem and the weighted rectilinear 2-center problem under L_∞-distance, *Discrete Appl. Math.*, 40 (1992), 397–410.

[158] M. T. Ko and R. C. T. Lee, On weighted rectilinear 2-center and 3-center problems, *Inform. Sci.*, 54 (1991), 169–190.

[159] M. T. Ko, R. C. T. Lee, and J. S. Chang, An optimal approximation algorithm for the rectilinear m-center problem, *Algorithmica*, 5 (1990), 341–352.

[160] N. M. Korneenko and H. Martini, Hyperplane approximation and related topics, in: *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), *Algorithms and Combinatorics*, Vol. 10, Springer-Verlag, Heidelberg, 1993, pp. 135–161.

[161] U. M. Landau, Estimation of circular arc and its radius, *Comput. Vision, Graphics, and Image Process*, 38 (1987), 317–326.

[162] V. B. Le and D. T. Lee, Out-of-roundness problem revisited, *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-13 (1991), 217–223.

[163] D. T. Lee and Y. F. Wu, Geometric complexity of some location problems, *Algorithmica*, 1 (1986), 193–211.

[164] D. Leven and M. Sharir, On the number of critical free contacts of a convex polygonal object moving in two-dimensional polygonal space, *Discrete Comput. Geom.*, 2 (1987), 255–270.

[165] C.-Y. Lo, J. Matoušek, and W. L. Steiger, Algorithms for ham-sandwich cuts, *Discrete Comput. Geom.*, 11 (1994), 433–452.

[166] C.-Y. Lo and W. Steiger, An optimal-time algorithm for ham-sandwich cuts in the plane, *Proc. 2nd Canad. Conf. Comput. Geom.*, 1990, pp. 5–9.

[167] W. Maass, On the complexity of nonconvex covering, *SIAM J. Comput.*, 15 (1986), 453–467.

[168] P. Magillo and L. De Floriani, Maintaining multiple levels of detail in the overlay of hierarchical subdivisions, *Proc. 8th Canad. Conf. Comput. Geom.*, 1996, pp. 190–195.

[169] J. Matoušek, Computing the center of planar point sets, in: *Computational Geometry: papers from the DIMACS special year* (J. E. Goodman, R. Pollack, and W. Steiger, eds.), Amer. Math. Soc., Providence, 1991, pp. 221–230.

[170] J. Matoušek, Randomized optimal algorithm for slope selection, *Inform. Process. Lett.*, 39 (1991), 183–187.

[171] J. Matoušek, Efficient partition trees, *Discrete Comput. Geom.*, 8 (1992), 315–334.

[172] J. Matoušek, Linear optimization queries, *J. Algorithms*, 14 (1993), 432–448.

[173] J. Matoušek, Lower bound for a subexponential optimization algorithm, *Random Structures & Algorithms*, 5 (1994), 591–607.

[174] J. Matoušek, On enclosing k points by a circle, *Inform. Process. Lett.*, 53 (1995), 217–221.

[175] J. Matoušek, On geometric optimization with few violated constraints, *Discrete Comput. Geom.*, 14 (1995), 365–384.

[176] J. Matoušek, D. M. Mount, and N. S. Netanyahu, Efficient randomized algorithms for the repeated median line estimator, *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, 1993, pp. 74–82.

[177] J. Matoušek and O. Schwarzkopf, A deterministic algorithm for the three-dimensional diameter problem, *Comput. Geom. Theory Appl.*, 6 (1996), 253–262.

[178] J. Matoušek, M. Sharir, and E. Welzl, A subexponential bound for linear programming, *Algorithmica*, 16 (1996), 498–516.

[179] N. Megiddo, Combinatorial optimization with rational objective functions, *Math. Oper. Res.*, 4 (1979), 414–424.

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

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

[182] N. Megiddo, The weighted Euclidean 1-center problem, *Math. Oper. Res.*, 8 (1983), 498–504.

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

[184] N. Megiddo, Partitioning with two lines in the plane, *J. Algorithms*, 6 (1985), 430–433.

