Translating a Planar Object to Maximize Point Containment*

Pankaj K. Agarwal† Torben Hagerup‡ Rahul Ray§
Micha Sharir¶ Michiel Smid‖ Emo Welzl**

Abstract

Let C be a compact set in $\mathbb{R}^2$ and let S be a set of n points in $\mathbb{R}^2$. We consider the problem of computing a translate of C that contains the maximum number, κ^*, of points of S. It is known that this problem can be solved in a time that is roughly quadratic in n. We show how random-sampling and bucketing techniques can be used to develop a near-linear-time Monte Carlo algorithm that computes a placement of

*Agarwal was supported by NSF grants ITR-333-1050, EIA-9870724, EIA-9972879, CCR-00-86013, and CCR-9732787, and by a grant from the U.S.-Israeli Binational Science Foundation. Sharir was supported by NSF Grants CCR-97-32101 and CCR-00-98246, by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing), by the Hermann Minkowski-MINERVA Center for Geometry at Tel Aviv University, and by a grant from the U.S.-Israeli Binational Science Foundation. Smid was supported by NSERC.

†Center for Geometric Computing and Department of Computer Science, Duke University, Durham, NC 27708-0129, U.S.A. E-mail: pankaj@cs.duke.edu.

‡Institut für Informatik, Universität Frankfurt, D-60054 Frankfurt am Main, Germany. E-mail: hagerup@ka.informatik.uni-frankfurt.de.

§Max-Planck-Institute for Computer Science, D-66123 Saarbrücken, Germany. E-mail: rahul@mpi-sb.mpg.de.

¶School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, U.S.A. E-mail: sharir@math.tau.ac.il.

‖School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6. E-mail: michiel@scs.carleton.ca.

**Institut für Theoretische Informatik, ETH Zürich, CH-8092 Zürich, Switzerland. E-mail: emo@inf.ethz.ch.

C containing at least $(1-\varepsilon)\kappa^*$ points of S, for given $\varepsilon > 0$, with high probability. We also present a deterministic algorithm that solves the ε-approximate version of the optimal-placement problem and runs in $O((n^{1+\delta} + n/\varepsilon)\log m)$ time, for arbitrary constant $\delta > 0$, if C is a convex m-gon.

1 Introduction

Let C be a compact set in $\mathbb{R}^2$ and let S be a set of n points in $\mathbb{R}^2$. We define the *optimal-placement* problem to be that of computing a point $t \in \mathbb{R}^2$ for which the translate $C + t$ of C contains the maximum number of points of S. Set

$$\kappa^*(C, S) = \max_{t \in \mathbb{R}^2} |S \cap (C + t)|.$$

Motivated by applications in clustering and pattern recognition (see [14, 16]), the optimal-placement problem has received much attention over the last two decades. Chazelle and Lee [7] presented an $O(n^2)$-time algorithm for the case in which C is a circular disk, and Eppstein and Erickson [11] proposed an $O(n \log n)$-time algorithm for rectangles. Efrat et al. [10] developed an algorithm for convex m-gons with a running time of $O(n\kappa^* \log n \log m + m)$, where $\kappa^* = \kappa^*(C, S)$, which was subsequently improved by Barequet et al. [3] to $O(n \log n + n\kappa^* \log(m\kappa^*) + m)$.

All the algorithms above, except the one for rectangles, require $\Omega(n^2)$ time in the worst case, i.e., when $\kappa^* = \Omega(n)$. This raises the question of whether a near-linear approximation algorithm exists for the optimal-placement problem. In this paper we answer the question in the affirmative by presenting Monte-Carlo and deterministic approximation algorithms.

We call a translate $C+t$ of C an *ε-approximate placement* if $|S \cap (C+t)| \geq (1-\varepsilon)\kappa^*(C, S)$, and an algorithm that produces an ε-approximate placement is called an *ε-approximation* algorithm. We define the *ε-optimal-placement* problem to be the one that asks for an ε-approximate placement.

We make the following assumptions about C and S:

(A1) The boundary of C, denoted by ∂C, is connected and consists of m edges, each of which is described by a polynomial of bounded degree. The endpoints of these edges are called the vertices of C. We will refer to C as a *disk*.

(A2) For all distinct $p, q \in S$, the boundaries of the two translates $C+p$ and $C+q$ of C intersect in at most s points, and the intersections can be computed in $I(m)$ time. By computing an intersection, we here mean determining the two edges, one of each translate, that are involved in the intersection. Moreover, for every point $x \in \mathbb{R}^2$, we can decide in $Q(m)$ time whether $x \in C$.

(A3) C is sandwiched between two axes-parallel rectangles whose widths and heights differ by factors of at most α and β, respectively, for positive integers α and β.

We call C *fat* if α and β in assumption (A3) can be chosen as constants. Schwarzkopf et al. [19] showed that after a suitable rotation, every compact convex set is fat with $\alpha = \beta = 2$.

We assume a model of computation in which the roots of a bounded-degree polynomial can be computed in $O(1)$ time. This implies that primitive operations on the edges of C, e.g., computing the intersection points between two edges of two translates $C+p$ and $C+q$, with $p, q \in S$, can be performed in $O(1)$ time.

In Section 2, we present two algorithms for the optimal-placement problem and show how bucketing can be used to expedite the running time, especially if C is fat. In particular, let $T(n)$ be the running time of an algorithm for the optimal-placement problem on a set of n points. Furthermore, let $T_{\mathrm{g}}(n)$ denote the time required to group the points of S according to the cells of an integer grid, i.e., stores S in a list such that for each grid cell B, all points of $S \cap B$ occur together in a contiguous sublist. Then the bucketing algorithm can compute an optimal placement in time $O(T_{\mathrm{g}}(n) + nT(\kappa^*)/\kappa^*)$, where $\kappa^* = \kappa^*(C, S)$, under suitable assumptions on T and T_{g} and if C is fat. Besides being interesting in its own right, this will be crucial for the approximation algorithms.

