Randomized Algorithms for Geometric Optimization Problems*

Pankaj K. Agarwal † Sandeep Sen ‡

June 6, 2000

Abstract

This chapter reviews randomization algorithms developed in the last few years to solve a wide range of geometric optimization problems. We review a number of general techniques, including randomized binary search, randomized linear-programming algorithms, and random sampling. Next, we describe several applications of these techniques, including facility location, proximity problems, nearest neighbor searching, statistical estimators, and Euclidean TSP.

*Work by the first author was supported by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF grants EIA–9870724, EIA–997287, and CCR–9732787, and by a grant from the U.S.-Israeli Binational Science Foundation.

†Center for Geometric Computing, Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA. E-mail: `pankaj@cs.duke.edu`

‡Department of Computer Science and Engineering, IIT Delhi, New Delhi 110016, India. E-mail:`ssen@cse.iitd.ernet.in`

1 Introduction

Combinatorial optimization typically deals with problems of maximizing or minimizing a function of one or more variables subject to a large number of inequality 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 several applications, the underlying optimization problem involves a constant number of variables and a large number of constraints that 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 wide range of geometric optimization problems.

Since the seminal paper by Rabin [163], which initiated the study of randomization in developing fast algorithms, randomization has permeated in several areas, including algorithmic number theory, machine learning, distributed computing, and complexity theory. Even though one of the problems studied in Rabin's paper was the closest-pair problem, a central problem in computational geometry. randomization did not become popular in computational geometry until the late 1980s. In the mid 1980s Clarkson was developing his random-sampling technique, which he extended to a surprisingly general framework in his 1988 paper [46, 54]. Around the same time Haussler and Welzl [100] introduced the idea of ε-nets and VC-dimensions. These two techniques brought randomization to the forefront of computational geometry and revolutionized the field. Numerous randomized divide-and-conquer and incremental algorithms, dynamic data structures, and analysis techniques (e.g., backward analysis, Mulmuley's probabilistic games) have been developed in the last decade. Detailed accounts of these developments can be found in the books by Motwani and Raghavan [156] and Mulmuley [157] and in the survey papers [51, 139, 158, 175].

In this chapter we focus on the impact of randomization in geometric optimization. We review a number of randomized techniques that have been successfully applied to geometric optimization problems and review some of their applications. Like other areas, randomization has lead to simpler and improved algorithms for a wide range of geometric optimization problems.

We begin by discussing *randomized binary search*. Several randomized techniques have been developed for searching over the solution space, each step of which discards a fraction of the candidate values with probability at least $1/2$. This simple technique leads to fast and simple algorithms for many geometric optimization problems. Next, we review randomized algorithms for linear programming. Seidel [174], Dyer and Frieze [71], and Clarkson [50] proposed 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, some of these technique are rather general, and are also applicable to a variety of other geometric optimization problems. We discuss some of these techniques in Section 3. Another significant progress on linear programming was made in the beginning of the 1990s, when new randomized algorithms for linear programming were obtained independently by Kalai [116], and by Matoušek *et al.* [145, 181] (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.[1] This new technique is presented in Section 4. The algorithm in [145, 181] 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 4.

We also describe a randomized algorithm for set cover, based on the linear-programming algorithm by Clarkson [50]. Since many geometric optimization problems can be formulated as set-cover of hitting-set problems, many of these problems have benefited from this simple algorithm.

Next, we survey various geometric applications of these techniques and discuss a few additional problem-specific techniques. 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., nearest-neighbor searching and computing the diameter of a point set), *statistical estimators and metrology* (e.g., computing the smallest-width annulus that contains a given planar point set), and *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).

Although the common theme of most of the applications reviewed here is that they can be solved efficiently using the general techniques described in the beginning, many of them require a problem-specific, and often fairly sophisticated, approach. For example, the heart of a typical randomized binary search is the design of an efficient algorithm for solving the appropriate problem-specific "decision procedure" (see below for details). We will discuss details of these solutions for some of the problems, but will omit them for most of the applications because of lack of space.

2 Randomized Binary Search

In this section we describe various techniques for performing a randomized binary search. A geometric optimization problem $\mathcal{P} = (\Sigma, \mathcal{W})$ can be described as follows. Σ is a family

[1]Recall that the polynomial-time algorithms by Khachiyan [121] and by Karmarkar [118] 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.

of "geometric" objects, and $\mathcal{W}$ is a cost function such that, for a finite subset $S \subseteq \Sigma$, $\mathcal{W}(S)$ gives the cost of the optimal solution for S. For example, for the Euclidean 1-center problem, $\mathcal{W}(S)$ is the radius of the smallest disk enclosing S; for the traveling sales person (TSP) problem, $\mathcal{W}(S)$ is the length of the shortest tour of S. Given a subset $S \subseteq \Sigma$, the problem $\mathcal{P}$ ask for computing the value of $\mathcal{W}(S)$.

A subset $A \subseteq S$ is called a *basis* $\mathcal{B}(S)$ of S if A is the smallest subset of S for which $\mathcal{W}(A) = \mathcal{W}(S)$. The size of a basis is constant for many geometric optimization problems. For example, for any given set S of points in the plane, there is a subset $A \subseteq S$ of size at most three so that the smallest disk enclosing S is the same as that of A. Therefore in this case, there is a basis of size at most three. For a given S, define

$$\Lambda = \{\mathcal{W}(A) \mid A \subset S \text{ and } |A| \leq |\mathcal{B}(S)|\}.$$

Let $\lambda^* = \mathcal{W}(S)$. By definition, $\lambda^* \in \Lambda$. We can, of course, compute the entire set Λ explicitly and choose λ^*. However, Λ is typically too large to be computed explicitly. We therefore want to search over Λ implicitly. We can associate a decision problem $\mathcal{D}_{\mathcal{P}}$ with the optimization problem $\mathcal{P}$, which, given a value λ, asks whether $\lambda < \lambda^*$, $\lambda = \lambda^*$, or $\lambda > \lambda^*$. Suppose we have an algorithm $\mathcal{A}$ for the decision problem that runs in time $T(n)$. The randomized binary search techniques use $\mathcal{A}$ to compute λ^* in time $O(T(n)\log^c n)$. In some applications even the decision algorithm $\mathcal{A}$ is randomized. We now describe a few techniques for performing an implicit binary search.

2.1 Randomized halving technique

The randomization halving technique is based on the following simple idea, which has been used in a number of problems, including selecting an item from an ordered set [156]. Suppose we know that $\lambda^\star$ lies in an 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 $\lambda^\star$ 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 $\lambda^\star$ 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 $I \cap \Lambda$.

We illustrate this technique by applying it to the so-called *slope-selection* problem. Given a set S of n points in the plane and an integer $k \leq \binom{n}{2}$, determine a line ℓ_k connecting two input points that has the kth smallest slope among all such segments. For $k = \lceil \binom{n}{2}/2 \rceil$, ℓ_k is called the *Theil-Sen estimator* of S. Using the duality transform [73], which maps a point $p = (a, b)$ to the line $p^* : y = -ax + b$ and a line $\ell : y = \alpha x + \beta$ to the point $\ell^* = (\alpha, \beta)$ (see Figure 1), we can formulate this problem 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

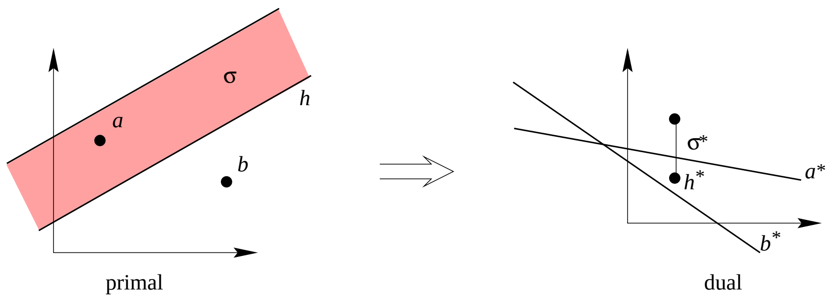

Figure 1: The duality transform in two dimensions.

an intersection point between two lines of L that has the kth 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 kth leftmost vertex of the *arrangement* $\mathcal{A}(L)$ of the lines in L;[2] see [11, 73, 179] for more details concerning arrangements. A complicated $O(n \log n)$ deterministic algorithm was developed by Cole *et al.* [55]; a simpler $O(n \log n)$-time deterministic algorithm, based on so-called *cuttings*, was later proposed by Brönnimann and Chazelle [34]. Here we present optimal randomized algorithms.

In this case Λ is the set of x-coordinates of the vertices of $\mathcal{A}(L)$. We describe the decision algorithm and show how it can be used to select a random element of $\Lambda \cap I$, for a given interval I. Given a vertical strip $W = (\alpha, \beta) \times \mathbb{R}$, let $L_\alpha = (\ell_1, \ell_2, \ldots, \ell_n)$ denote the sequence of lines in L sorted by their intercepts with $x = \alpha$, and let $L_\beta = (\ell_{\pi(1)}, \ell_{\pi(2)}, \ldots, \ell_{\pi(n)})$ denote the sequence of these lines sorted by their intercepts with $x = \beta$. An easy observation is that two lines ℓ_i, ℓ_j, with $i < j$, intersect inside W if and only if $\pi(i) > \pi(j)$. In other words, $|I \cap W|$ can be computed, in $O(n \log n)$ time, by counting the number of *inversions* in the permutation π, using the merge-sort procedure as follows [123]. We compute the sequence L_β and then sort it by the intercepts with $x = \alpha$ using the merge sort. We maintain a global counter C to count the number of inversions. Let $A = (a_1, \ldots, a_{n/2})$ be the sequence of the first half of L_β sorted by their intercepts along $x = \alpha$, and let $B = (b_1, \ldots, b_{n/2})$ be the second half of the lines in L_β sorted by their intercepts along $x = \beta$. Suppose that we have recursively computed A and B and that we have also counted the number of inversions (i, j) such that both i, j lie either in A or in B. The merge step merges A and B into a sorted sequence and counts the number of inversions of the form (b_j, a_i). The algorithm keeps track of the last element of B that was added to the overall sorted sequence. For $1 \le i \le n/2$, suppose $b_{k(i)}$ was the last element of B inserted in the sequence before adding a_i. Then $b_1, \ldots b_{k(i)} < a_i$, so (b_j, a_i), $1 \le j \le k(i)$, is an inversion. We therefore increment the counter C by $k(i)$ after inserting a_i. The overall running time of the algorithm is $O(n \log n)$.

[2]The arrangement of L, denoted as $\mathcal{A}(L)$, is the planar subdivision induced by L whose vertices are the intersection points of lines in L, edges are the portions of lines not containing any vertex of $\mathcal{A}(L)$, and whose faces are the maximal connected components of $\mathbb{R}^2 \setminus \bigcup L$.