[185] N. Megiddo, On the ball spanned by balls, *Discrete Comput. Geom.*, 4 (1989), 605–610.

[186] N. Megiddo, On the complexity of some geometric problems in unbounded dimension, *J. Symbolic Comput.*, 10 (1990), 327–334.

[187] N. Megiddo and K. J. Supowit, On the complexity of some common geometric location problems, *SIAM J. Comput.*, 13 (1984), 182–196.

[188] N. Megiddo and A. Tamir, On the complexity of locating linear facilities in the plane, *Oper. Res. Lett.*, 1 (1982), 194–197.

[189] N. Megiddo and E. Zemel, A randomized $O(n \log n)$ algorithm for the weighted Euclidean 1-center problem, *J. Algorithms*, 7 (1986), 358–368.

[190] J. S. B. Mitchell, Shortest paths among obstacles in the plane, *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993, pp. 308–317.

[191] J. S. B. Mitchell, Shortest paths and networks, Technical Report, State University of New York at Stony Brook, 1996.

[192] J. S. B. Mitchell and S. Suri, Separation and approximation of polyhedral objects, *Comput. Geom. Theory Appl.*, 5 (1995), 95–114.

[193] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, NY, 1995.

[194] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[195] N. Naor and M. Sharir, Computing a point in the center of a point set in three dimensions, *Proc. 2nd Canad. Conf. Comput. Geom.*, 1990, pp. 10–13.

[196] C. H. Norton, S. A. Plotkin, and É. Tardos, Using separation algorithms in fixed dimensions, *J. Algorithms*, 13 (1992), 79–98.

[197] C. H. Papadimitriou, Worst-case and probabilistic analysis of a geometric location problem, *SIAM J. Comput.*, 10 (1981), 542–557.

[198] M. Pellegrini, Ray shooting on triangles in 3-space, *Algorithmica*, 9 (1993), 471–494.

[199] M. Pellegrini, On collision-free placements of simplices and the closest pair of lines in 3-space, *SIAM J. on Computing*, 23 (1994), 133–153.

[200] M. Pellegrini, Repetitive hidden surface removal for polyhedra, *J. Algorithms*, 21 (1996), 80–101.

[201] M. J. Post, Minimum spanning ellipsoids, *Proc. 16th Annu. ACM Sympos. Theory Comput.*, 1984, pp. 108–116.

[202] E. Ramos, Intersection of unit-balls and diameter of a point set in R^3, *Computat. Geom. Theory Appl.*, 6 (1996), in press.

[203] M. Reichling, On the detection of a common intersection of k convex objects in the plane, *Inform. Process. Lett.*, 29 (1988), 25–29.

[204] M. Reichling, On the detection of a common intersection of k convex polyhedra, in: *Computational Geometry and its Applications*, *Lecture Notes in Computer Science*, Vol. 333, Springer-Verlag, 1988, pp. 180–186.

[205] J. H. Reif and J. A. Storer, A single-exponential upper bound for finding shortest paths in three dimensions, *J. ACM*, 41 (1994), 1013–1019.

[206] U. Roy and X. Zhang, Establishment of a pair of concentric circles with the minimum radial separation for assessing roundness error, *Computer Aided Design*, 24 (1992), 161–168.

[207] J. Salowe, L_∞ interdistance selection by parametric search, *Inform. Process. Lett.*, 30 (1989), 9–14.

[208] N. Sarnak and R. E. Tarjan, Planar point location using persistent search trees, *Commun. ACM*, 29 (1986), 669–679.

[209] E. Schömer, J. Sellen, M. Teichmann, and C. Yap, Efficient algorithms for the smallest enclosing cylinder problem, *Proc. 8th Canad. Conf. Comput. Geom.*, 1996, pp. 264–269.

[210] E. Schömer and C. Thiel, Efficient collision detection for moving polyhedra, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 51–60.

[211] R. Seidel, Small-dimensional linear programming and convex hulls made easy, *Discrete Comput. Geom.*, 6 (1991), 423–434.

[212] R. Seidel, Backwards analysis of randomized geometric algorithms, in: *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), Springer-Verlag, Heidelberg, Germany, 1993, pp. 37–68.