In Section 3, we show that using random sampling and possibly bucketing, we can transform any deterministic algorithm for the optimal-placement problem to a Monte-Carlo algorithm for the ε-optimal-placement problem. Given a parameter $\gamma > 0$, the first algorithm in Section 3, based on random-sampling, computes an ε-approximate placement in $O(n + T(\gamma n))$ time with error probability at most $sne^{-\varepsilon^2\gamma\kappa^*}$, under suitable assumptions on T. The second algorithm combines random sampling with bucketing and computes an ε-approximate placement in $O(T_{\mathrm{g}}(n) + nQ(m) + nT(\gamma\kappa^*)/\kappa^*)$ time with error probability at most $s\kappa^* e^{-\varepsilon^2\gamma\kappa^*}$, again under suitable assumptions on T

Time	Error probability	Reference
$n + (\gamma n)^2$	$ne^{-\gamma\kappa^*}$	Section 3.1
$n \log n$	$e^{-\sqrt{\kappa^* \log n}}$	Section 3.2
n	$ne^{-\sqrt{\kappa^*}}$	Section 3.2
$n^{1+\delta}$	0	Section 4

Table 1: Worst-case running times (constant factors omitted) and error probabilities of our ε-approximation algorithms if C is a circular disk; δ and ε are arbitrarily small positive constants and γ is an arbitrary positive real number.

---

and T_g and if C is fat. For circular disks and constant ε, the running time becomes $O(n \log n)$ and the error probability is at most $e^{-\sqrt{\kappa^* \log n}}$; see Table 1. If C is fat and $m = O(1)$, by combining two levels of random sampling with the bucketing technique, we can compute an ε-approximate placement in $O(n)$ time for constant ε with error probability at most $ne^{-\sqrt{\kappa^*}}$.

Finally, in Section 4, we present a deterministic algorithm, based on the cutting algorithm of Chazelle [6], that computes an ε-approximate placement for convex sets C. If C has $O(1)$ edges, the algorithm runs in $O(n^{1+\delta} + n/\varepsilon)$ time, for any given constant $\delta > 0$. If C is a convex m-gon, the running time is $O((n^{1+\delta} + n/\varepsilon) \log m)$.

2 Preliminaries and Exact Algorithms

Let C be a disk satisfying assumptions (A1)–(A3) and let S be a set of n points in $\mathbb{R}^2$. For simplicity, we assume that the origin lies inside C. For a point $p \in \mathbb{R}^2$, define $C_p = p - C = \{p - c \mid c \in C\}$. Let $\mathcal{C} = \{C_p \mid p \in S\}$. For a point set R and a point $t \in \mathbb{R}^2$, the *depth* of t with respect to R, denoted by $d_R(t)$, is the number of points $p \in R$ for which C_p contains t. This is the same as $|R \cap (C + t)|$, so that $\kappa^*(C, S) = \max_{t \in \mathbb{R}^2} d_S(t)$. Hence, the problem of computing an optimal placement reduces to computing a point of maximum depth with respect to S.

Consider the arrangement $\mathcal{A}(\mathcal{C})$ defined by the boundaries of the sets C_p, where $p \in S$. Each vertex in $\mathcal{A}(\mathcal{C})$ is either a vertex of C_p, for some

point $p \in S$ (a *type 1* vertex), or an intersection point of C_p and C_q, for two points $p, q \in S$ (a *type 2* vertex). The maximum depth of a point is realized by a type 2 vertex of $\mathcal{A}(\mathcal{C})$ (unless the maximum depth is 1). Using this observation, $\kappa^*(C, S)$ can be computed as follows.

Simple algorithms for computing $\kappa^*(C, S)$. As in [10], we repeat the following step for each point $p \in S$. Compute the intersections between ∂C_p and all other boundaries ∂C_q, where $q \in S$, and sort them along ∂C_p. Compute the depth of one intersection with respect to S by brute force and step through the remaining ones in sorted order while maintaining the depth. Finally report a point of the maximum depth encountered for any $p \in S$.

It is clear that the depth of the point reported is equal to $\kappa^*(C, S)$. To analyze the running time, consider a point $p \in S$. It takes $O(nI(m))$ time to compute the intersections between ∂C_p and all other boundaries ∂C_q, where $q \in S$. Since there are at most sn intersections, the sorting takes $O(sn \log(sn))$ time. Computing the depth of one intersection point takes $O(nQ(m))$ time. Finally, the total time for maintaining the depth while visiting the other intersections is $O(sn)$. Hence, the time spent by this algorithm for one point $p \in S$ is $O(n(I(m) + Q(m)) + sn \log(sn))$, giving a total running time of $O(n^2(I(m) + Q(m)) + sn^2 \log(sn))$.

In the algorithm above, we solved the optimal-placement problem by solving a restricted problem for each point p of S independently. Our second algorithm solves all these restricted problems "together".

We compute $\mathcal{A}(\mathcal{C})$ with the algorithm of Amato et al. [2] and use a standard graph-traversal algorithm, such as depth-first search, to compute a type 2 vertex of maximum depth (with respect to S). Since $\mathcal{A}(\mathcal{C})$ has $O(mn + sn^2)$ vertices, this algorithm takes $O(mn \log(mn) + nQ(m) + sn^2)$ time. Hence, we obtain the following.

Theorem 2.1 *Let S be a set of n points in the plane and let C be a disk satisfying assumptions (A1)–(A3). The optimal-placement problem can be solved in time $O\left(n^2(I(m) + Q(m)) + sn^2 \log(sn)\right)$ or in time $O\left(mn \log(mn) + nQ(m) + sn^2\right)$.*

If $m = O(1)$, then $s, Q(m) = O(1)$, and if C is convex, then $s = 2$ [15] and $I(m), Q(m) = O(\log m)$ [18]. (For the upper bounds on $I(m)$ and $Q(m)$, an $O(m)$-time preprocessing step is needed.) Therefore Theorem 2.1 implies the following.

Corollary 2.2 *Let S be a set of n points in the plane and let C be a disk satisfying assumptions (A1)–(A3). The optimal-placement problem can be solved in $O(n^2)$ time if C is fat and has $O(1)$ edges, and in $O(n^2 \log(mn)+m)$ or $O(mn \log(mn) + n^2)$ time if C is a convex m-gon.*

Bucketing and estimating $\kappa^*(C, S)$. For any two positive real numbers r and r', we denote by $\mathcal{G}_{r,r'}$ the two-dimensional grid through the origin whose cells have horizontal and vertical sides of lengths r and r', respectively. Hence, each cell of $\mathcal{G}_{r,r'}$ is of the form $B_{ij} = [ir, (i+1)r) \times [jr', (j+1)r')$ for some integers i and j. We call the pair (i, j) the *index* of B_{ij}.