The above algorithm can be extended to to generate a multiset of q random vertices of $\mathcal{A}(L) \cap W$ in time $O(n \log n + q)$ as follows. Using the above algorithm, we compute the number ν of vertices of $\mathcal{A}(L)$ lying in W. We then generate a random multiset Q of q integers in the range $[1, \nu]$, and sort it in $O(q)$ time using the bucket-sort algorithm. We run the inversion-counting algorithm once again, but with the following additional twist. Suppose C_i is the value of the counter C after we inserted a_i to the sorted sequence. If Q contains a value $j \in [C_{i-1}+1, C_i]$, we return the intersection point of a_i and $b_{j-C_{i-1}}$. The overall running time of the algorithm is $O(n \log n + q)$. Using this procedure, Matoušek [138] obtained the following simple slope-selection algorithm: Each step of the algorithm maintains a vertical strip $W(a,b) = \{(x,y) \mid a \le x \le b\}$ that is guaranteed to contain the kth 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 $k \le n$, the kth 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 kth 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 [156], that the expected running time of the above algorithm is $O(n \log n)$. This technique is quite general and has been used for many other problems.

Shafer and Steiger [177] gave a slightly different $O(n \log n)$ expected-time algorithm for the slope-selection problem. 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.* [55] 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 kth 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. Dillencourt *et al.* [63] proposed yet another randomized slope-selection algorithm.

2.2 Clarkson-Shor technique

Clarkson and Shor [54] gave another randomized technique for solving a variety of geometric optimization problems. Although they had originally proposed the algorithm for computing the diameter of a set of points in $\mathbb{R}^3$ (see Section 7.2), their approach works under the following more general setting. Let S be a set of objects, and let $\mu : S \to \mathbb{R}$ be a function such that $\mathcal{W}(S) = \min_{p \in S} \mu(p)$. For example, consider the closest-pair problem: Given a set S of n points in $\mathbb{R}^d$, find the minimum distance between a pair of distinct points of S. We set $\mathcal{W}(S) = \min_{p,q \in S, p \ne q} d(p,q)$. Define $\mu(p) = \min_{q \ne p} d(p,q)$. Then $\mathcal{W}(S) = \min_{p \in S} \mu(p)$.

We can then compute $\mathcal{W}(S)$ as shown in Figure 2.

```
function procedure OPTIMIZE (S);
    choose a random point p ∈ S;
    ⟨p_1, ..., p_n⟩:  a random permutation of S.
    curr = ∞
    for i = 1 to n do
      if μ(p_i) < curr
        curr = μ(p_i)        (⋆)
    return curr
```

Figure 2: Clarkson-Shor algorithm.

Note that Step ($\star$) is executed in the ith iteration only if $\mu(p_i) < \mu(p_j)$ for all $j < i$. Since we choose a random permutation of S,

$$\Pr[\mu(p_i) = \min_{1 \le j \le i} \mu(p_j)] = \frac{1}{i}.$$

Therefore, the expected number of times Step ($\star$) is executed is $O(\log n)$. The above algorithm is thus useful when, possibly after some preprocessing, deciding whether $\mu(p_i) < curr$ is easier than computing $\mu(p_i)$. Suppose for a given $curr$, S can be preprocessed in time $P(n)$ so that one can determine in $Q(n)$ time whether $\mu(p) < curr$, then the expected running time of the above algorithm is $O(P(n)\log n + nQ(n))$. For the 3D diameter problem, Clarkson and Shor showed that for a given r, S can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$ so that whether $\mu(p) < curr$ can be determined in $O(\log n)$ time, thereby attaining an $O(n \log^2 n)$ time algorithm. In Section 7.2, we will show that the expected running time can be further improved to $O(n \log n)$ using another observation.

Preprocessing S for answering a query of the form "is $\mu(p) < curr$?" is expensive in general. Agarwal and Sharir [9] combined the Clarkson-Shor technique with the random-sampling technique to solve a class of geometric optimization problems that can be formulated as computing a closest pair in some metric.

2.3 Chan's algorithm

Chan [38] extended the Clarkson-Shor technique to a more general framework. Suppose $\mathcal{P}$ is decomposable in the sense that we can decompose, in time $D(n)$, a given input S into subsets $S_1, \ldots, S_r$, for some integer $r > 0$, each of size at most αn so that $\mathcal{W}(S) = \min_{1 \le i \le r} \mathcal{W}(S_i)$. Further suppose given a real value $curr$, we can determine whether $\mathcal{W}(A) < curr$ in $D(|A|)$ time. Then we can use the Clarkson-Shor technique as follows. First compute the subproblems $S_1, \ldots, S_r$, and then processes these subproblems in a random order. For each subproblem S_i being processed, first determine whether $\mathcal{W}(S_i) < curr$ using the decision algorithm. If the answer is YES, then compute $\mathcal{W}(S_i)$ recursively. The expected number of subproblems for which the optimum value is recursively computed (to reset the current minimum) is only $O(\log r)$. Chan shows, and as we will see in the following sections, that in many instances the expected running time of this technique is $O(D(n))$.

3 Linear Programming

As noted in the introduction, several randomized algorithms have been developed for linear programming in the last decade [1, 71, 50, 174, 145]. These algorithms run in expected linear time for any fixed dimension. In this section we describe the algorithms by Seidel and Clarkson. In the next section we will discuss the algorithm by Matoušek *et al.* [145]. We are given a set $H = \{h_1, \ldots, h_n\}$ of n halfspaces in $\mathbb{R}^d$, called *constraints*, and a vector c called the *objective* function. We wish to compute an $x \in \mathbb{R}^d$ so that cx is minimized over the feasible region $K = \bigcap_{i=1}^n h_i$.

```
function procedure SEIDEL_lp(H, c)              /* H: n constraints in ℝ^d
    if n ≤ d then
        return Basis (H)                         /* returns v(H)
    else
        choose a random h ∈ H
        x := SEIDEL_lp(H \ {h})
        if x ∈ h
            return x
            else return SEIDEL_lp ({h ∩ g | g ∈ H \ {h}})      (⋆)
```

Figure 3: Seidel's randomized LP algorithm.

Seidel's algorithm is sketched in Figure 3. Given a set H of d constraints in general position, the function `Basis(H)` computes $x \in \bigcap H$ that minimizes cx. He proved that the probability of executing Step $(\star)$ is at most d/n. Since the recursive call in this step is an instance $(d-1)$-dimensional linear programming, the expected running time of the algorithm is $O(d^d n)$.

Dyer and Freize [71] gave another randomized algorithm based on the random-sampling technique with $O(d^d n)$ expected running time. Agarwal *et al.* [12] later extended this approach to solve a number of other optimization problems.

In the late 1980s an open question was whether there exists an algorithm whose running time is linear in n but the dependency on d is better than d^d. Clarkson developed two randomized algorithms with faster expected running time. His first algorithm is recursive and is summarized in Figure 4.

```
function procedure RECURSIVE_lp(H, c)                /* H: n constraints in ℝ^d
    if n ≤ 9d² then
        return SIMPLEX_lp (H)                          /* returns v(H)
    else
        V = ∅, r = d√n
        choose random R ∈ (H choose r)
        repeat
            x := RECURSIVE_lp(R ∪ V, c)
            V_x := {h ∈ H | x violates h}
            if |V_x| ≤ 2√n then
                V = V ∪ V_x          (⋆)
        until V = ∅
        return x
```

Figure 4: Clarkson's recursive LP algorithm.

We call an iteration *successful* if Step $(\star)$ is executed. It can be shown that an iteration is successful with probability at least $1/2$ and that there are at most $d+1$ successful iterations. Using these observations it can be shown that the expected running time of the algorithm is $O(d^2n+d^{d/2+\log\log n+O(1)})$. Chan [36] showed that **RECURSIVE_lp** algorithm can be modified to answer linear-programming queries. That is, preprocess a set H of n constraints, so that for a query objective function c, the point $x \in \bigcap H$ that minimizes cx can be computed efficiently. He sets $r = n/b$ where $b > 9d^2$ is a sufficiently large constant. Consequently, with probability at least $1/2$, $|V_x| \le \log n$. He preprocesses H and a family of random subsets of H for halfspace range-reporting queries so that V_x can be computed efficiently. He shows how to solve the recursive subproblem for $R \cup V$. For a parameter $n \le m \le n^{\lfloor d/2 \rfloor}$, H can be preprocessed in $m \log^{O(1)} n$ time into a data structure of $O(m)$ size so that a linear-programming query can be answered in $O((n/m^{1/\lfloor d/2 \rfloor}) \log^{2d+1} n)$ expected time. See the original paper for details. The query time was slightly improved by Ramos [166]. In particular, his algorithm can answer a query in time $n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)}$ using $O(n)$ space.

We now describe the second algorithm by Clarkson. 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 $|H| \le 6d^2$, 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 h 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 $\mu(V) \le 3\mu(H)/d$, we double the weight of each constraint in V; in any case, we repeat the sampling procedure. See Figure 5 for a pseudocode of the algorithm.

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

Figure 5: Clarkson's iterative 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 $O((d^2 n + d^{d+1}) \log n)$. By combining this algorithm with RECURSIVE_lp algorithm, i.e., using ITERATIVE_lp inside the loop of RECURSIVE_lp, the expected running time can be improved to $O(d^2 n) + d^{d/2+O(1)} \log n$.

Clarkson showed that ITERATIVE_lp can be extended to integer linear programming by choosing an appropriate value of the sample size and using Lenstra's integer-programming algorithm [131] for the base case. Let b denote the maximum number of bits required to specify the coefficients in a constraint or in the objective function. Then the algorithm performs $O(2^d dn + 8^d d\sqrt{n \log n} \log n) + d^{O(d)} b \log n$ expected number of operations, each involving at most $d^{O(1)} b$ bit numbers. See [117] for an earlier result on integer programming in fixed dimensions.

4 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) inscribed in convex polytopes in $\mathbb{R}^d$, computing the distance between polytopes in $\mathbb{R}^d$, general convex programming, and many other problems. Sharir and Welzl [181] and Matoušek *et al.* [145] 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, 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 [90] and Goldwasser [92] also survey the known results on LP-type problems. A dual version of the algorithm was independently obtained by Kalai [116], but only in the context of linear programming.