[213] S. Sen, Parallel multidimensional search using approximation algorithms: with applications to linear-programming and related problems, *Proc. 8th ACM Sympos. Paral. Algorithms and Architectures*, 1996, pp. 251–260.

[214] L. Shafer and W. Steiger, Randomizing optimal geometric algorithms, *Proc. 5th Canad. Conf. Comput. Geom.*, 1993, pp. 133–138.

[215] M. Sharir, A near-linear algorithm for the planar 2-center problem, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 106–112.

[216] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[217] M. Sharir and S. Toledo, Extremal polygon containment problems, *Comput. Geom. Theory Appl.*, 4 (1994), 99–118.

[218] M. Sharir and E. Welzl, A combinatorial bound for linear programming and related problems, *Proc. 9th Sympos. Theoret. Aspects Comput. Sci.*, *Lecture Notes in Computer Science*, Vol. 577, Springer-Verlag, 1992, pp. 569–579.

[219] M. Sharir and E. Welzl, Rectilinear and polygonal p-piercing and p-center problems, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 122–132.

[220] D. M. H. Sommerville, *Analytical Geometry in Three Dimensions*, Cambridge University Press, Cambridge, 1951.

[221] A. Stein and M. Werman, Finding the repeated median regression line, *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 1992, pp. 409–413.

[222] A. Stein and M. Werman, Robust statistics in shape fitting, *Proc. IEEE Internat. Conf. Comput. Vision Pattern. Recogn.*, 1992, pp. 540–546.

[223] K. Swanson, D. T. Lee, and V. L. Wu, An optimal algorithm for roundness determination on convex polygons, *Comput. Geom. Theory Appl.*, 5 (1995), 225–235.

[224] S. M. Thomas and Y. T. Chen, A simple approach for the estimation of circular arc and its radius, *Comput. Vision, Graphics, and Image Process*, 45 (1989), 362–370.

[225] S. Toledo, *Extremal Polygon Containment Problems and Other Issues in Parametric Searching*, M.S. Thesis, Dept. Comput. Sci., Tel Aviv Univ., Tel Aviv, 1991.

[226] S. Toledo, Approximate parametric search, *Inform. Process. Lett.*, 47 (1993), 1–4.

[227] S. Toledo, Maximizing non-linear concave functions in fixed dimension, in: *Complexity in Numerical Computations* (P. M. Pardalos, ed.), World Scientific, Singapore, 1993, pp. 429–447.

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

[229] K. R. Varadarajan, Approximating monotone polygonal curves using the uniform metric, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 311–318.

[230] K. R. Varadarajan and P. K. Agarwal, Linear approximation of simple objects, *Proc. 7th Canad. Conf. Comput. Geom.*, 1995, pp. 13–18.

[231] H. Voelcker, Current perspective on tolerancing and metrology, *Manufacturing Review*, 6 (1993), 258–268.

[232] E. Welzl, Smallest enclosing disks (balls and ellipsoids), in: *New Results and New Trends in Computer Science* (H. Maurer, ed.), *Lecture Notes in Computer Science*, Vol. 555, Springer-Verlag, 1991, pp. 359–370.

[233] G. Wesolowsky, The Weber problem: History and perspective, *Location Science*, 1 (1993), 5–23.

[234] A. C. Yao and F. F. Yao, A general approach to D-dimensional geometric queries, *Proc. 17th Annu. ACM Sympos. Theory Comput.*, 1985, pp. 163–168.

[235] E. Zemel, A linear time randomizing algorithm for searching ranked functions, *Algorithmica*, 2 (1987), 81–90.