We need an algorithm that groups the points of S according to the cells of some grid $\mathcal{G}_{r,r'}$, i.e., stores S in a list such that for each grid cell B, all points of S in B occur together in a contiguous sublist. This operation is similar to a sorting of the elements of S by their associated grid cells, but does not require the full power of sorting. Let $T_{\mathrm{g}}(n)$ denote the time needed to perform such a grouping of n points according to some grid and assume that T_{g} is nondecreasing and *smooth* in the sense that $T_{\mathrm{g}}(O(n)) = O(T_{\mathrm{g}}(n))$ (informally, a smooth function grows polynomially).

Lemma 2.3 *Let S be a set of n points in $\mathbb{R}^2$ and let C be a disk satisfying assumption (A3). Let $a, b > 0$ be such that $R_0 \subseteq C \subseteq R_1$ for axes-parallel rectangles R_0 of width a and height b and R_1 of width αa and height βb. Let M be the maximum number of points of S contained in any cell of the grid $\mathcal{G}_{a,b}$. Then $M \leq \kappa^*(C, S) \leq (\alpha+1)(\beta+1)M$.*

Proof. Let B be a grid cell that contains M points of S, and let $t \in \mathbb{R}^2$ be the point such that the translated rectangle $R_0 + t$ coincides with B. Then the set $C + t$ contains $R_0 + t$ and, hence, $C + t$ contains at least M points of S. This proves that $\kappa^*(C, S) \geq M$.

To prove the second inequality, let $t \in \mathbb{R}^2$ be a point such that the set $C + t$ contains $\kappa^*(C, S)$ points of S. The set $C + t$ is contained in the translated rectangle $R_1 + t$. It is easy to see that the number of grid cells that overlap $R_1 + t$ is at most $(\alpha + 1)(\beta + 1)$. Since all these grid cells together contain at least $\kappa^*(C, S)$ points of S, one of them contains at least $\kappa^*(C, S)/((\alpha+1)(\beta+1))$ points of S. ∎

Lemma 2.3 shows that an approximation M to $\kappa^*(C, S)$ with $M \leq \kappa^*(C, S) \leq (\alpha + 1)(\beta + 1)M$ can be computed in $O(T_{\mathrm{g}}(n))$ time. Let us

see how the grouping of S can be implemented. It is clear that once each point p of S has been mapped to the index (i, j) of the cell containing p of a grid under consideration, S can be grouped with respect to the grid in $O(n \log n)$ time by sorting the pairs (i, j) lexicographically. The mapping of points to grid indices uses the nonalgebraic floor function. To avoid this, we can replace the grid by the *degraded grid* introduced in [8, 17], which can be constructed in $O(n \log n)$ time without using the floor function, and for which Lemma 2.3 also holds. Given any point $p \in S$, the cell of the degraded grid that contains p can be found in $O(\log n)$ time, so that the grouping can be completed in $O(n \log n)$ time.

In a more powerful model of computation, after mapping S to grid indices, we can carry out the grouping by means of hashing. Combining the universal class of Dietzfelbinger et al. [9] with a hashing scheme of Bast and Hagerup [4], we obtain a Las Vegas grouping algorithm that finishes in $O(n)$ time with probability at least $1 - 2^{-n^\mu}$ for some fixed $\mu > 0$. We omit the details.

A bucketing algorithm. We can use Lemma 2.3 and Theorem 2.1 to obtain a faster algorithm for computing $\kappa^* = \kappa^*(C, S)$ in some cases. Suppose we have an algorithm $\mathcal{A}$ that computes κ^* in time $T(n)$. We assume that the functions $T(n)$ and $T(n)/n$ are nondecreasing and that T is smooth.

Using Lemma 2.3, we first compute M and, for each pair $(i, j) \in \{0, 1, 2\}^2$, consider the grid $\mathcal{G}^{ij}$ obtained by shifting $\mathcal{G}_{3\alpha a, 3\beta b}$ by the vector $(i\alpha a, j\beta b)$. For each cell B of $\mathcal{G}^{ij}$ that contains at least M points of S, we run $\mathcal{A}$ on the set $S \cap B$ to obtain a point y_B of maximum depth k_B with respect to $S \cap B$. Finally we return a point y_B for which k_B is maximum over all runs of $\mathcal{A}$.

To see the correctness of the algorithm, let $t \in \mathbb{R}^2$ satisfy $d_S(t) = \kappa^*$. Observe that for some $0 \le i, j \le 2$, t lies in the middle ninth of some cell B of $\mathcal{G}^{ij}$, in which case $C + t \subseteq B$. It is now clear that $|S \cap B| \ge d_S(t) = \kappa^* \ge M$ and $k_B = \kappa^*$.

Let us analyze the running time. The algorithm spends $T_{\mathrm{g}}(n)$ time to compute M, and for each of $0 \le i, j \le 2$, it spends $T_{\mathrm{g}}(n)$ time to partition S among the grid cells. Since at most n/M cells of $\mathcal{G}^{ij}$ contain at least M points of S and no cell contains more than $9\alpha\beta M$ points, the total running time is $O(T_{\mathrm{g}}(n) + nT(\alpha\beta M)/M) = O(T_{\mathrm{g}}(n) + nT(\alpha\beta\kappa^*)/\kappa^*)$, where in the last step we used the relation $M \le \kappa^*$ and the assumption that $T(n)/n$ is nondecreasing. We thus obtain the following result.

Theorem 2.4 *Let S be a set of n points in the plane, let C be a disk satisfying assumptions (A1)–(A3), and let $\mathcal{A}$ be an algorithm that computes $\kappa^* = \kappa^*(C,S)$ in time $T(n)$. Then the optimal-placement problem can be solved in $O(T_{\mathrm{g}}(n) + nT(\alpha\beta\kappa^*)/\kappa^*)$ time.*

Corollary 2.5 *Let S be a set of n points in the plane, let C be a disk satisfying assumptions (A1)–(A3), and let $\kappa^* = \kappa^*(C,S)$. The optimal-placement problem can be solved in $O(T_{\mathrm{g}}(n) + n\kappa^*)$ time if C is fat and has $O(1)$ edges, and in $O(T_{\mathrm{g}}(n) + n\kappa^* \log(m\kappa^*) + m)$ or $O(T_{\mathrm{g}}(n) + mn\log(m\kappa^*) + n\kappa^*)$ time if C is a convex m-gon.*

Proof. The proof follows by taking for $\mathcal{A}$ any of the three algorithms of Corollary 2.2 and applying Theorem 2.4 to it. ∎

3 Monte-Carlo Algorithms

In this section we present Monte-Carlo algorithms for the ε-optimal-placement problem. These algorithms use one of the deterministic algorithms described in Theorem 2.1 as a subroutine. We will refer to this algorithm as $\mathcal{A}$ and to its running time as $T(n)$. We assume that $T(n)$ and $T(n)/n$ are nondecreasing and that T is smooth.