4.1 An abstract framework

Let us consider optimization problems specified by a pair (H, w), where H is a finite set, and $w : 2^H \to \mathcal{O}$ is a function into a linearly ordered set $(\mathcal{O}, \leq)$; we assume that $\mathcal{O}$ 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 $w(B_H) = w(H)$, 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: 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 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 6, 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 [145], one can show that basis-regular LP-type problems can be solved with an expected number of 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. A surprising feature of the LP-type problems is that some of them are instances of nonconvex programming. We will point out a few such instances below. Earlier algorithms for linear programming did not extend to nonconvex programming. Matoušek [140] 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.

```
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 6: A randomized algorithm for LP-type problems.

4.2 Linear programming

Returning to linear programming, let H be the given set of n constraints and c the objective function. We can assume that the objective vector is $c = (1, 0, 0, \ldots, 0)$. For a subset $G \subseteq H$, define $w(G)$ to be the lexicographically smallest point (vertex) of the intersection of halfspaces in G. 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 `ITERATIVE_lp` algorithm (given in Figure 5), 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 on the performance of `SUBEX_lp`, one should aim to exploit additional properties of linear programming if one wants to obtain a better bound on the performance of the algorithm for linear programming; this is still a major open problem.

4.3 Extensions

Matoušek [142] 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. For example, given a set H of n halfplanes in the plane and an integer $k \leq n$, the point with the smallest y-coordinate that lies in at least $n-k$ halfplanes can be computed in time $O(n \log k + k^3 \log^2 n)$ [142, 169]. For larger values of k, the running time is $O(n \log n + nk)$ [38]. Using the technique developed in Section 2.3, Chan [38] showed that if the intersection of the halfspaces is nonempty, the running time can be improved to $O(n + k(n/k)^{\varepsilon} \log n)$ for any $\varepsilon > 0$. Recently Gärtner and Welzl [91] have proved tail estimates on the running times of some of the LP-type problems. Chazelle and Matoušek [44] 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.

Amenta [17] considered 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 [17, 18] for more details and applications of this result.

5 Geometric Set Cover

Let $\Sigma = (X, \mathcal{R})$ be a set system, where X is a set of *objects* and $\mathcal{R}$, a family of subsets of X, is a set of *ranges*. A *set cover* C of Σ is a subset of $\mathcal{R}$ such that $X = \bigcup C$. The *set-cover* problem is to find a set cover of the smallest size. The *hitting-set* problem, which is dual of set cover, asks for computing a subset $H \subseteq X$ that intersects all ranges of $\mathcal{R}$. In a geometric setting, X is a set of geometric objects, e.g., points, lines, hyperplanes, spheres, etc., and $\mathcal{R}$ are geometric ranges. Let $\mathbb{H}$ be the set of all d-dimensional halfspaces. Here are two examples of geometric set systems: (i) $(\mathbb{R}^d, \mathbb{H})$, and (ii) $(\mathcal{H}, \{\{h \in \mathbb{H} \mid p \in h\} \mid p \in \mathbb{R}^d\}$.

It is known that the set-cover problem is NP-Hard even in geometric settings. For example, Fowler *et al.* [82] proved that it is NP-Hard to decide whether a given set of n points can be covered by k unit squares. The greedy algorithm can be used for computing an $O(\log n)$ approximation. However, one can do slightly better if the VC-dimension of the set system Σ is finite; see [100] for the definition of VC-dimension. Clarkson [48] modified the `ITERATIVE_lp` algorithm for computing a convex polytope of small complexity that lies between two nested convex polytopes, by reducing it to a geometric set-cover problem. Later Brönnimann and Goodrich [35] showed that Clarkson's algorithm works for any instance of set cover. For simplicity, we describe the algorithm for computing a hitting set. It performs a binary search on the size of the hitting set. At each stage, given an integer k, it either returns a hitting set of size $O(k \log k)$, or it returns that there is no hitting set of size at most k. Figure 7 summarizes the algorithm.

We call an iteration *successful* if $\mu(r) \leq \mu(X)/2k$ (see Figure 7 for the definition of μ). Following the same argument as for the linear-programming algorithm, Clarkson (and Brönniman and Goodrich) showed that an iteration is successful with probability at least

