Practical Methods for Shape Fitting and Kinetic Data Structures using Core Sets

Hai Yu† Pankaj K. Agarwal‡ Raghunath Poredy§ Kasturi R. Varadarajan¶

Abstract

The notion of ε-kernel was introduced by Agarwal et al. [5] to set up a unified framework for computing various extent measures of a point set P approximately. Roughly speaking, a subset Q ⊆ P is an ε-kernel of P if for every slab W containing Q, the expanded slab (1 + ε)W contains P. They illustrated the significance of an ε-kernel by showing that it yields approximation algorithms for a wide range of problems.

We present a simpler and more practical algorithm for computing the ε-kernel of a set P of points in R². We demonstrate the practicality of our algorithm by showing its empirical performance on various inputs. We then describe an incremental algorithm for fitting various shapes and use the ideas of our algorithm for computing ε-kernels to analyze the performance of this algorithm. We illustrate the versatility and practicality of this technique by implementing approximation algorithms for minimum enclosing cylinder, minimum-volume bounding box, and minimum-width annulus. Finally, we show that ε-kernels can be effectively used to expedite the algorithms for maintaining extents of moving points.

Categories and Subject Descriptors: F.2.2 [Numerical Algorithms and Problems]: Geometrical problems and computations

*Research by the first two authors is supported by NSF under grants CCR-00-68013, EIA-98-70724, EIA-01-31905, and CCR-02-04118, and by a grant from the U.S.-based Binational Science Foundation. Research by the fourth author is supported by NSF CAREER award CCR-0237431.

1. Introduction

Motivated by numerous applications, considerable work has been done on computing various descriptors of the extent of a set P of n points in R². These measures, called extent measures, either compute certain statistics of P itself or they compute certain statistics of a (possibly nonconvex) geometric shape (e.g., sphere, box, cylinder, etc.) enclosing P. Examples of the former include computing the kth largest distance between pairs of points in P, and the examples of the latter include computing the smallest radius of a sphere (or cylinder), the minimum volume (or surface area) of a box, and the smallest width of a slab (or a spherical or cylindrical shell) that contain P. Although traditionally P is assumed to be stationary (insertion/deletion of points at discrete times have been considered), recently there has been work on maintaining extent measures of a set of moving points [4]. In the latter, the goal is to maintain the extent measure as the points move, e.g., using the kinetic data structure framework [10].

The exact algorithms for computing extent measures are generally expensive, e.g., for d = 3, the best known algorithm for computing the smallest simplex containing P requires O(n³) time [24], and the smallest enclosing cylindrical shell requires O(n²³/²) time [3]. Consequently, attention has shifted to developing approximation algorithms, and several approximation algorithms for specific problems exist, see [1, 3, 9, 13, 24] and references therein. Although these algorithms are tailored to specific problems, they rely on similar techniques. A natural open question is thus whether a unified framework exists for computing extent measures approximately. Addressing this issue, Agarwal et al. [5] introduced the notion of ε-kernel for a point set P. Roughly speaking, a subset Q ⊆ P is an ε-kernel of P if for every slab W containing Q, the expanded slab (1 + ε)W contains P. They presented an O(n + n²ε⁻¹) time algorithm for computing an ε-kernel of size O(1/ε²⁻¹) or an O(n³ε⁻⁵/³) time algorithm for computing an ε-kernel of size O(1/ε²⁻¹/²). They also introduce the notion of ε-kernel for a set F of functions as follows. The extent of F at a point x ∈ R² is