3.1 A random-sampling approach

We first present an algorithm based on *random-sampling*. We carry out the probabilistic analysis using the following variant of the well-known Chernoff bounds. The proof of the first claim can be found in Hagerup and Rüb [13]. The proof of the second claim will be given at the end of this section.

Lemma 3.1 *Let R be a binomially distributed random variable and let $0 \leq \lambda \leq 1$.*

1. *For every $r \leq E(R)$, $\Pr[R \leq (1-\lambda)r] \leq e^{-\lambda^2 r/2}$.*
2. *For every $r \geq E(R)$, $\Pr[R \geq (1+\lambda)r] \leq e^{-\lambda^2 r/3}$.*

Theorem 3.2 *Let S be a set of n points in the plane and let C be a disk satisfying assumptions (A1)–(A3). For arbitrary $\varepsilon, \gamma > 0$, an ε-approximate solution to the optimal-placement problem can be computed in $O(n + T(\gamma n))$ time with error probability at most $sne^{-\varepsilon^2 \gamma \kappa^*}$, where $\kappa^* = \kappa^*(C, S)$.*

Proof. Let $\bar{\varepsilon} = \min\{\varepsilon, 1/2\}$ and $\bar{\gamma} = \min\{288\gamma, 1\}$. The algorithm first draws a $\bar{\gamma}$-sample S' of S, i.e., includes every point of S in S' with probability $\bar{\gamma}$ and independently of all other points. If $|S'| > 2\bar{\gamma}n$ (the sampling *fails*), the algorithm returns an arbitrary point. Otherwise it uses $\mathcal{A}$ to return a point y of maximum depth with respect to S'.

Since $\bar{\gamma} = O(\gamma)$ and T is smooth, it is clear that the algorithm can be executed in $O(n + T(\gamma n))$ time. By Lemma 3.1, the sampling fails with probability at most $e^{-\bar{\gamma}n/3}$. If $\varepsilon \geq 1$ or $\bar{\gamma} = 1$, the output is obviously correct. Assume that this is not the case and that the sampling succeeds.

Let us write d for d_S and d' for $d_{S'}$ and let $t \in \mathbb{R}^2$ be a point with $d(t) = \kappa^*$. Informally, our proof proceeds as follows. Let $Z = \{z \in \mathbb{R}^2 \mid d(z) < (1-\varepsilon)\kappa^*\}$ be the set of "bad" points. The error probability is equal to $\Pr[y \in Z]$. We first show that $d'(y)$ is likely to be large, where "large" means at least $(1 - \bar{\varepsilon}/2)\bar{\gamma}\kappa^*$. Subsequently we show that for every $z \in Z$, $d'(z)$ is not likely to be large. Combining the two assertions shows that except with small probability, $y \notin Z$.

The first part is easy: Since $E(d'(t)) = \bar{\gamma}\kappa^*$ and $d'(y) \geq d'(t)$, Lemma 3.1 implies that

$$\Pr[d'(y) < (1 - \bar{\varepsilon}/2)\bar{\gamma}\kappa^*] \leq e^{-\bar{\varepsilon}^2\bar{\gamma}\kappa^*/8}.$$

Now fix $z \in Z$. Since $1 - \bar{\varepsilon}/2 \geq (1 + \bar{\varepsilon}/2)(1 - \bar{\varepsilon})$ and $E(d'(z)) = \bar{\gamma}d(z) < (1 - \varepsilon)\bar{\gamma}\kappa^* \leq (1 - \bar{\varepsilon})\bar{\gamma}\kappa^*$, we have

$$\Pr[d'(z) \geq (1 - \bar{\varepsilon}/2)\bar{\gamma}\kappa^*] \leq \Pr[d'(z) \geq (1 + \bar{\varepsilon}/2)(1 - \bar{\varepsilon})\bar{\gamma}\kappa^*] \leq e^{-\bar{\varepsilon}^2(1-\bar{\varepsilon})\bar{\gamma}\kappa^*/12}.$$

The preceding argument applies to a fixed $z \in Z$. A priori, we have to deal with an infinite number of candidate points $z \in Z$. However, using the fact that the arrangement defined by the boundaries of the sets C_p, with $p \in S$, has $O(sn^2)$ vertices of type 2, it is not difficult to see that there is a set X with $|X| = O(sn^2)$ such that for every $z \in Z$, there is a $\hat{z} \in X \cap Z$ with $d'(\hat{z}) = d'(z)$. Therefore the probability that $d'(z) \geq (1 - \bar{\varepsilon}/2)\bar{\gamma}\kappa^*$ for some $z \in Z$ is $O(sn^2 e^{-\bar{\varepsilon}^2(1-\bar{\varepsilon})\bar{\gamma}\kappa^*/12})$. The other failure probabilities identified above are no larger. Now, $\bar{\varepsilon} \geq \varepsilon/2$, $1 - \bar{\varepsilon} \geq 1/2$, and $\bar{\gamma} = 3 \cdot 8 \cdot 12\gamma$. Moreover, we can assume that $e^{\varepsilon^2\gamma\kappa^*} \geq sn$, since otherwise the theorem claims nothing.

But then $sn^2 e^{-\bar{\varepsilon}^2(1-\bar{\varepsilon})\bar{\gamma}\kappa^*/12} \leq sn^2 e^{-3\varepsilon^2\gamma\kappa^*} \leq (1/s)e^{-\varepsilon^2\gamma\kappa^*}$. Therefore, except if n is bounded by some constant (in which case we can use a deterministic algorithm), the failure probability is at most $sne^{-\varepsilon^2\gamma\kappa^*}$. ∎

Combining Theorem 3.2 with Corollary 2.2, we obtain the following result.