```
function procedure HITTING_SET(X, R);
    r := O(k log k);  successful := 0;
    μ(x) = 1 ∀x ∈ X
    repeat
        choose a random H ∈ (X choose r);
        if H ∩ r ≠ ∅ for all r ∈ R then
            return H
        choose a r ∈ R s.t.  r ∩ H = ∅
        if μ(r) ≤ μ(X)/2k then
            successful := successful + 1
            for all x ∈ r do μ(x) = 2μ(x)
    until successful := 12k log n
    return NO
```

Figure 7: A randomized hitting-set algorithm.

$1/2$ and that if there exists a hitting set of size at most k, then the algorithm returns a hitting set of size of $O(k \log k)$ within $12k \log n$ successful iterations. The only nontrivial steps in the algorithm are determining whether H intersects all ranges, choosing a range that does not intersect H, and updating the weights of objects. If each iteration can be performed in $T(n) \log n$ time, then the expected running time of the decision algorithm is $O(kT(n) \log n)$. If the set system is maintained implicitly, then $T(n)$ is proportional to $\sum_{r \in \mathcal{R}} |r|$. But $\mathcal{R}$ could be quite large and in many geometric settings it is defined explicitly. In such cases the main difficulty is to run the above algorithm efficiently without computing $\mathcal{R}$ explicitly. Such algorithms have been proposed in some cases [6, 8], and we will mention them below.

6 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 a given cost function $f(D, S)$ is minimized. A widely studied cost function, known as k-*center*, is the maximum distance between a demand point and its nearest supply object is minimized. That is, we minimize, over all possible appropriate sets S of supply objects, 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

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

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 $x \in \mathbb{R}^d$ is defined as $w_i \delta(d_i, x)$. The book by Drezner [67] describes many other variants of the facility-location problem. For a parameter $\varepsilon > 0$, a solution to the clustering problem is called an *ε-approximate* solution if its cost is at most $(1+\varepsilon)$ times that of an optimal solution. 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 $c \geq n/p$.

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, shape analysis [96, 146, 173], data compression and vector quantization [137], information retrieval [58, 59], drug design [81], and data mining [16, 33, 176].

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 ε-approximate solution is being sought (provided that ε is a sufficiently small constant) [80, 93, 126, 136, 149, 150]. 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, which can be solved using randomization, especially when p is a small constant.

6.1 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 randomized binary search or parametric searching [147]. 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.

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. The Euclidean 1-center problem is an LP-type problem, with combinatorial dimension $d+1$ [181, 186]. 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. The combinatorial dimension is $d+1$ because at most $d+1$ points are needed to determine the smallest enclosing ball. This problem, however, is not basis regular (the smallest enclosing ball may be determined by any number, between

2 and $d+1$, of points), and a naïve implementation of the basis-changing operation may be quite costly (in d). Nevertheless, Gärtner [89] 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 $O(d^2 n) + e^{O(\sqrt{d \log d})}$.

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 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 [141]. The best known deterministic algorithms are somewhat slower [62, 77, 74]. Matoušek [142] also showed that the smallest disk covering all but k points can be computed in time[3] $O(n \log n + k^3 n^{\varepsilon})$. Chan [38] presented a randomized algorithm for computing the discrete 1-center in $\mathbb{R}^3$ whose expected running time is $O(n \log n)$.

There are several other extensions of the smallest-enclosing-ball problem. They include (i) computing the smallest enclosing ellipsoid of a point set [44, 70, 161, 186], (ii) computing the largest ellipsoid (or ball) inscribed inside a convex polytope in $\mathbb{R}^d$ [89], (iii) computing a smallest ball that intersects (or contains) a given set of convex objects in $\mathbb{R}^d$ (see [148], 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 4. However, not all of them run in subexponential expected time because they are not all basis regular.

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 [65]. Matoušek [138] gave an algorithm with $O(n^2 \log^2 n)$ expected time by using the randomized halving technique described in Section 2.1. The running time of the decision algorithm was improved by Hershberger [103] to $O(n^2)$, which has been utilized in the best near-quadratic solution, by Jaromczyk and Kowaluk [113], which runs in $O(n^2 \log n)$ time; see also [114].

A major progress on this problem was made by Sharir [178], 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 [85]. Eppstein [76] simplified Sharir's algorithm, using randomization and better data structures, and obtained an improved solution, whose expected running time is $O(n \log^2 n)$. Halperin *et al.* [95] studied the 2-center problem amid obstacles. That is, given a set D of n demand points in $\mathbb{R}^2$ and a set O of pairwise disjoint simple polygons, called *obstacles*, with a total of m vertices, compute two supply points outside O so that the

[3] In this chapter, the meaning of complexity bounds that depend on an arbitrary parameter $\varepsilon > 0$, such as 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.

maximum distance between a point its nearest supply point is minimized. Following the same approach as in [76, 178] but using some novel data structures, they presented an $O(mn \log^2(mn) \log n)$ expected time randomized algorithm for this problem. They also presented an ε-approximation algorithm with $O((1/\varepsilon) \log(1/\varepsilon)(m + n \log n) \log(mn))$ expected running time.

Inaba *et al.* [106] have studied the problem of partitioning D into two clusters so that a function on the variance of points within each cluster is minimized. More precisely, let $\overline{x}(S)$ denote the centroid of a point set S. Then the variance of S, denoted as $\mathrm{Var}(S)$, is

$$\mathrm{Var}(S) = \frac{1}{|S|} \sum_{x_i \in S} \|x_i - \overline{x}(S)\|^2.$$

Define $\psi^\alpha(S) = |S|^\alpha \, \mathrm{Var}(S)$. Inaba *et al.* [106] define the cost of an optimal clustering to be $c(S) = \min \psi^\alpha(S_1) + \psi^\alpha(S_2)$, where the minimum is taken over all partitions S_1, S_2 of S. They consider $\alpha = 1, 2$. For a given $\varepsilon > 0$, they presented an $O(n/\varepsilon^2)$ expected time randomized algorithm for partitioning D into two clusters so that the cost of the clustering is at most $(1 + \varepsilon)c(S)$. Recently, their algorithm was derandomized by Matoušek [143]. See [108] for some recent results on 2-clustering in higher dimensions.

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-Complete if p is part of the input and $d \geq 2$, or if d is part of the input and $p \geq 3$ [82, 149]. Ko *et al.* [126] showed that computing a solution set S with $c(D, S) < 2r^*$, where r^* is the size of an optimal solution, is also NP-Complete.

The rectilinear 1-center problem is trivially solved in linear time. See [66, 124, 125, 149] for some earlier results. Sharir and Welzl [182] 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). This is an instance of nonconvex programming that is LP-type. Using the technique described in Section 2.3, Chan [38] proposed an $O(n \log n)$ expected time randomized algorithm for computing a rectilinear 5-center, which is optimal in the worst-case. Chan also presented an $O(n \log n)$ expected time algorithm for computing the smallest square that contains k of a given set of n points in the plane. See [120, 182] for additional related results.

6.2 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 [151] showed that the problem of determining whether $w^* = 0$ (i.e, D can be covered by p lines) is NP-Complete, which not only proves that the p-line-center problem is NP-Complete, but also proves that approximating w^* within a constant factor is NP-Complete.

Since even approximating w^* is NP-Complete, Agarwal and Procopiuc [8] developed an efficient algorithm for the case in which one approximates both w^* and p. In particular, let $w_p^* = w_p^*(D)$ denote the size of the Euclidean p-line center of S. By modifying the hitting-set algorithm described in Section 5, they presented a randomized algorithm that computes $O(p \log p)$ strips of width at most $6w_p^*$ that contain S. The expected running time of their algorithm is $O(np^2 \log^3 n \log(p \log n))$ provided that $p \leq \sqrt{n}$. They also extended their algorithm to higher dimensions in some cases. The main contribution of this result is that the expected running time of the algorithm is near linear as a function of n. In most practical applications, n is quite large and p is a small constant. One can, of course, use `HITTING_SET` algorithm to compute $O(p \log p)$ strips of width at most w_p^* in polynomial time, as the problem can be reduced to an instance of the hitting-set problem. If one maintains the underlying set system explicitly, the expected running time of the algorithm is $\Omega(n^3 p)$. Agarwal and Procopiuc showed that by maintaining the set system implicitly, the expected running time can be improved to roughly $n^{4/3}p^{4/3}$, but this algorithm is not practical. A few other approximation algorithms for this problem are given in [98].

6.3 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 $d \geq 2$ [150]. The special case of $d = 2, p = 1$ is the classical *Fermant-Weber problem*, and it goes back to the seventeenth century. It is known that the solution for the Fermant-Weber problem is unique and algebraic provided that all points of D are not collinear. Several numerical approaches have been proposed to compute an approximate solution. See [40, 187] for the history of the problem and for the known algorithms, and [159] for some heuristics for the p-median problem that work well for a set of random points. Recently, Arora *et al.* [23] described an ε-approximation algorithm for the p-median problem in the plane whose running time is $n^{O(1/\varepsilon)}$. For $d > 2$, the running time of their algorithm is $n^{O((\log n/\varepsilon)^{d-1})}$. Their algorithm is an extension of Arora's approximation algorithm for TSP, which we will describe in Section 10 below. The bound was later improved by Kolliopoulos and Rao [128], who proposed a Monte Carlo ε-approximation algorithm that runs in time $O(2^{1/\varepsilon^d} n \log n \log p)$ with probability at least 1/2.

There has also been much work on the p-median problem in arbitrary metric spaces. Lin and Vitter [133] (see also [134]) proposed a randomized algorithm, based on a randomized-rounding scheme, that, for any parameter $\varepsilon > 0$, computes $(1 + 1/\varepsilon)p$ clusters with a total cost of $2(1+\varepsilon)$ times that of an optimal solution. If we want to return exactly p clusters, Bartal [29] gave an $O(\log n \log\log n)$-approximation algorithm for the p-median problem, which was later improved by Charikar *et al.* [41] to $O(\log p \log\log p)$ using a randomized embedding technique. The best known approximation algorithm is by Charikar and Guha [42] that computes a 0.853-approximate solution in time. See also [43].

7 Proximity Problems

7.1 Nearest-neighbor searching

The *nearest-neighbor query* (*NN query*) problem, also known as the *post-office* problem [123], is defined as follows: Preprocess a set S of points in $\mathbb{R}^d$ into a data structure so that a point in S closest to a query point ξ can be reported quickly. This is one of the most widely studied problems not only in computational geometry but in several areas of computer science, including pattern recognition [57, 68], data compression [24, 164], information retrieval [79, 171], CAD [152], molecular biology [183], image analysis [127, 129], data mining [78, 99], machine learning [56], and geographic information systems [170, 184]. Most applications use so-called *feature vectors* to map a complex object to a point in high dimensions. Examples of feature vectors include color histograms, shape descriptors, Fourier vectors, and text descriptors.

For simplicity, we assume that the distance between points is measured in the Euclidean metric, though a more complicated metric can be used depending on the application. Dobkin and Lipton [64] described a *locus based* method that partitions the space into connected regions such that all points in a region have the same nearest neighbor among the n points. So the problem reduces to point-location for which the method required $n^{2^{O(d)}}$ space and $O(2^d \log n)$ query time. For $d = 2$, one can construct the Voronoi diagram of S and preprocess it for point-location queries in $O(n \log n)$ time using $O(n)$ space so that an NN query can be answered in $O(\log n)$ time [162]. For higher dimensions, Clarkson [47], presented a data structure of size $O(n^{\lceil d/2\rceil+\varepsilon})$, for any constant $\varepsilon > 0$, that can answer a query in $2^{O(d)} \log n$ time. The data structure can be constructed in $O(n^{\lceil d/2\rceil+\varepsilon})$ expected time. This paper was one of the earliest applications of random sampling in computational geometry. The query time can be improved to $O(d^5 \log n)$, using a technique of Meiser [153]. Note that the query time of the above approach is exponential in d, so it is impractical even for moderate values of d (say $d \approx 10$). This exponential dependence on dimension is called the *curse of dimension*. Several heuristics have been developed, especially in higher dimensions, which use practical data structures such as *kd*-trees, R-trees, R*-trees, and Hilbert R-trees; see e.g. [87, 104, 129, 127, 78, 99, 170, 184]. Even these algorithms suffer from the curse of dimensionality.