Appendix: Multidimensional Parametric Searching

In this appendix we describe how to extend the parametric searching technique to higher dimensions. Suppose we have a d-variate (strictly) concave function $F(\lambda)$, where λ varies over $\mathbb{R}^d$. We wish to compute the point $\lambda^* \in \mathbb{R}^d$ at which $F(\lambda)$ attains its maximum value. Let A_s be, as above, an algorithm that can compute $F(\lambda_0)$ for any given λ_0. As in the parametric searching, we assume that the control flow of A_s is governed by comparisons, each of which amounts to computing the sign of a d-variate polynomial $p(\lambda)$ of a constant maximum degree. We also need a few additional assumptions on A_s. We call a variable in A_s *dynamic* if its value depends on λ. The only operations allowed on dynamic variables are: (i) evaluating a polynomial $p(\lambda)$ of degree at most δ, where δ is a constant, and assigning the value to a dynamic variable, (ii) adding two dynamic variables, and (iii) multiplying a dynamic variable with a constant. These assumptions imply that if λ is indeterminant, then each dynamic variable is a polynomial in λ of degree at most δ; and that F is a piecewise polynomial, each piece being a polynomial of degree at most δ.

We run A_s generically at λ^*. Each comparison involving λ now amounts to evaluating the sign of a d-variate polynomial $p(\lambda_1, \ldots, \lambda_d)$ at λ^*.

First consider the case where p is a linear function of the form $a_0 + \sum_{1 \le i \le d} a_i \lambda_i$, such that $a_d \neq 0$. Consider the hyperplane $h : \lambda_d = -(a_0 + \sum_{i=1}^{d-1} a_i \lambda_i)/a_d$. It suffices to describe an algorithm for computing the point $\lambda_h^* \in h$ such that $F(\lambda_h^*) = \max_{\lambda \in h} F(\lambda)$. By invoking this algorithm on h and two other hyperplanes h_{ε^+} and h_{ε^-}, where

$$h_{\varepsilon^+} : \lambda_d = -(a_0 + \varepsilon + \sum_{i=1}^{d-1} a_i \lambda_i)/a_d \quad \text{and} \quad \lambda_d = -(a_0 - \varepsilon + \sum_{i=1}^{d-1} a_i \lambda_i)/a_d$$

for some arbitrarily small constant ε, we can determine whether $\lambda^* \in h$, $\lambda^* \in h^+$, or $\lambda^* \in h^-$, where h^+ and h^- are the two open halfspaces bounded by h. (Technically, one can, and should, treat ε as an infinitesimal quantity; see [183] for details. Also, a similar perturbation scheme works when $a_d = 0$.) We solve the following more general problem: Let g be a k-flat in $\mathbb{R}^d$ contained in a $(k+1)$-flat η, and let $g^+ \subset \eta$ (resp. $g^- \subset \eta$) be the halfspace of η lying above (resp. below) g, relative to a direction in η orthogonal to g. We wish to compute the point $\lambda_g^* \in g$ such that $F(\lambda_g^*) = \max_{\lambda \in g} F(\lambda)$. Denote by $A_s^{(k)}$ an algorithm for solving this problem. As above, by running $A_s^{(k)}$ on g and on two infinitesimally shifted copies of g within η, we can determine whether the point λ_η^* where F attains its maximum on η lies in g, in g^+, or in g^-. Notice that $A_s^0 = A_s$, and that $A_s^{(d-1)}$ is the algorithm for computing λ_h^*. Inductively, assume that we have an algorithm $A_s^{(k-1)}$ that can solve this problem for any $(k-1)$-dimensional flat. We run A_s generically at λ_g^*, where λ varies over g. Each comparison involves the determination of the side of a $(k-1)$-flat $g' \subset g$ that contains λ_g^*. Running $A_s^{(k-1)}$ on g' and on two other infinitesimally shifted copies, g'_{ε^+} and g'_{ε^-}, of g' within g, we can perform the desired location of λ_g^* with respect to g', and thereby resolve the comparison. When the simulation of A_s terminates, λ_g^* will be found.