Corollary 3.3 *Let S be a set of n points in the plane, let C be a disk satisfying assumptions (A1)–(A3), and let $\kappa^* = \kappa^*(C,S)$. For arbitrary $\varepsilon, \gamma > 0$, an ε-approximate placement can be computed with error probability at most $ne^{-\varepsilon^2\gamma\kappa^*}$ in $O(n + (\gamma n)^2)$ time if C is fat and has $O(1)$ edges, and in $O(n + (\gamma n)^2 \log(\gamma mn) + m)$ or $O(n + \gamma mn \log(\gamma mn) + (\gamma n)^2)$ time if C is a convex m-gon.*

Our random-sampling approach can also be used to solve related problems. As an example, let H be a set of n halfplanes. The *depth* of a point $t \in \mathbb{R}^2$ with respect to H is defined as the number of halfplanes in H that contain t. We consider the problem of computing a point of maximum depth κ^*.

Consider a coordinate system such that none of the lines bounding the halfplanes in H is vertical. Then any halfplane in H is either an "upper" or a "lower" halfplane. At least half of the elements of H are all upper halfplanes, or all lower halfplanes. Assume w.l.o.g. that the first case holds. Then any point t that is above the lines bounding all these upper halfplanes, is in at least $n/2$ of the halfplanes of H. Since such a point t exists, it follows that $k^* \geq n/2$.

By computing and traversing the arrangement defined by the bounding lines of the halfplanes, one can compute κ^* in $O(n^2)$ time. Since a corresponding decision problem is 3SUM-hard (see Gajentaan and Overmars [12]), it is unlikely that it can be solved in subquadratic time. If we apply our random-sampling transformation with $\gamma = c/\sqrt{n}$ for a suitable constant $c > 0$, we obtain the following result.

Theorem 3.4 *Let H be a set of n halfplanes. For arbitrary constant $\varepsilon > 0$, in $O(n)$ time we can compute a point in $\mathbb{R}^2$ whose depth with respect to H is at least $(1-\varepsilon)\kappa^*$, except with probability at most $e^{-\sqrt{n}}$.*

Proof of the second claim in Lemma 3.1. Let N be a positive integer and let μ be a real number such that R can be interpreted as the number of

heads in a sequence of N coin flips, each one coming up heads with probability μ.

Let $r \geq E(R)$. If $r > N$, then $\Pr[R \geq (1+\lambda)r] = 0$, and the claim clearly holds. So assume that $r \leq N$. Let $\mu' := r/N$ and let R' be binomially distributed with parameters N and μ'. Then $E(R') = \mu' N = r$. A variant of the Chernoff bounds (see Hagerup and Rüb [13]) states that

$$\Pr[R' \geq (1+\lambda)r] \leq e^{-\lambda^2 r/3}.$$

We have $\mu = E(R)/N \leq r/N = \mu'$. The proof of the second claim in Lemma 3.1 follows from the fact that

$$\Pr[R \geq (1+\lambda)r] \leq \Pr[R' \geq (1+\lambda)r]. \tag{1}$$

To prove (1), consider two coins. The first one comes up heads with probability μ/μ', whereas the second one comes up heads with probability μ'. For each i with $1 \leq i \leq N$, we do the following experiment: Flip both coins and define two random variables U_i and U_i'. The value of U_i is one if both coins come up heads and zero otherwise. The value of U_i' is one if the second coin comes up heads and zero otherwise. We have $\Pr[U_i = 1] = \mu$ and $\Pr[U_i' = 1] = \mu'$. Since $U_i \leq U_i'$ for all i, we have

$$\Pr\left[\sum_{i=1}^{N} U_i \geq (1+\lambda)r\right] \leq \Pr\left[\sum_{i=1}^{N} U_i' \geq (1+\lambda)r\right].$$

This implies that (1) holds, because $\sum_{i=1}^{N} U_i$ and $\sum_{i=1}^{N} U_i'$ have the same probability distributions as R and R', respectively. ∎

3.2 Bucketing and sampling combined

We now present a Monte Carlo algorithm that combines Theorem 3.2 with the bucketing algorithm described in Section 2.

First compute M, as defined in Lemma 2.3. Next, for each $(i,j) \in \{0,1,2\}^2$, consider the grid $\mathcal{G}^{ij}$ as in Section 2. Fix a parameter $\gamma > 0$. For each cell B of $\mathcal{G}^{ij}$ with $|S \cap B| \geq M$, run the algorithm described in the proof of Theorem 3.2 on the set $S \cap B$ to obtain a point y_B and compute the value $k_B = d_{S \cap B}(y_B)$. Finally return a point y_B for which k_B is maximum.

We first analyze the running time of this algorithm. Fix $(i,j) \in \{0,1,2\}^2$. For each cell B of $\mathcal{G}^{ij}$ with $|S \cap B| \geq M$, the algorithm spends $O(|S \cap B| +$

$T(\gamma|S \cap B|) + |S \cap B|Q(m)) = O(T(\gamma|S \cap B|) + |S \cap B|Q(m))$ time. Since there are at most n/M such cells B and $|S \cap B| \leq 9\alpha\beta M$, the total running time is $O(T_{\rm g}(n) + \alpha\beta nQ(m) + nT(\alpha\beta\gamma\kappa^*)/\kappa^*)$.

Next, we estimate the error probability. Consider the depth k_B of the point y_B returned by the algorithm. The error probability of our algorithm is $\Pr[k_B < (1-\epsilon)\kappa^*]$, where $\kappa* = \kappa^*(C,S)$. Let $t \in \mathbb{R}^2$ be a point such that the set $C+t$ contains κ^* points of S and let i' and j' be integers such that the cell B' of the grid $\mathcal{G}^{i'j'}$ that contains t contains all points of $S \cap (C+t)$. We have $M \leq \kappa^* \leq |S \cap B'| \leq 9\alpha\beta\kappa^*$. Consider the value $k_{B'} = d_{S\cap B'}(y_{B'})$ that is computed when cell B' is considered. Since $k_B \geq k_{B'}$, the error probability is less than or equal to $\Pr[k_{B'} < (1-\epsilon)\kappa^*]$. Since B' does not depend on the outcome of our random process, Theorem 3.2 gives an upper bound on the latter probability. We have proved the following.