This has lead to the development of algorithms for finding approximate nearest neighbors

[25, 26, 27, 49, 122, 129] or for special cases, such as when the distribution of query points is known in advance [52, 188]. For a given parameter $\varepsilon > 0$ and a query point ξ, an ε-approximate nearest-neighbor query (ε-*NN query*) asks for returning a point $p \in S$ so that $d(p, \xi) \leq (1+\varepsilon)d(p', \xi)$ for all $p' \in S$. This relaxation is quite meaningful in the context of the applications mentioned above. Arya *et al.* [27] showed that an ε-NN query can be answered in $O((1/\varepsilon^d)\log n)$ time using $O(n)$ space. Note that the size of their data structure is independent of ε, and that ε can be specified as a part of the query. Although the query bound was later improved by Clarkson [49] and Chan [37] to $O((1/\varepsilon)^{(d-1)/2}\log n)$, ε is fixed for all queries in both the data structures and the size of their data structures depends on ε. Moreover, the data structure by Arya *et al.* [27] is practical and works well for dimensions up to $20--30$.

Many approximation techniques based on *distance preserving* random projections of points onto lower dimensional subspaces have been proposed, which result in randomized algorithms with query time polynomial in d and $\log n$ [112, 111, 122, 130]. Many of these techniques rely on the following classical result by Johnson and Lindenstrauss [115], which was subsequently improved and simplified in [61, 83, 84]: Any set of n points in d-dimensional Euclidean space $\mathbb{E}^d$ can be embedded in $O((1/\varepsilon^2)\log n)$ dimensions with at most ε relative error in the pairwise distances of S. Simpler proofs using elementary probabilistic techniques have been proposed [61, 111], which immediately give randomized polynomial-time algorithms for computing such an embedding.

Although distance preserving hashing had been used earlier for points in $\mathbb{R}^1$ [135] and for searching in higher dimensions [112], Kleinberg [122] was perhaps the first to exploit random projections in the context ε-NN searching. His algorithm relies on the following observation:

Lemma 7.1 ([122]) *Let $x, y \in \mathbb{R}^d$ be two vectors such that $(1+\gamma)\|x\| \leq \|y\|$ for some $\gamma \leq 1/2$. Then for a random unit vector v chosen uniformly over $\mathbb{S}^{d-1}$,*

$$\Pr[|v \cdot x| \geq |v \cdot y|] \leq \frac{1}{2} - \frac{\gamma}{3}.$$

This lemma implies that if the ratio of the lengths of two vectors is at least $(1+\gamma)$, then their random projections can be distinguished in a probabilistic sense. Based on this observation, an ε-NN query can be answered as follows. Choose a sufficiently large set of random unit vectors $V = \{v_1, v_2, \ldots v_L\} \subset \mathbb{S}^{d-1}$ and project the points in S on each of these vectors. For a point $a \in \mathbb{R}^d$, let $a^{(i)}$ denote the projection of a on v_i. For a point $q \in \mathbb{R}^d$, we say that a point a *dominates* b with respect to v_i if $|q^{(i)} - a^{(i)}| \leq |q^{(i)} - b^{(i)}|$, and that $a \prec_q b$ if a dominates b with respect to at least $L/2$ vectors in V. To answer an ε-NN query for a point q, we compute the point p in S that is minimal in the $\prec_q$-ordering. Kleinberg showed that S can be stored into a data structure of size $O((n/\varepsilon^2)^{2d}\log^{2d} n)$ so that for a query point q, the minimal point in the $\prec_q$-ordering can be computed in time $(d^2/\varepsilon^2)\log^{O(1)} n$. He also proposed another data structure of size $(nd/\varepsilon^2)\log^{O(1)} n$ that answers an ε-NN query in time $O((d/\varepsilon^2)\log d \log n \log(d\log n))$. These algorithms can report k ε-approximate nearest neighbors in additional $O(k)$ time.

Kleinberg's result was subsequently improved by Kushilevitz *et al.* [130]. Using a clustering argument, they showed that ε-NN searching in $\mathbb{E}^d$ can be reduced to ε'-NN searching on d-dimensional hypercube $\mathbb{Q}^d = \{0,1\}^d$, for some $\varepsilon' = \varepsilon/O(1)$, under the Hamming metric. (A similar reduction was used by Indyk and Motwani [111].) More precisely, Kushilevitz *et al.* [130] showed that an ε-NN data structure in $\mathbb{Q}^d$ using $S(n,d,\varepsilon)$ storage and $Q(n,d,\varepsilon)$ query time leads to an ε-NN structure in $\mathbb{E}^d$ using $O(n^2)S(n,\Delta,\varepsilon')$ storage and $((d\log n)/\varepsilon)^{O(1)} + \log n Q(n,\Delta,\varepsilon'))$ query time, where $\Delta = O((d/\varepsilon^8)\log^2(d/\varepsilon))$ and $\varepsilon' = \varepsilon/O(1)$. Their data structure for answering an ε-NN query in $\mathbb{Q}^d$ is based on the following observation. Let $H(\cdot,\cdot)$ denote Hamming distance.

Lemma 7.2 ([130]) *Let x, y be two points in $\mathbb{Q}^d$, let $\gamma > 0$ be a constant, let $0 \le \ell \le d$ be an integer, and let q be a point with $H(q,x) \le \ell$ and $H(q,y) > (1+\gamma)\ell$. Suppose we choose a random vector $r = (r^1, \ldots, r^d) \in \mathbb{Q}^d$ where $\Pr[r_i = 1] = 1/2\ell$, for each $1 \le i \le d$. For two vectors $u, v \in \mathbb{Q}^d$, let*

$$\tau_r(v) = \sum r^i \cdot v^i \pmod 2 \quad \textit{and} \quad \Delta(u,v) = \Pr[\tau_r(u) \ne \tau_r(v)].$$

Then there are constants $\delta, \delta_1 > 0$ depending only on γ such that $\Delta(q,x) \le \delta_1$ and $\Delta(q,y) > \delta + \delta_1$.

For a query point $q \in \mathbb{Q}^d$, we can now decide whether there exists a point S within distance ℓ from q, as follows. Choose a set $R = \{r_1, \ldots, r_t\}$ of $t = O((1/\varepsilon^2)\log(n\log d))$ of random vectors as described in the above lemma. Let a be a point in S. Let $\mu = \mu(a)$ be the number of vectors $r \in R$ for which $\tau_r(a) \ne \tau_r(q)$. Using Lemma 7.2 and Chernoff's bound, it can be shown that if $H(q,a) \le \ell$ then $\mu \le (\delta_1 + \delta/3)t$ with probability at least $1 - e^{-2\delta^2 t/9}$, and that if $H(q,a) \ge (1+\varepsilon)\ell$ then $\mu \ge (\delta_1 + 2\delta/3)$ with probability at least $1 - e^{-2\delta^2 t/9}$. We thus need to determine whether there exists a point $a \in S$ for which $\mu(a) \le (\delta_1 + \delta/3)t$. Based on this observation, Kushilevitz *et al.* showed that, with probability at least $1-\rho$, S can be preprocessed in a data structure of size $d(n\log d)^{O(1/\varepsilon^2)}$ so that a query can be answered in time $O((d^2/\varepsilon^2)\log(n\log d/\rho))$. Constructing this structure for every $0 \le \ell \le d$ and using the above procedure as the decision procedure in a binary search, we can answer an ε-NN query with probability at least $1-\rho$ in $O((d^2/\varepsilon^2)\log(n\log d/\rho)\log d)$ using $d^2(n\log d)^{O(1/\varepsilon^2)}$ storage.

Indyk and Motwani [111] proposed another randomized approach for answering ε-NN queries. They develop a data structure called *ring cover trees*, which lets them reduce the ε-NN searching problem to the ε-PLEB (ε-approximate point location in equal balls) defined as follows: Given a set S of n points in $\mathbb{R}^d$ and two parameters $\varepsilon, r > 0$, preprocess S into a data structure that for a query point q performs as follows: If there is a point

$p \in S$ so that $d(p,q) \leq r$, then it returns YES and a point p' with $d(p,p') \leq (1+\varepsilon)r$. If $d(p,q) > (1+\varepsilon)r$ for all $p \in S$, then it returns NO. Otherwise, it either returns NO or a point p with $d(p,q) \leq (1+\varepsilon)r$. They introduced the notion of *locality sensitive hashing* to answer ε-PLEB queries. For given parameters $r_1 > r_2 \geq 0$ and $1 > p_1 > p_2 > 0$, a family $\mathcal{H} = \{h : S \to U\}$ of hash functions is called (r_1, r_2, p_1, p_2)*-sensitive* if for every $p, q \in S$: (i) $d(p,q) \leq r_1$ implies that $\Pr_{\mathcal{H}}[h(p) = h(q)] \geq p_1$, and (ii) $d(p,q) > r_2$ implies that $\Pr_{\mathcal{H}}[h(p) = h(q)] \leq p_2$. Using such a family of hash functions, they answer an ε-PLEB query as follows. For simplicity, we will describe the data structure for points in $\mathbb{Q}^d$ under Hamming distance. Set $k = \log_{1/p_2} n$, $\rho = \log p_1 / \log p_2$, and $\ell = n^\rho$. Define a family

$$\mathcal{G} = \{(h_1, h_2, \ldots, h_k) : \mathbb{Q}^d \to U^k \mid h_1, h_2, \ldots, h_k \in \mathcal{H}\}.$$

Choose a random subset of ℓ functions $g_1, \ldots g_\ell \in \mathcal{G}$. Let T be a table of size $|U|^k$. For each $p \in S$ and $i \leq \ell$, we store p in the cell $g_i(p)$ of T, i.e., each point of S is stored in ℓ cells of T. Therefore only $O(n\ell) = O(n^{1+\rho})$ cells of T are nonempty. We can use the data structure by Fredman *et al.* [86] to store these nonempty entries in a table of size $O(n^{1+\rho})$ so that for a point q, we can access the cell of T corresponding to $g(q)$ in time $O(1)$ time after having computed the value of $g(q)$, which in turns requires evaluating k hash functions. For a query point $q \in \mathbb{Q}^d$, the algorithms proceed as follows. It accesses the cells $g_1(q), \ldots, g_\ell(q)$ of T, and checks whether any of the points stored in these cells is within distance $(1+\varepsilon)r$. If so, it returns such a point. If more than a total of 2ℓ points (including duplicates) are stored in these cells, the procedure checks at most 2ℓ points. It returns NO, if no point (among at most 2ℓ points) within distance $(1+\varepsilon)r$ from q was found. The correctness of this procedure follows from the following lemma:

Lemma 7.3 ([111]) *With probability at least* $1/2$, *for any* $p \in S$ *so that* $d(p,q) \leq r$, $g_j(p) = g_j(q)$ *for some* $j \leq \ell$, *and* $\sum_{j=1}^{\ell} |\{p \in S \mid d(p,q) > (1+\varepsilon)r \wedge g_j(p) = g_j(q)\}| < 2\ell$.

The size of the data structure is $O(n^{1+\rho})$ and a query requires evaluating ℓ hash functions. Andersson *et al.* [19] showed that for any $r, \varepsilon > 0$, the family of projection functions, $\mathcal{H} = \{h_i : h_i(x_1, \ldots x_d) = x_i \mid 1 \leq i \leq d\}$ is $(r, (1+\varepsilon)r, 1 - r/d, 1 - (1+\varepsilon)r/d)$-sensitive. Plugging these values, we obtain that an ε-PLEB query can be answered with probability at least $1/2$ in $O(dn^{1/(1+\varepsilon)})$ time using $O(n^{1+1/(1+\varepsilon)})$ space.

All the algorithms described above are Monte Carlo algorithms. Recently, Indyk [109] proposed a Las Vegas algorithm that answer an ε-NN query under L_1-metric in $((d/\varepsilon)\log n)^{O(1)}$ expected time using polynomial space. This also yields constant-factor approximation algorithms for ε-NN searching under L_2- and Hamming-metrics. See also [107, 110] for related results. The above data structures have recently been used to answer farthest neighbor queries, to compute the diameter of a point set, and for several other proximity problems. See [32, 97, 109] for details.

7.2 Diameter in $\mathbb{R}^3$

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 [94, 102], 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.) The technique described in Section 2.2 can compute the diameter in $\mathbb{R}^3$ in $O(n \log^2 n)$ expected time. Clarkson and Shor [54] showed that their technique can be refined so that the diameter can be computed in $O(n \log n)$ expected time. The basic observation is that the size of the problem can also be reduced in each step. Figure 8 gives an outline of the algorithm.

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

Figure 8: 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 $\delta(\cdot,\cdot)$ 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., [172]). Hence, each recursive step of the algorithm takes $O(|S| \log |S|)$ expected time. Since p is chosen randomly, $|S_1| = i$ with probability $1/n$, which implies that the expected running time of the overall algorithm is $O(n \log n)$. After several attempts, an $O(n \log n)$ deterministic algorithm was recently obtained by Ramos [165]. Using the recent techniques for nearest-neighbor searching, Monte Carlo ε-approximation algorithms have been developed for computing the diameter of a point set in $\mathbb{E}^d$ whose running time is subquadratic in n and polynomial in d [32, 108].

7.3 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 [89] 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 $\mathcal{P}_2$.

7.4 Selecting distances

Let S be a set of n points in the plane, and let $1 \le k \le \binom{n}{2}$ be an integer. We wish to compute the kth 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 $\sum_{p \in S} |D_r(p) \cap (S \setminus \{p\})|$, 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 $\le r$.) Agarwal *et al.* [4] gave a randomized algorithm, with $O(n^{4/3} \log^{4/3} n)$ expected time, for the decision problem, using the random-sampling technique of [54], which yields an $O(n^{4/3} \log^{8/3} n)$ expected-time algorithm for the distance-selection problem. Matoušek [138] showed that the randomized halving technique can be used to solve this problem in time $O(n^{4/3} \log^{5/3} n)$ time.