\[ \mathcal{E}_F(x) = \max_{f \in F} f(x) - \min_{f \in F} f(x), \] (1)
and a subset $G \subseteq F$ is an $\varepsilon$-kernel of $F$ if $\mathcal{G}_G(x) \geq (1 - \varepsilon)\mathcal{G}_F(x)$ for all $x \in \mathbb{R}^{\dim}$. Using their result on computing the $\varepsilon$-kernel of points and the linearization technique, they compute in $O(n + 1/\varepsilon^{O(1)})$ time an $\varepsilon$-kernel of $F$ of size $O(1/\varepsilon^{\omega})$ if each $g_i$ is of the form $g^k_i$, where $g_i$ is a polynomial, $r$ is a positive integer, $\sigma = \min\{d - 1, k/2\}$, and $k$ is the dimension of linearization for $g_i$'s, see the original paper for the definition of $k$. They illustrated the power of their technique by showing that it yields approximation algorithms for a wide range of problems, including algorithms for minimum-width spherical and cylindrical shells containing $P$, kinet data structures for maintaining approximate extent measures of moving points, which process $1/\varepsilon^{O(1)}$ events; and maintaining extent measures in a streaming model in polylog($n$) time. No algorithms with similar running time were known for these problems.

Their work spurred a flurry of activity in this direction, and many subsequent papers have used a similar approach for other geometric optimization problems, including clustering and other extent-measure problems [7, 11, 12, 18, 19, 20]. These approaches compute a subset $Q$ of $P$ of small size and solve the underlying optimization problem for $Q$. The term core-set is now commonly used to refer to such a subset.

Although the algorithm by Aggarwal et al. [5] is conceptually simple, it is not really practical. It works in two stages: the first stage computes a $(\varepsilon/2)$-kernel $Q'$ of size $O(1/\varepsilon^{\omega})$, and the second stage computes a $(\varepsilon/\delta)$-kernel $Q$ of size $O(1/\varepsilon^{\omega-1})$. The second stage invokes $O(1/\varepsilon^{\omega-1})$ times Gärtner's randomized algorithm [16] (or a deterministic counterpart) that computes the face of conv($Q'$) closest to a given point lying outside the convex hull. The deterministic counterparts are rather complicated and even Gärtner's randomized algorithm is not that simple and efficient. We present a simpler and more practical algorithm for computing an $\varepsilon$-kernel of a set of points that needs only an approximate nearest-neighbor searching data structure. The worst-case running time of the simplest version of this algorithm is the same as that of Aggarwal et al. [5].

Using the duality transform and the linearization technique, as described in Aggarwal et al. [5], we can then compute $\varepsilon$-kernels for a set of linear functions, polynomials, or their roots.

We have implemented our $\varepsilon$-kernel algorithm, using the ANN library for answering approximate nearest-neighbor queries [22], and tested on a variety of inputs in dimension up to 8. Our empirical results show that our algorithm works extremely well in low dimensions ($\leq 4$) both in terms of size of the kernel and the running time. For example, to achieve an error below 0.05, the size of the $\varepsilon$-kernel is usually a few hundreds and the running time is just a few seconds. The size and running time increase exponentially with the dimension.

In Section 3 we study the shape-fitting-based extent measures. As in [5], we can simply plug our $\varepsilon$-kernel algorithm to compute various shape-fitting-based extent measures in time $O(n + 1/\varepsilon^{O(1)})$. However, we present a very simple incremental algorithm for these problems. We analyze the number of iterations of the algorithm for the case of minimum-width slab. The analysis exploits the argument for the correctness of our algorithm for computing $\varepsilon$-kernels. The analysis extends to other shape-fitting problems including minimum-width spherical and cylindrical shell, minimum-radius cylinder and the minimum-volume box enclosing a given set $P$ of points. Although we can only prove a weaker bound of $O(1/\varepsilon^{d/(d-1)/2})$ on the worst-case number of iterations of this algorithm for the case of slabs, our empirical results show that in practice the algorithm returns an $\varepsilon$-approximate solution in very few iterations. Our approach is reminiscent of Clarkson's linear-programming algorithm [15]. We illustrate the versatility of the technique by implementing approximation algorithms for minimum enclosing cylinder, minimum-volume bounding box, and minimum-width annulus. We compare the incremental algorithm for these problems with the natural algorithm of computing a kernel first and then solving the problem on the kernel. Our experiments confirm our expectation that quite often the number of iterations of the incremental algorithm is small and so it outperforms the latter, though in the worst case the incremental algorithm can be somewhat slower than the latter. Note that the incremental algorithm produces an $\varepsilon$-approximate solution in time linear in the number of points. The existence of such a natural and generic algorithm is quite interesting from the theoretical perspective also, particularly because these shape fitting problems have received considerable attention.