Theorem 3.5 *Let S be a set of n points in the plane and let C be a disk satisfying assumptions (A1)–(A3). For arbitrary $\varepsilon, \gamma > 0$, an ε-approximate placement of C can be computed in $O(T_{\rm g}(n) + \alpha\beta nQ(m) + nT(\alpha\beta\gamma\kappa^*)/\kappa^*)$ time by a Monte Carlo algorithm with error probability at most $s\alpha\beta\kappa^* e^{-\varepsilon^2\gamma\kappa^*}$, where $\kappa^* = \kappa^*(C,S)$.*

Corollary 3.6 *Let S be a set of n points in the plane and let C be a disk satisfying assumptions (A1)–(A3). For arbitrary constant $\varepsilon > 0$, an ε-approximate placement of C can be computed in $O(n \log n)$ time with probability of error at most $e^{-\sqrt{\kappa^* \log n}}$ if C is fat and has $O(1)$ edges, and in $O(n\log(mn)+m)$ time with probability of error at most $e^{-\sqrt{\kappa^* \log(mn)/\log(m\kappa^*)}}$ if C is a convex m-gon.*

Proof. To prove the first claim, let $\mathcal{A}$ be the first algorithm of Corollary 2.2. Then $T(n) = O(n^2)$. Applying Theorem 3.5 to $\mathcal{A}$, we obtain an ε-approximation algorithm for the optimal-placement problem with running time $O(n\log n + \gamma^2 n\kappa^*)$ and error probability $O(\kappa^* e^{-\varepsilon^2\gamma\kappa^*})$. If we choose $\gamma = (2/\varepsilon^2)\sqrt{(\log n)/M}$, where M is as in Lemma 2.3, the running time and the error probability are as claimed for n larger than some constant.

For the proof of the second claim, we take $\mathcal{A}$ to be the second algorithm of Corollary 2.2. Then $T(n) = O(n^2\log(mn) + m)$. If we apply Theorem 3.5 to $\mathcal{A}$, we obtain an ε-approximation algorithm for the optimal-placement problem with running time $O(n\log(mn) + n\gamma^2\kappa^*\log(m\gamma\kappa^*) + m)$ and error probability $O(\kappa^* e^{-\varepsilon^2\gamma\kappa^*})$. We choose $\gamma = c\sqrt{\log(mn)/(M\log(mM))}$, where

M is as in Lemma 2.3 and c is a sufficiently large constant. Except if $\gamma > 1$, in which case we can use the second algorithm of Corollary 2.5, this gives the running time and the error probability claimed for n sufficiently large. ∎

Theorem 3.7 *Let S be a set of n points in the plane and let C be a fat disk that satisfies assumptions (A1)–(A3) and has $O(1)$ edges. For arbitrary constant $\varepsilon > 0$, an ε-approximate placement of C can be computed in $O(n)$ time with error probability at most $ne^{-\sqrt{\kappa^*}}$.*

Proof. We use the following algorithm, which might be described as sampling followed by bucketing followed by sampling: Draw a random γ-sample S' of S, where $\gamma = \min\{L/\log n, 1\}$ for a constant $L > 0$ to be chosen below. If $|S'| > 2\gamma n$, return an arbitrary point. Otherwise apply the first algorithm of Corollary 3.6 to S', but with approximation parameter $\varepsilon/4$, rather than ε, and return the point returned by that algorithm. The overall running time is clearly $O(n)$.

As in the proof of Theorem 3.2, we write d for d_S and d' for $d_{S'}$. Assume that $\varepsilon \leq 1$ and that $|S'| \leq 2\gamma n$ and let κ' be the maximum value of $d'(t)$ over all points t in the plane. By Lemma 3.1, $\Pr[\kappa' < (1-\varepsilon/4)\gamma\kappa^*] \leq e^{-c_1\gamma\kappa^*}$ for some constant $c_1 > 0$. Moreover, the analysis in the proof of Theorem 3.2 shows the probability that $d'(z) \geq (1-\varepsilon/2)\gamma\kappa^*$ for some $z \in \mathbb{R}^2$ with $d(z) < (1-\varepsilon)\kappa^*$ to be $O(ne^{-c_2\gamma\kappa^*})$ for some other constant $c_2 > 0$. Finally, the probability that the algorithm of Corollary 3.6 returns a point y with $d'(y) < (1-\varepsilon/4)\kappa'$ is at most $e^{-\sqrt{\kappa'\log n}}$. Observe that $\kappa' \geq (1-\varepsilon/4)\gamma\kappa^*$ and $d'(y) \geq (1-\varepsilon/4)\kappa'$ imply $d'(y) \geq (1-\varepsilon/2)\gamma\kappa^*$. Therefore, for some constant $c > 0$, the answer is correct, except with probability $O(ne^{-c\gamma\kappa^*} + e^{-c\sqrt{\gamma\kappa^*\log n}})$. If $\gamma < 1$, the probability under consideration is $O(ne^{-cL\kappa^*/\log n} + e^{-c\sqrt{L\kappa^*}})$. We can assume that $\kappa^* \geq (\log n)^2$, since otherwise there is nothing to prove. But then it is clear that for L chosen sufficiently large and for n larger than some constant, the error probability is at most $ne^{-\sqrt{\kappa^*}}$. ∎

4 A Deterministic Approximation Algorithm

In this section we present a deterministic algorithm for the ε-optimal-placement problem. For simplicity, we assume that C is convex and has $O(1)$ edges. The algorithm can be extended to more general cases. For example, at the

cost of an extra $O(\log m)$-factor in the running time, the algorithm can be extended to arbitrary convex m-gons. Throughout this section, S denotes a set of n points in the plane, $\mathcal{C}$ denotes the set of n translates C_p, with $p \in S$, $\mathcal{A}(\mathcal{C})$ denotes the arrangement defined by the boundaries of the elements of $\mathcal{C}$, and κ^* denotes $\kappa^*(C, S)$.