The total running time of the algorithm $A_s^{(k)}$ is $O(T_s^{k+1})$. The details of this approach can be found in [17, 65, 172, 196]. If we also have a parallel algorithm A_p that evaluates $F(\lambda_0)$ in time T_p using P processors, then the running time of $A_s^{(k)}$ can be improved, as in the one-dimensional case, by executing A_p generically at λ_g^* in each recursive step. A parallel step, however, requires resolving P independent comparisons. The goal is therefore to resolve, by invoking $A_s^{(k-1)}$ a constant number of times, a fixed fraction of these P comparisons, where each comparison requires the location of λ_g^* with respect to a $(k-1)$-flat $g' \subset g$. Cohen and Megiddo [65] developed such a procedure that yields a $2^{O(d^2)} T_s (T_p \log P)^d$-time algorithm for computing λ^*; see also [183]. Agarwala and Fernández-Baca [21] extended Cole's improvement of Megiddo's parametric searching to multidimensional parametric searching, which improves the running time of the Cohen-Megiddo algorithm in some cases by a polylogarithmic factor. Agarwal et al. [17] showed that these procedures can be simplified and improved, using $(1/r)$-*cuttings*, to $d^{O(d)} T_s (T_p \log P)^d$.

Toledo [227] extended the above approach to resolving the signs of nonlinear polynomials, using Collins's cylindrical algebraic decomposition [69]. We describe his algorithm for

$d = 2$. That is, we want to compute the sign of a bivariate, constant-degree polynomial p at λ^*. Let $\vee p$ denote the set of roots of p. We compute Collins' cylindrical algebraic decomposition Π of $\mathbb{R}^2$ so that the sign of p is invariant within each cell of Π [34, 69]. Our aim is to determine the cell $\tau \in \Pi$ that contains λ^*, thereby determining the sign of p at λ^*.

The cells of Π are delimited by $O(1)$ y-vertical lines — each passing through a self-intersection point of $\vee p$ or through a point of vertical tangency of $\vee p$; see Figure 8. For each vertical line ℓ, we run the standard 1-dimensional parametric-searching procedure to determine which side of ℓ contains λ^*. If any of these substeps returns λ^*, we are done. Otherwise, we obtain a vertical strip σ that contains λ^*. We still have to search through the cells of Π within σ, which are stacked one above the other in the y-direction, to determine which of them contains λ^*. We note that the number of roots of p along any vertical line $\ell : x = x_0$ within σ is the same, that each root varies continuously with x_0, and that their relative y-order is the same for each vertical line. In other words, the roots of $\vee p$ in σ constitute a collection of disjoint, x-monotone arcs $\gamma_1, \ldots, \gamma_t$ whose endpoints lie on the boundary lines of σ. We can regard each γ_i as the graph of a univariate function $\gamma_i(x)$.

Next, for each γ_i, we determine whether λ^* lies below, above, or on γ_i. Let x^* be the x-coordinate of λ^*, and let ℓ be the vertical line $x = x^*$. If we knew x^*, we could have run A_s at each $\gamma_i \cap \ell$, and could have located λ^* with respect to γ_i, as desired. Since we do not know x^*, we execute the 1-dimensional parametric-searching algorithm generically, on the line ℓ, with the intention of simulating it at the unknown point $\lambda_i = \gamma_i \cap \ell$. This time, performing a comparison involves computing the sign of some bivariate, constant-degree polynomial g at λ_i (we prefer to treat g as a bivariate polynomial, although we could have eliminated one variable, by restricting λ to lie on γ_i). We compute the roots $r_1, \ldots, r_u$ of g that lie on γ_i, and set r_0 and r_{u+1} to be the left and right endpoints of γ_i, respectively. As above, we compute the index j so that λ^* lies in the vertical strip σ' bounded between r_j and r_{j+1}. Notice that the sign of g is the same for all points on γ_i within the strip σ', so we can now compute the sign of g at λ_i.