7.5 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.

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 Joseph [60] 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 [155] 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, where k_{OPT} is the minimum number of vertices in a convex polytope lying between P_1 and 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 [48] 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 [35] 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 [14] showed that this problem is NP-Hard. Agarwal and Desikan [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 [101] summarizes most of the known results on terrain simplification.

Instead of fixing ε and minimizing the size of the approximating surface, we can fix the size and ask for the best approximation. That 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. If the vertices of Q are required to be a subset of S, the best known algorithm is by Agarwal and Varadarajan [15]; it is based on the randomized halving technique described in Section 2.1, and its running time is $O(n^{4/3+\varepsilon})$.

8 Statistical Estimators and Related Problems

8.1 Line fitting

Fitting a line to a set $S = \{p_1, \ldots, p_n\}$ of n points in the plane is an important problem in statistical estimation. In order to cope with outliers, there has been much interest in defining *robust* line estimators whose slopes do not change much by a few outliers. One such estimator is the Theil-Sen estimator defined in Section 2.1 for which several optimal $O(n \log n)$ algorithms exist. Another commonly used estimators is the *repeated median* (RM) estimator, defined as follows. For each $p_i \in S$, let σ_i be the median of the slopes of the $n-1$ lines passing through p_i and another point of S, and let σ be the median of $\{\sigma_1, \ldots, \sigma_n\}$. Then the RM estimator of S is the line of slope σ passing through a pair of input points. Using a variant of the randomized halving technique described in Section 2.1 and some sophisticated range-searching data structures, Matoušek *et al.* [144] described an $O(n \log n)$ expected time algorithm for computing the RM estimator. They also described a somewhat simpler randomized algorithm with $O(n \log^2 n)$ expected time.

8.2 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 6.2. It can be shown that either one of the parallel planes determining the width contains a vertex and the other contains a face of $\mathrm{conv}(S)$, or each of the two parallel planes contains an edge of $\mathrm{conv}(S)$. Houle and Toussaint [105] gave an $O(n^2)$-time algorithm for computing the width in $\mathbb{R}^3$. They show that the first type of pairs of planes can be computed in $O(n \log n)$ time, but there could be $\Theta(n^2)$ *antipodal* pairs of convex hull edges. (A pair of edges e_1, e_2 of $\mathrm{conv}(S)$ is called antipodal if there exist two parallel planes π_1, π_2 supporting them such that S lies between π_1 and π_2.) The problem of computing a closest pair of antipodal edges can be reduced to a number of subproblems, each of which asks for 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' [11, 9]. Agarwal and Sharir [9] developed an $O(n^{3/2+\varepsilon})$ expected-time randomized algorithm for this problem, which implies that the width can also be computed in expected time $O(n^{3/2+\varepsilon})$.

8.3 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.* [72] observed that the center of a minimum-width annulus is a vertex of the closest-point Voronoi diagram of S, 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. Agarwal and Sharir [9] reduced this problem to computing a bichromatic closest pair in two given sets of lines in $\mathbb{R}^3$, under an appropriate distance function. Using the technique described in Section 2.2, they showed that such a closest pair can be computed in $O(n^{3/2+\varepsilon})$ expected time, which in turn implies that the minimum-width annulus can be computed within that time. If we know the angular ordering of points with respect to the center of the minimum-width annulus, which is the case in some of the applications, Ramos [88] showed that the problem becomes an LP-type problem and can therefore be solved in $O(n)$ expected time. Recently Chan [39] developed an approximation algorithm that using his linear-programming data structure (mentioned in Section 3) can compute in $O(n + 1/\varepsilon^{16/3} \log n)$ expected time an annulus containing S whose width is at most $(1+\varepsilon)$ times that of the thinnest annulus.

8.4 Center points

Let S be a set of n points in $\mathbb{R}^d$. Let h be a hyperplane, and let S^+, S^- be the subset of points lying in the positive and negative open halfspaces determined by h. For $0 \le \beta \le 1/2$, we say that h β-splits S if $\max\{|S^+|/|S|, |S^-|/|S|\} \le (1-\beta)$. A point $c \in \mathbb{R}^d$ is a *β-center* of S if any hyperplane containing c $(1-\beta)$-splits S. It is a known consequence of Helly's Theorem that a $1/(d+1)$-center always exists. Computing $1/(d+1)$-center is expensive in high dimensions, so approximation algorithms have been proposed. For a given parameter $0 < \delta < 1$, Clarkson *et al.* [53] gave a randomized algorithm that runs in $O((d\log(1/\delta))^{\log d})$ time and computes an $\Omega(1/d^2)$-center with probability at least $1-\delta$. By combining this approach with linear programming, they developed another algorithm that computes in $O(d/\varepsilon)^{O(d)}\log(1/\delta)$ time a $(1/(d+1)-\varepsilon)$-center with probability at least $1-\delta$.

8.5 Bucketing

Let $S = \{p_1, \dots, p_n\}$ be a set of n points in $\mathbb{R}^2$ and $1 \le b \le n$ an integer. We want to partition S into b equal-size buckets so that the maximum number of points in a bucket is minimized. We consider two types of buckets. First, we consider the case in which the buckets are strips. That is, we want to find $b+1$ equally spaced parallel lines so that all points of S lie between the extremal lines, the extreme lines contain at least one points of S, and the maximum number of points in a bucket is minimized; see Figure 9. We refer to this problem as the *uniform-projection* problem. If the lines have slope θ, we refer to these

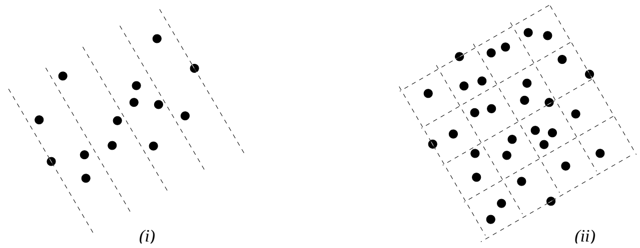

Figure 9: (i) Uniform-projection problem; (ii) two-dimensional partitioning problem.

buckets as the θ-cut of S. For each θ, there is unique θ-cut of S.

It is convenient to consider the problem in the dual plane (see Section 2.1 for the definition of the duality transform). Let ℓ_i denote the line dual to the point $p_i \in S$, and let $\mathcal{L} = \{\ell_i \mid 1 \le i \le n\}$. The dual of a strip σ bounded by two parallel lines ℓ_1 and ℓ_2 is the vertical segment $\sigma^* = \ell_1^*\ell_2^*$; a point p lies in σ if and only if the line σ^* intersects the segment σ^*. Let $\mathcal{A}(\mathcal{L})$ be the arrangement of $\mathcal{L}$. For a fixed x-coordinate θ, let $s(\theta)$ denote the vertical segment connecting the points on the upper and lower envelopes of $\mathcal{A}(L)$. The dual of the θ-cut is the partition of $s(\theta)$ into b equal subsegments. A point $p_i \in S$ lies in the jth bucket of the θ-cut if the dual line ℓ_i intersects the jth subsegment of $s(\theta)$.

The goal is therefore to compute θ so that the maximum number of lines intersecting a subsegment of $s(\theta)$ is minimum (over all θ). By sweeping the dual plane with a vertical line from $x = -\infty$ to $x = +\infty$ and maintaining the intersection points of $s(\theta)$ and L during this sweep, the optimal solution can be computed in a straightforward manner. Asano and Tokuyama [28] showed that such a sweep can be performed in $O(n^2)$ time. Agarwal *et al.* [5] developed a Monte Carlo algorithm that computes an optimal solution, with high probability, in subquadratic time in certain cases. They choose a random subset $R \subseteq L$ of lines and compute an optimal uniform projection for this subset. Using this solution, they compute a set of x-intervals that contains an optimal θ-cut with high probability and sweep a vertical line only through these x-intervals. Suppose the maximum number of points in a bucket of an optimal solution is $\frac{n}{b} + \Delta$. By choosing a subset R of appropriate size and by performing the sweep carefully, their algorithm computes an optimal solution in time $O(\min\{bn^{5/3} \log^{7/3} n + (b^2\Delta)n \log^3 n,\, n^2\})$, with probability at least $1 - 1/n$. In particular, the algorithm can detect and report whether there is a uniform projection (i.e., with $\Delta = 0$) in $O(\min\{bn^{5/3} \log^{7/3} n, n^2\})$ time.

The second type of buckets that one can consider are rectangular buckets. See Figure 9 (ii) for an example. A natural extension of the previous approach results in an algorithm that runs in time $O(\min\{b^{1/2}n^{5/3} \log^{7/3} n + (b^{3/2}\Delta)n \log^3 n,\, n^2\})$, with probability at least $1 - 1/n$, where the optimal value is $(n/b) + \Delta$.

9 Placement and Intersection

9.1 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? If the answer is yes, return a point in I, say the smallest point v^* in the lexicographical order. We assume that each polytope $P_i \in \mathcal{P}$ is preprocessed so that we can determine in $O(\log n)$ time whether a query point lies inside P_i. Let us call the query procedure `Feasibility`.

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. Note that there exist three polytopes $P_i, P_j, P_k \in \mathcal{P}$ so that v^* is the smallest vertex of $P_i \cap P_j \cap P_k$. Reichling [168] and Eppstein [75] showed that one can compute the smallest vertex of $P_i \cap P_j \cap P_k$ in $O(\log^3 n)$ time using `Feasibility` as a subroutine. Let us call this procedure as `Intersect`. Using `Intersect` and `Feasibility`, we can compute v^* as follows. For each triple $1 \le i, j, k \le m$, compute the smallest vertex v_{ijk} of $P_i \cap P_j \cap P_k$ if there exists one and check whether v_{ijk} lies inside all other polytopes of $\mathcal{P}$. Hence, computing v_{ijk} and checking whether $v_{ijk} \in I$ require $O(\log^3 n + m \log n)$ time. We then return the smallest vertex that lies inside I. The total time spent is $O(m^3 \log^3 n + m^4 \log n)$. Eppstein [75] presented a randomized recursive algorithm for computing v^*. At each step his algorithm solves a recursive subproblem for a subset of $\mathcal{P}$ with one less polytope, say P_i, and uses `Feasibility` to check whether the solution returned by the sub problem lies in P_i. At the base case, it invokes `Intersect`. Using randomization and the order in which his algorithm calls recursive subproblems, he proved that the expected number of times his algorithm executes `Intersect` and `Feasibility` procedures are $O(\sqrt{m}\log m)$ and $O(m\log m)$, respectively. Hence, the expected running time of his algorithm is $O(m\log m\log n + \sqrt{m}\log^3 n)$.

9.2 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 Delaunay triangulation induced by P within Q, Chew and Kedem [45] obtained an $O(m^4 n^2 2^{\alpha(n)}\log n)$-time algorithm. Faster algorithms can be developed using randomization and search parametric searching [3, 180]. The decision problem in this case can be defined as follows: 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. FP is the union of a collection of cells of an arrangement of $O(mn)$ *contact surfaces* in $\mathbb{R}^3$. Leven and Sharir [132] 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 [179] (it is almost linear in n for any fixed s). 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.

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 [13], and later refined in [2, 9]. 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 algorithm for the merge step 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.* [13] have shown that this problem can be reduced to the following: We are given a set S of points and a set Γ 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 Γ. Agarwal and Sharir [9] 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 $O(n^{3/2+\varepsilon})$.

Another related placement problem is the *penetration-depth* problem: Let P and Q be two polytopes in $\mathbb{R}^3$. The penetration depth of P and Q is the minimum distance by which Q has to be translated in a fixed direction so that P and Q become disjoint. Recently, Agarwal *et al.* [7] showed that the algorithm for computing the width can be used to compute the penetration depth of two convex polytopes in $\mathbb{R}^3$.

10 Network Design Problems

In this section, we review a randomized technique that has lead to approximation algorithms for several intractable network-design problems in a geometric setting, including Euclidean traveling salesperson, Euclidean Steiner tree, Euclidean k-MST, and Euclidean k-TSP. Until recently, it was not known whether polynomial-time approximation schemes (PTAS) exist for the Euclidean version of these problems even in the planar case (see [160, 22, 31, 189] for the previously best known approximation schemes). For the general problem (including the metric case), no polynomial-time ε-approximation algorithm can be obtained unless $P = NP$ ([22]). Recently it was shown that the some of these problems are MAX SNP-Hard even in the Euclidean setting if the dimension is part of the input [185]. In a significant breakthrough, Arora [20, 21] obtained an ε-approximate polynomial-time algorithms for the above problems in any fixed dimension. See also [154]. For simplicity, we describe his technique for Euclidean TSP in $\mathbb{R}^2$.