4.1 Cuttings

We will refer to a simply connected region with at most four edges (left and right edges being vertical segments and top and bottom edges being portions of the boundaries of translates of $-C$) as a *pseudo-trapezoid*. For technical reasons, we will also regard vertical segments and portions of the boundaries of translates of $-C$ as 1-dimensional pseudo-trapezoids. For a pseudo-trapezoid Δ and a set $\mathcal{F}$ of translates of $-C$, we will use $\mathcal{F}_\Delta \subseteq \mathcal{F}$ to denote the set of all elements of $\mathcal{F}$ whose boundaries intersect Δ. The vertical decomposition $\mathcal{A}^{\|}(\mathcal{C})$ of $\mathcal{A}(\mathcal{C})$ partitions the plane into pseudo-trapezoids. Let $\mathcal{F}$ be a subset of $\mathcal{C}$ and let Δ be a pseudo-trapezoid. We will denote by $\chi(\mathcal{F}, \Delta)$ the number of pairs of elements of $\mathcal{F}$ whose boundaries intersect inside Δ. If Δ is the entire plane, we use the notation $\chi(\mathcal{F})$ to denote $\chi(\mathcal{F}, \Delta)$, for brevity. Given a parameter $r \geq 1$, a partition Ξ of Δ into a collection of pairwise openly-disjoint pseudo-trapezoids is called a $(1/r)$-*cutting* of $(\mathcal{F}, \Delta)$ if $|\mathcal{F}_\tau| \leq |\mathcal{F}|/r$ for every pseudo-trapezoid $\tau \in \Xi(\mathcal{F}, \Delta)$. The *conflict list* of a pseudo-trapezoid τ in Ξ is the set $\mathcal{F}_\tau$.

We state the following technical result in full generality, since it is of independent interest and may find additional applications. We apply it here only with Δ being the whole plane.

Theorem 4.1 *Let Δ be a pseudo-trapezoid, let $r \geq 1$, and let $\delta > 0$ be an arbitrarily small constant. A $(1/r)$-cutting of $(\mathcal{C}, \Delta)$ of size $O(r^{1+\delta} + \chi(\mathcal{C}, \Delta)r^2/n^2)$, along with the conflict lists of its pseudo-trapezoids, can be computed in $O(nr^\delta + \chi(\mathcal{C}, \Delta)r/n)$ time, where the constants of proportionality depend on δ.*

We prove this theorem by adapting Chazelle's cutting algorithm [6] to our setting. We call a subset $\mathcal{F}$ of $\mathcal{C}$ a $(1/r)$-*approximation* if, for every pseudo-trapezoid Δ,

$$\left| \frac{|\mathcal{C}_\Delta|}{|\mathcal{C}|} - \frac{|\mathcal{F}_\Delta|}{|\mathcal{F}|} \right| < \frac{1}{r}.$$

A result by Brönnimann et al. [5] implies that a $(1/r)$-approximation of $\mathcal{C}$ of size $O(r^2 \log r)$ can be computed in time $nr^{O(1)}$. Moreover, an argument similar to the one in [6] implies the following:

Lemma 4.2 *Let $\mathcal{F}$ be a $(1/r)$-approximation of $\mathcal{C}$. For every pseudo-trapezoid Δ,*

$$\left| \frac{\chi(\mathcal{C}, \Delta)}{|\mathcal{C}|^2} - \frac{\chi(\mathcal{F}, \Delta)}{|\mathcal{F}|^2} \right| < \frac{1}{r}.$$

Next, we call a subset $\mathcal{H}$ of $\mathcal{C}$ a $(1/r)$-*net* of $\mathcal{C}$ if, for any pseudo-trapezoid Δ, $|\mathcal{C}_\Delta| > |\mathcal{C}|/r$ implies that $\mathcal{H}_\Delta \neq \emptyset$. A $(1/r)$-net $\mathcal{H}$ is called *sparse* for Δ if

$$\frac{\chi(\mathcal{H}, \Delta)}{\chi(\mathcal{C}, \Delta)} \leq 4 \left(\frac{|\mathcal{H}|}{n} \right)^2.$$

As in [6], we can prove the following.

Lemma 4.3 *Given a pseudo-trapezoid Δ and a parameter $r \geq 1$, we can compute, in $n^{O(1)}$ time, a $(1/r)$-net of $\mathcal{C}$ having size $O(r \log n)$ and that is sparse for Δ.*

Proof of Theorem 4.1. Using Lemmas 4.2 and 4.3, we compute a $(1/r)$-cutting of $(\mathcal{C}, \Delta)$ as follows. Let c be a sufficiently large constant whose value will be chosen later. For every $0 \leq j \leq \lceil \log_c r \rceil$, we compute a $(1/c^j)$-cutting Ξ_j of $(\mathcal{C}, \Delta)$. The final cutting is a $(1/r)$-cutting of $(\mathcal{C}, \Delta)$. While computing Ξ_j, we also compute the conflict lists of the pseudo-trapezoids in Ξ_j.

Ξ_0 is Δ itself, and $\mathcal{C}$ is the conflict list of Δ. We compute Ξ_j from Ξ_{j-1} as follows. For each pseudo-trapezoid $\tau \in \Xi_{j-1}$, if $|\mathcal{C}_\tau| \leq n/c^j$, then we do nothing in τ. Otherwise, we first compute a $(1/2c)$-approximation $\mathcal{F}_\tau$ of $\mathcal{C}_\tau$ of size $O(c^2 \log c)$ and then a $(1/2c)$-net $\mathcal{H}_\tau$ of $\mathcal{F}_\tau$ of size $O(c \log c)$ that is sparse for τ. Note that $\mathcal{H}_\tau$ is a $(1/c)$-net of $\mathcal{C}_\tau$. We then compute the vertical decomposition $\mathcal{A}^{\|}(\mathcal{H}_\tau)$ of $\mathcal{A}(\mathcal{H}_\tau)$ within τ. $\mathcal{A}^{\|}(\mathcal{H}_\tau)$ consists of $O(|\mathcal{H}_\tau| + \chi(\mathcal{H}_\tau, \tau))$ cells. We replace τ with the pseudo-trapezoids of $\mathcal{A}^{\|}(\mathcal{H}_\tau)$. Repeating this for all $\tau \in \Xi_{j-1}$, we form Ξ_j from Ξ_{j-1}. It is easy to see that Ξ_j is a $(1/c^j)$-cutting of $(\mathcal{C}, \Delta)$.

By an analysis similar to the one in [6], it can be shown that by choosing the constant c sufficiently large (but depending on δ), the size of the final cutting is $O(r^{1+\delta} + \chi(\mathcal{C}, \Delta) r^2/n^2)$ and that the running time of the algorithm is $O(nr^\delta + \chi(\mathcal{C}, \Delta) r/n)$. This completes the proof of Theorem 4.1. ■

By a result of Sharir [20], $\chi(\mathcal{C}, \Delta) = O(n\kappa^*)$, so Theorem 4.1 implies the following.