When the generic algorithm being simulated on λ_i terminates, it returns a constant-degree polynomial $F_i(x, y)$, corresponding to the value of F at λ_i (i.e., $F_i(\lambda_i) = F(\lambda_i)$), and a vertical strip $\sigma_i \subseteq \sigma$ that contains λ^*. Let $\rho_i(x) = F_i(x, \gamma_i(x))$. Let γ_i^+ (resp. γ_i^-) be the copy of γ_i translated by an infinitesimally small amount in the $(+y)$-direction (resp. $(-y)$-direction), i.e., $\gamma_i^+(x) = \gamma_i(x) + \varepsilon$ (resp. $\gamma_i^-(x) = \gamma_i(x) - \varepsilon$), where $\varepsilon > 0$ is an infinitesimal. We next simulate the algorithm at $\lambda_i^+ = \gamma_i^+ \cap \ell$ and $\lambda_i^- = \gamma_i^- \cap \ell$. We thus obtain two functions $\rho_i^+(x)$, $\rho_i^-(x)$ and two vertical strips σ_i^+, σ_i^-. Let $\hat{\sigma}_i = \sigma_i \cap \sigma_i^+ \cap \sigma_i^-$. We need to evaluate the signs of $\rho_i(x^*) - \rho_i^+(x^*)$ and $\rho_i(x^*) - \rho_i^-(x^*)$ to determine the location of λ^* with respect to γ_i (this is justified by the concavity of F). We compute the x-coordinates of the intersection points of (the graphs of) $\rho_i, \rho_i^+, \rho_i^-$ that lie inside $\hat{\sigma}_i$. Let $x_1 \leq x_2 \leq \cdots \leq x_s$ be these x-coordinates, and let x_0, x_{s+1} be the x-coordinates of the left and right boundaries of $\hat{\sigma}_i$, respectively. By running A_s on the vertical lines $x = x_j$, for $1 \leq j \leq s$, we determine

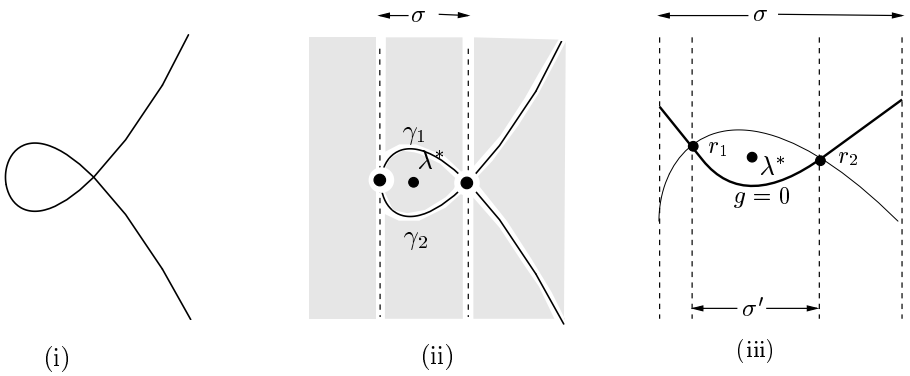

Figure 8: (i) roots of p; (ii) the cylindrical algebraic decomposition of p; (iii) the curves $g = 0$ and γ_1

the vertical strip $W_i = [x_j, x_{j+1}] \times \mathbb{R}$ that contains λ^*. Notice that the signs of polynomials $\rho_i(x) - \rho_i^+(x), \rho_i(x) - \rho_i^-(x)$ are fixed for all $x \in [x_j, x_{j+1}]$. By evaluating $\rho_i, \rho_i^+, \rho_i^-$ for any $x_0 \in [x_j, x_{j+1}]$, we can compute the signs of $\rho_i(x^*) - \rho_i^+(x^*)$ and of $\rho_i(x^*) - \rho^+(x^*)$.

Repeating this procedure for all γ_i's we can determine the cell of Π that contains λ^*, and thus resolve the comparison involving p. We then resume the execution of the generic algorithm.

The execution of the 1-dimensional procedure takes $O(T_s^2)$ steps, which implies that the generic simulation of the 1-dimensional procedure requires $O(T_s^3)$ time. The total time spent in resolving the sign of p at λ^* is therefore $O(T_s^3)$. Hence, the total running time of the 2-dimensional algorithm is $O(T_s^4)$. As above, using a parallel version of the algorithm for the generic simulation reduces the running time considerably. In d dimensions, the running time of Toledo's original algorithm is $O(T_s(T_p \log n)^{2^d-1})$, which can be improved to $T_s(T_p \log n)^{O(d^2)})$, using the result by Chazelle et al. [50] on vertical decomposition of arrangements of algebraic surfaces.