Let S be a set of n points in the plane, and let $\varepsilon > 0$ be a given parameter. The goal is to compute a tour of S (i.e., a cycle that visits every point of S exactly once) whose length is at most $(1+\varepsilon)\tau^*$, where τ^* is the length of an optimal tour. Since we are interested in an approximation algorithm, a well-known perturbation argument allows us to assume that the minimum distance between any two points is 8 and the maximum distance between any two points is $O(n/\varepsilon)$, and that the coordinates of points are in the interval $[0, L]$ for $L = O(n/\varepsilon)$ (see e.g. [21]). Let us assume that L is of the form 2^k for some integer k. Let B be the square $[0, 2L] \times [0, 2L]$. Choose two random integers $a, b \in [0, L]$ and translate each point of S by the vector (a, b). We will use S to denote the translated copy of S (i.e., $S = S + (a, b)$). Construct a quad tree Q on B for (the translated copy of) S, i.e., we recursively divide a square into four equal squares, starting from B, until the square contains at most one point; see Figure 10.

For two integers m, r, an *m-regular portal set* for Q is a set of points on the edges of each square σ of Q (at all levels) so that each corner of σ contains a portal and there are

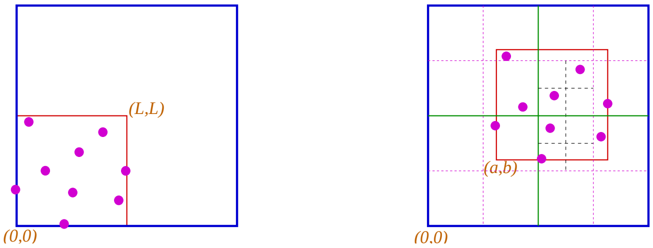

Figure 10: (i) Box B and original point set S; (ii) translated point set and the quad tree Q.

m other equally spaced points on each edge of σ. A tour π of S is called (m,r)-light if it crosses an edge of every square in Q only at portals and if it crosses an edge at most r times. The crux of Arora's algorithm lies in the following lemma.

Lemma 10.1 *Let S and Q be as above, and let $\varepsilon > 0$ be a parameter. With probability at least 1/2, there is an $((c/\varepsilon)\log n, c/\varepsilon)$-light tour of S of length at most $(1+\varepsilon)\tau^*$, where c is a constant.*

The proof of this lemma relies on two observations. Let π be a tour of S. First, if π intersects a line segment e of length λ at least three times, then there exists another tour of S of length at most $|\pi| + 3\lambda$ that intersects e at most twice. This argument was used by several heuristics for TSP in the past (see e.g. [30, 119]). Second, if we draw the integer grid inside B, then the number of intersections between π and the grid lines is at most $2|\pi|$. These lemmas and a simple probabilistic argument imply the above lemma.

The above lemma leads to a natural dynamic programming approach for computing an $((c/\varepsilon)\log n, c/\varepsilon)$-light tour π of S. With probability at least 1/2, thelength of π is at most $(1+\varepsilon)\tau^*$. With some care, the dynamic programming can be executed in time $n(\log n)^{O(1/\varepsilon)}$. Since the only randomization step in the above algorithm is choosing the vector (a,b), we can derandomize the algorithm by running the above algorithm for all values of $a, b \in [0, L]$. These algorithms extend to higher dimensions in straightforward manner. Rao and Smith [167] improved Arora's Monte Carlo algorithm by combining his dynamic programming approach with spanners of small overall weight. Their algorithm computes an ε-approximate tour in time $(\sqrt{d}/\varepsilon)^{O(d(\sqrt{d/\varepsilon})^{d-1})}n + O(dn\log n)$ with probability at least 1/2. Note that the running time of these algorithms is doubly exponential in d. In view of the hardness result by Trevisan [185], which shows that the Euclidean TSP problem is MAX SNP-hard in $\mathbb{R}^{\log n}$, this dependence is necessary unless NP has subexponential algorithms.

11 Discussion

In this chapter we reviewed several randomized techniques and algorithms for a wide range of geometric optimization problems. We mostly focussed on summarizing the known techniques and did not state open problems. There are, however, several interesting open problems in this area, e.g., nearest-neighbor searching and clustering algorithms in high dimensions, practical approximation algorithms for network-design problems, strongly polynomial-time algorithms for linear programming, faster algorithms for surface simplification etc. In the last few years several elegant techniques, e.g., Monte Carlo algorithms using random walks on expander graphs and randomized rounding in conjunction with semidefinite programming, have been developed for nongeometric problems. It would be interesting to explore whether geometric optimization problems can benefit from these techniques. Similarly, the recent work on hardness of approximation algorithms has not been applicable to geometric optimization problems in fixed dimensions.

Although we covered a variety of topics, we did not attempt to cover all applications of randomization in geometric optimization, as it would be an impossible task. Additional applications of randomization in geometric optimization include volume estimation of convex bodies [69], learning geometric concepts, and shape matching. Interested readers can find more material on geometric optimization in [10, 31, 90] and on randomized geometric algorithms in various chapters of this book as well as in [157, 139, 51].

References

[1] I. Adler and R. Shamir, A randomization scheme for speeding up algorithms for linear and convex quadratic programming problems with a high constraints-to-variables ratio, Technical Report 21-90, Rutgers Univ., New Brunswick, NJ, May 1990.

[2] P. K. Agarwal, B. Aronov, and M. Sharir, Computing envelopes in four dimensions with applications, *SIAM J. Comput.*, 26 (1997), 1714–1732.

[3] P. K. Agarwal, B. Aronov, and M. Sharir, Motion planning for a convex polygon in a polygonal environment, Tech. Report CS-1997-17, Department of Computer Science, Duke University, 1997.

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

[5] P. K. Agarwal, B. K. Bhattacharya, and S. Sen, Output-sensitive algorithms for uniform partitions of points, *Proc. 10th Intl. Sympos. Algorithms and Computation*, 1999.

[6] P. K. Agarwal and P. K. Desikan, An approximation algorithm for terrain simplification, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997, pp. 139–147.

[7] P. K. Agarwal, L. J. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir, Computing the penetration depth of two convex polytopes in 3d, *Proc. 7th Scandanavian Workshop on Algorithmic Theory*, 2000, to appear.

[8] P. K. Agarwal and C. M. Procopiuc, Approximation algorithms for projective clustering, *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, 2000, pp. 538–547.

[9] P. K. Agarwal and M. Sharir, Efficient randomized algorithms for some geometric optimization problems, *Discrete Comput. Geom.*, 16 (1996), 317–337.

[10] P. K. Agarwal and M. Sharir, Efficient algorithms for geometric optimization, *ACM Comput. Surv.*, 30 (1998), 412–458.

[11] P. K. Agarwal and M. Sharir, Arrangements and their applications, in: *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, eds.), Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000, pp. 49–119.

[12] 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.

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

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

[15] P. K. Agarwal and K. R. Varadarajan, Approximating monotone polygonal curves using the uniform metric, *Discrete Comput. Geom.*, 23 (2000), 273–291.

[16] R. Agrawal, A. Ghosh, T. Imielinski, B. Iyer, and A. Swami, An interval classifier for database mining applications, *Proc. of the 18th Conf. on Very Large Databases*, 1992, pp. 560–573.

[17] 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.

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

[19] A. Andersson, P. B. Miltersen, S. Riis, and M. Thorup, Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{\log n/\log\log n})$ is necessary and sufficient, *Proc. 37th Sympos. on Foundations of Computer Science*, 1996, pp. 441–450.

[20] 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.

[21] S. Arora, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, *J. ACM*, (1998), 753–782.

[22] S. Arora and C. Lund, Hardness of approximations, in: *Approximation Algorithms for NP-Hard Problems* (D. S. Hochbaum, ed.), PWS Publishing Company, Boston, MA, 1997, pp. 399–446.

[23] S. Arora, P. Raghavan, and S. Rao, Approximation schemes for Euclidean k-median and related problems, *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998, pp. 106–113.

[24] S. Arya and D. M. Mount, Algorithms for fast vector quantization, *Data Compression Conference*, IEEE Press, 1993, pp. 381–390.

[25] S. Arya and D. M. Mount, Approximate nearest neighbor queries in fixed dimensions, *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, 1993, pp. 271–280.

[26] S. Arya, D. M. Mount, and O. Narayan, Accounting for boundary effects in nearest-neighbor searching, *Discrete Comput. Geom.*, 16 (1996), 155–176.

[27] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, *J. ACM*, 45 (1998), 891–923.

[28] T. Asano and T. Tokuyama, Algorithms for projecting points to give the most uniform distribution with application to hashing, *Algorithmica*, 9 (1993), 572–590.

[29] Y. Bartal, Probabilistic approximation of metric spaces and its algorithmic applications, *Proc. 37th Sympos. on Foundations of Computer Science*, 1996, pp. 184–193.

[30] J. Beardwood, J. H. Halton, and J. M. Hammersley, The shortest path through many points, *Math. Proc. Camb. Phil. Soc.*, 55 (1959), 299–327.

[31] M. Bern and D. Eppstein, Approximation algorithms for geometric problems, in: *Approximation Algorithms for NP-Hard Problems* (D. S. Hochbaum, ed.), PWS Publishing Company, Boston, MA, 1997, pp. 296–345.

[32] A. Borodin, R. Ostrovsky, and Y. Rabani, Subquadratic approximation algorithms for clustering problems in high dimensional spaces, *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999, pp. 435–444.

[33] T. Brinkhoff and H.-P. Kriegel, The impact of global clusterings on spatial database systems, *Proc. of the International Conf. on Very Large Databases*, 1994, pp. 168–179.

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

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

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

[37] T. M. Chan, Approximate nearest neighbor queries revisited, *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, pp. 352–358.

[38] T. M. Chan, Geometric applications of a randomized optimization technique, *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, 1998, pp. 269–278.

[39] T. M. Chan, Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus, *Proc. 16th Sympos. Comput. Geom.*, 2000, p. to appear.

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

[41] M. Charikar, C. Chekuri, A. Goel, and S. Guha, Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median, *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998, pp. 114–123.

[42] M. Charikar and S. Guha, Improved combinatorial algorithms for facility location and k-median problems, *Proc. 40th Sympos. on Foundations of Computer Science*, 1999, pp. 378–388.

[43] M. Charikar, S. Guha, E. Tardos, and D. Shmoys, A constant-factor approximation algorithm for the k-median problem, *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999, pp. 1–10.

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

[45] 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.

[46] K. L. Clarkson, Applications of random sampling in computational geometry, II, *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, 1988, pp. 1–11.

[47] K. L. Clarkson, A randomized algorithm for closest-point queries, *SIAM J. Comput.*, 17 (1988), 830–847.

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

[49] K. L. Clarkson, An algorithm for approximate closest-point queries, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 160–164.

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

[51] K. L. Clarkson, Randomized geometric algorithms, in: *Computing in Euclidean Geometry* (D.-Z. Du and F. Hwang, eds.), *Lecture Notes Series on Computing*, Vol. 4, World Scientific, Singapore, 2nd edition, 1995, pp. 149–194.

[52] K. L. Clarkson, Nearest neighbor queries in metric spaces, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 609–617.

[53] K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S.-H. Teng, Approximating center points with iterative Radon points, *Internat. J. Comput. Geom. Appl.*, 6 (1996), 357–377.

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

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

[56] S. Cost and S. Salzberg, A weighted nearest neighbor algorithm for learning with symbolic features, *Machine Learning*, 10 (1993), 57–67.

[57] T. M. Cover and P. E. Hart, Nearest neighbor pattern classification, *IEEE Trans. Inform. Theory*, 13 (1967), 21–27.

[58] D. R. Cutting, D. R. Karger, and J. O. Pedersen, Constant interaction-time scatter/gather browsing of very large document collections, *Proc. 16th Annu. Internat. ACM SIGIR Conf. Research and Develop. in Inform. Retrieval*, 1993, pp. 126–134.