Corollary 4.4 *Let $r \geq 1$ and let $\delta > 0$ be an arbitrarily small constant. A $(1/r)$-cutting $\Xi(\mathcal{C})$ of size $O(r^{1+\delta} + \kappa^* r^2/n)$, along with the conflict lists, can be computed in $O(nr^\delta + \kappa^* r)$ time.*

4.2 The approximation algorithm

Let $\delta, \varepsilon > 0$ be real numbers. We now present a deterministic ε-approximation algorithm. We will need the following lemma.

Lemma 4.5 *Let $r \geq 1$ and let Ξ be a $(1/r)$-cutting of $\mathcal{C}$. Then we can compute a point of depth (with respect to S) at least $\kappa^* - n/r$ in $O(|\Xi|n/r)$ time.*

Proof. Let Δ be a pseudo-trapezoid of Ξ. The maximum depth (with respect to S) of any point inside Δ is at most n/r plus the depth (with respect to S) of any vertex of Δ. It thus suffices to return a vertex of (a pseudo-trapezoid of) Ξ of maximum depth (with respect to S). We can compute the depths of all vertices of Ξ by following an Eulerian tour on the dual graph of the planar subdivision induced by Ξ; see, e.g., [1]. As shown in [1], the time spent in this step is proportional to the total size of all the conflict lists in Ξ. Since the size of each conflict list is at most n/r, the claim follows. ∎

Our algorithm works in two stages. In the first stage, we estimate the value of κ^* to within a factor of 9, and then we use Lemma 4.5 to compute an ε-approximation of κ^*.

I. Using the bucketing algorithm of Section 2, we obtain a coarse estimate k_0 of κ^* that satisfies $k_0/9 \leq \kappa^* \leq k_0$. (Since we assume that C is convex, we have $\alpha = \beta = 2$, as follows from [19], which leads to the constant 9 in the estimate above.)

II. We set $r = \min\left\{\frac{9n}{\varepsilon k_0}, n\right\}$, compute a $(1/r)$-cutting Ξ of $\mathcal{C}$, and return a point of maximum depth (with respect to S) among the vertices of Ξ. Denote this maximum depth by k.

By Lemma 4.5, and assuming that $r = \frac{9n}{\varepsilon k_0}$,

$$k \geq \kappa^* - \frac{n}{r} = \kappa^* - \frac{n}{9n/(\varepsilon k_0)} = \kappa^* - \frac{\varepsilon k_0}{9} \geq (1-\varepsilon)\kappa^*.$$

If $r = n$, then $k \geq \kappa^* - n/r = \kappa^* - 1$, which is at least $(1-\varepsilon)\kappa^*$, since we may assume that $\varepsilon \geq 1/\kappa^*$.

As shown in Section 2, Step I can be implemented in $O(n \log n)$ time. Using Theorem 4.1 and Lemma 4.5, Step II takes time

$$O\left(nr^\delta + \frac{\kappa^* n}{\varepsilon k_0}\right) = O\left(n^{1+\delta} + \frac{n}{\varepsilon}\right).$$

Hence, we conclude the following.

Theorem 4.6 *Let S be a set of n points in the plane and let C be a convex disk with $O(1)$ edges. Assume that assumptions (A1)–(A3) are satisfied and let $\varepsilon > 0$ be a real number. An ε-approximate placement of C can be computed in $O(n^{1+\delta} + n/\epsilon)$ time, for any given constant $\delta > 0$.*

5 Concluding Remarks

We have given a unified approach to solve the problem of computing a translate of a compact set C such that it contains a largest subset of a given set of n points. We started with two basic algorithms. Then we presented two transformations, based on random sampling and bucketing, respectively. By combining these transformations with any of the basic algorithms, we obtained different algorithms, either solving the exact optimal placement problem, or the approximation version of this problem. In the latter case, the algorithm is of the Monte Carlo type, with a guaranteed upper bound on the error probability. We also presented a deterministic algorithm for the approximation problem using an adaptation of Chazelle's cutting algorithm.

We leave open the problem of improving the error probabilities of our Monte Carlo approximation algorithms for small values of $\kappa^* = \kappa^*(C, S)$. Consider for example the case when the set C is a circular disk. We obtained a running time of $O(n \log n)$ with error probability at most $e^{-\sqrt{\kappa^* \log n}}$, see Corollary 3.6. We also obtained a running time of $O(n)$, except with probability at most $ne^{-\sqrt{\kappa^*}}$, see Theorem 3.7, which is only "good" if $k^* \geq \log n$.

References

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

[2] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, pages 705–706, 2000.

[3] G. Barequet, M. Dickerson, and P. Pau. Translating a convex polygon to contain a maximum number of points. *Comput. Geom. Theory Appl.*, 8:167–179, 1997.

[4] H. Bast and T. Hagerup. Fast and reliable parallel hashing. In *Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 50–61, 1991.

[5] H. Brönnimann, B. Chazelle, and J. Matoušek. Product range spaces, sensitive sampling, and derandomization. *SIAM J. Comput.*, 28:1552–1575, 1999.

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

[7] B. Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36:1–16, 1986.

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

[9] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25:19–51, 1997.

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

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

[12] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

[13] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Inform. Process. Lett.*, 33:305–308, 1990.

[14] D. P. Huttenlocher and S. Ullman. Object recognition using alignment. In *Proc. 1st Internat. Conf. Comput. Vision*, pages 102–111, 1987.

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

[16] Y. Lamdan, J. T. Schwartz, and H. J. Wolfson. Object recognition by affine invariant matching. In *Proc. IEEE Internat. Conf. Comput. Vision Pattern. Recogn.*, pages 335–344, 1988.

[17] H. P. Lenhof and M. Smid. Sequential and parallel algorithms for the k closest pairs problem. *Internat. J. Comput. Geom. Appl.*, 5:273–288, 1995.

[18] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, Berlin, 1988.

[19] O. Schwarzkopf, U. Fuchs, G. Rote, and E. Welzl. Approximation of convex figures by pairs of rectangles. *Comput. Geom. Theory Appl.*, 10:77–87, 1998.

[20] M. Sharir. On k-sets in arrangements of curves and surfaces. *Discrete Comput. Geom.*, 6:593–613, 1991.