[59] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, Scatter/gather: A cluster-based approach to browsing large document collections, *Proc. 16th Annu. Internat. ACM SIGIR Conf. Research and Develop. in Inform. Retrieval*, 1992, pp. 318–329.

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

[61] S. Dasgupta and A. Gupta, An elementary proof of the Johnson-Lindenstrauss lemma, Tech. Rep. TR-99-06, Intl. Comput. Sci. Inst., Berkeley, CA, 1999.

[62] 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.

[63] 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.

[64] D. P. Dobkin and R. J. Lipton, Multidimensional searching problems, *SIAM J. Comput.*, 5 (1976), 181–186.

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

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

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

[68] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley-Interscience, New York, 1973.

[69] M. Dyer and A. Frieze, Computing the volume of convex bodies: A case where randomness provably helps, in: *Probabilistic Combinatorics and Its Applications* (B. Bollobás, ed.), American Mathematical Society, Providence, RI, 1991, pp. 123–169.

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

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

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

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

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

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

[76] D. Eppstein, Faster construction of planar two-centers, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997, pp. 131–138.

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

[78] C. Faloutsos and K.-I. Lin, FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia databases, *Proc. ACM SIGMOD Conf. on Management of Data*, 1995, pp. 163–173.

[79] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, Fast subsequence matching in time-series databases, *Proc. ACM SIGMOD Conf. on Management of Data*, 1994, pp. 86–93.

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

[81] P. Finn, L. E. Kavraki, J.-C. Latombe, R. Motwani, C. Shelton, S. Venkatasubramanian, and A. Yao, Rapid: randomized pharmacophore identification for drug design, *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, pp. 324–333.

[82] 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.

[83] P. Frankl and H. Maehara, The Johnson-Lindenstrauss lemmas and the sphericity of some graphs, *J. Combin. Theory, Ser. B*, 44 (1988), 355–362.

[84] P. Frankl and H. Maehara, Some geometric applications of the beta distribution, *Ann. Inst. Stat. Math*, 42 (1990), 463–474.

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

[86] M. L. Fredman, J. Komlos, and E. Szemeredi, Storing a sparse table with $o(1)$ worst case access time, *J. ACM*, 31 (1984), 538–544.

[87] J. H. Friedman, J. L. Bentley, and R. A. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Trans. Math. Softw.*, 3 (1977), 209–226.

[88] J. García-Lopez and P. Ramos, Fitting a set of points by a circle, *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, pp. 139–146.

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

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

[91] B. Gärtner and E. Welzl, Random sampling in geometric optimization: New insights and applications, *Proc. 16th Sympos. Comput. Geom.*, 2000, p. to appear.

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

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

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

[95] D. Halperin, M. Sharir, and K. Goldberg, The 2-center problem with obstacles, *Proc. 16th Sympos. Comput. Geom.*, 2000, p. to appear.

[96] K.-A. Han and S.-H. Myaeng, Image organization and retrieval with automatically constructed feature vectors, *Proc. 16th Annu. Internat. ACM SIGIR Conf. Research and Develop. in Inform. Retrieval* (H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, eds.), ACM Press, August 1996, pp. 157–165.

[97] S. Har-Peled and P. Indyk, When crossings count: Approximating the minimum spanning tree, *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, 2000. to appear.

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

[99] T. Hastie and R. Tibshirani, Discriminant adaptive nearest neighbor classification, *IEEE Trans. Pattern Anal. Mach. Intell.*, 18 (1996), 607–616.

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

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

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

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

[104] G. R. Hjaltason and H. Samet, Ranking in spatial databases, in: *Advances in Spatial Databases: Fourth International Symposium* (M. J. Egenhofer and J. R. Herring, eds.), *Lecture Notes Comput. Sci.*, Vol 951, 1995, pp. 83–95.

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

[106] M. Inaba, N. Katoh, and H. Imai, Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 332–339.

[107] P. Indyk, On approximate nearest neighbors in non-Euclidean spaces, *Proc. 39th Sympos. on Foundations of Computer Science*, 1998, pp. 148–155.

[108] P. Indyk, A sublinear-time approximation scheme for clustering in metric spaces, *Proc. 40th Sympos. on Foundations of Computer Science*, 1999, pp. 154–159.

[109] P. Indyk, Dimensionality reduction techniques for proximity problems, *Proc. 11th ACM-SIAM Sympos. on Discrete Algorithms*, 2000, pp. 371–378.

[110] P. Indyk and M. Farach-Colton, Approximate nearest neighbor algorithms for hausdorff metrics via embeddings, *Proc. 40th Sympos. on Foundations of Computer Science*, 1999, pp. 171–180.

[111] P. Indyk and R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998, pp. 604–613.

[112] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, Locality-preserving hashing in multidimensional spaces, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 618–625.

[113] 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.

[114] 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.

[115] W. Johnson and J. Lindenstrauss, Extensions of Lipschitz maps into a Hilbert space, *Contemp. Math.*, 26 (1984), 189–206.

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

[117] R. Kannan, Improved algorithms for integer programming and related lattice problems, *Proc. 15th Annu. ACM Sympos. Theory Comput.*, 1983, pp. 193–206.

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

[119] R. M. Karp, Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane, *Math. Oper. Res.*, 2 (1977), 209–224.

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

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

[122] J. Kleinberg, Two algorithms for nearest-neighbor search in high dimension, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 599–608.

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

[124] 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.

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

[126] 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.

[127] V. Koivune and S. Kassam, Nearest neighbor filters for multivariate data, *IEEE Workshop on Nonlinear Signal and Image Processing*, 1995.

[128] S. Kolliopoulos and S. Rao, A nearly linear-time approximation scheme for the Euclidean k-median problem, *Proc. 7th European Sympos. Algorithms*, 1999, pp. 378–389.

[129] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapa, Fast nearest neighbor search in medical image database, *Proc. 22nd VLDB Conference*, 1996, pp. 215–226.

[130] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, Efficient search for approximate nearest neighbor in high dimensional spaces, *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998, pp. 614–623.

[131] H. W. Lenstra, Integer programming with a fixed number of variables, *Math Oper. Research*, 8 (1983), 538–548.

[132] 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.

[133] J.-H. Lin and J. S. Vitter, Approximation algorithms for geometric median problems, *Inform. Process. Lett.*, 44 (1992), 245–249.

[134] J.-H. Lin and J. S. Vitter, ϵ-approximations with minimum packing constraint violation, *Proc. 24th Annu. ACM Sympos. Theory Comput.*, 1992, pp. 771–782.

[135] N. Linial and O. Sasson, Non-expansive hashing, *Proc. 28th Annu. ACM Sympos. Theory Comput.*, 1996, pp. 509–517.

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

[137] J. Makhoul, S. Roucos, and H. Gish, Vector quantization in speech coding, *Proc. IEEE*, 73 (1985), 1551–1588.

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

[139] J. Matoušek, Epsilon-nets and computational geometry, in: *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), Springer-Verlag, 1993, pp. 69–89.

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

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

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

[143] J. Matoušek, On approximate geometric k-clustering, to appear in *Discrete Comput. Geom.*, 2000.

[144] 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*, January 1993, pp. 74–82.

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

[146] C. Meghini, An image retrieval model based on classical logic, *Proc. 16th Annu. Internat. ACM SIGIR Conf. Research and Develop. in Inform. Retrieval*, 1995, pp. 300–308.

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

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

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

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

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

[152] H. Mehrotra and J. E. Gary, Feature-based retrieval of similar shapes, *Proc. 9th IEEE Intl. Conf. on Data Engineering*, 1996, pp. 108–115.

[153] S. Meiser, Point location in arrangements of hyperplanes, *Inform. Comput.*, 106 (1993), 286–303.

[154] J. S. B. Mitchell, A. Blum, P. Chalasani, and S. Vempala, A constant-factor approximation algorithm for the geometric k-MST problem in the plane, *SIAM J. Comput.*, 28 (1998), 771–781.

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

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

[157] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[158] K. Mulmuley, Randomized algorithms in computational geometry, in: *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, eds.), Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000, pp. 703–724.

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

[160] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.

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

[162] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[163] M. O. Rabin, Probabilistic algorithms, in: *Algorithms and Complexity: New Directions and Recent Results* (J. F. Traub, ed.), Academic Press, New York, NY, 1976, pp. 21–39.

[164] V. Ramasubramanian and K. K. Paliwal, Fast k-dimensional tree algorithms for nearest neighbor search with applications to vector quantization encoding, *IEEE Trans. Signal Processing*, 40 (1992), 518–531.

[165] E. A. Ramos, Deterministic algorithms for 3-D diameter and some 2-D lower envelopes, *Proc. 16th Sympos. Comput. Geom.*, 2000, p. to appear.

[166] E. A. Ramos, Linear programming queries revisited, *Proc. 16th Sympos. Comput. Geom.*, 2000, p. to appear.

[167] S. Rao and W. D. Smith, Improved approximation schemes for geometric graphs via "spanners" and "banyans", *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998. 540–550.

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

[169] T. Roos and P. Widmayer, k-violation linear programming, *Inform. Process. Lett.*, 52 (1994), 109–114.

[170] N. Roussopoulos, S. Kelley, and F. Vincent, Nearest neighbor queries, *Proc. ACM SIGMOD Conf. on Management of Data*, 1995, pp. 71–79.

[171] G. Salton, *Automatic Text Processing*, Addison-Wesley, Reading, MA, 1989.

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

[173] P. Schroeter and J. Bigün, Hierarchical image segmentation by multi-dimensional clustering and orientation-adaptive boundary refinement, *Pattern Recogn.*, 28 (1995), 695–709.

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

[175] 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.

[176] J. Shafer, R. Agrawal, and M. Mehta, Sprint: A scalable parallel classifier for data mining., *Proceedings of the International Conference on Very Large Databases*, Morgan Kauffman, 1996, pp. 544–555.

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

[178] M. Sharir, A near-linear algorithm for the planar 2-center problem, *Discrete Comput. Geom.*, 18 (1997), 125–134.

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

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

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

[182] 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.

[183] B. K. Shoichet, D. L. Bodian, and I. D. Kuntz, Molecular docking using shape descriptors, *J. Computational Chemistry*, 13 (1992), 380–397.

[184] R. F. Sproull, Refinements to nearest-neighbor searching, *Algorithmica*, 6 (1991), 579–589.

[185] L. Trevisan, When Hamming meets Euclid: The approximability of geometric TSP and MST, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 21–29.

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

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

[188] P. N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, 1993, pp. 311–321.

[189] A. Zelikovsky, Minimum base of weighted k-polymatroid and Steiner tree problem, Report MPI-I-92-121, Max-Planck-Institut Inform., Saarbrücken, Germany, 1992.