In Section 4 we apply our $\varepsilon$-kernel algorithm to maintain an approximation to the minimum orthogonal bounding box and the convex hull of a set of points moving in $\mathbb{R}^d$. We show that our algorithm processes very few events and maintains a good approximation of the extent as the points move. For example, for a set of 100,000 quadratically moving points, we were able to maintain an approximate minimum orthogonal bounding box with error below 0.05 over time by choosing a kernel of size $2\varepsilon$. These experimental results affirmed the significance of $\varepsilon$-kernels to kinet data structures in practice.

2. Computing an $\varepsilon$-Kemel

In this section we describe our algorithm for computing the $\varepsilon$-kernel of any set $P$ of points in $\mathbb{R}^d$, analyze its running time, and provide empirical results.

Geometric preliminaries. We first define various terms that we will need later, most of which are taken from [5]. Let $\mathbb{S}^{d-1}$ denote the unit sphere centered at the origin in $\mathbb{R}^d$. For any set $P$ of points in $\mathbb{R}^d$ and any direction $u \in \mathbb{S}^{d-1}$, we define the directional width of $P$ in direction $u$, denoted by $\omega(u, P)$, to be

$$\omega(u, P) = \max_{p \in P} \langle u, p \rangle - \min_{p \in P} \langle u, p \rangle,$$

where $\langle \cdot, \cdot \rangle$ is the inner product. Let $\varepsilon > 0$ be a parameter. A subset $Q \subseteq P$ is called an $\varepsilon$-kernel of $P$ if for each $u \in \mathbb{S}^{d-1},$

$$(1 - \varepsilon)\omega(u, P) \leq \omega(u, Q).$$

Clearly, $\omega(u, Q) \leq \omega(u, P)$.

We call $P$ $\alpha$-fat for $\alpha \leq 1$, if there exists a point $p \in \mathbb{R}^d$ and a hypercube $C$ centered at the origin so that

$$p + \alpha C \subseteq \text{conv}(P) \subseteq p + C.$$ 

A stronger version of the following lemma was proved in [5].
Lemma 2.1. Let P be a set of n points in \( \mathbb{R}^d \), and let \( C = [-1,1]^d \). One can compute in \( O(n) \) time an affine transform \( \tau \) so that \( \tau(P) \) is an \( \alpha \)-fat point set satisfying \( \alpha C \subset \text{conv}(\tau(P)) \subset C \), where \( \alpha \) is a constant depending on \( d \), and so that \( Q \) is an \( \varepsilon \)-kernel of \( P \) if and only if \( \tau(Q) \) is an \( \varepsilon \)-kernel of \( \tau(P) \).

Algorithm. In view of Lemma 2.1, we can assume that \( P \) is an \( \alpha \)-fat point set so that \( \alpha C \subset \text{conv}(P) \subset C \), where \( C = [-1,1]^d \). Suppose \( \varepsilon \leq 1/4 \). Let \( S \) be the sphere of radius \( \sqrt{d} + 1 \) centered at the origin. Set \( \delta = \sqrt{d} + 1/2 \). It is well known that one can construct a set \( I \) of \( O(1/\delta^{d-1}) \) points on the sphere \( S \) such that for any point \( x \) on \( S \), there exists a point \( y \in I \) such that \( ||x - y|| \leq \delta \) (see [23]). We process \( P \) into a data structure that can answer \( \varepsilon \)-approximate nearest-neighbor queries. For a query point \( q \), let \( \varphi(q) \) be the point of \( P \) returned by this data structure. For each point \( y \in I \), we compute \( \varphi(y) \) using this data structure. We return the set \( Q = \{\varphi(y) \mid y \in I\} \).

Proof of correctness. We claim that \( Q \) is an \( \varepsilon \)-kernel of \( P \).

For simplicity, we first prove the claim under the assumption that \( \varphi(y) \) is the exact nearest-neighbor of \( y \) in \( P \).

Fix a direction \( u \in S^{d-1} \). Let \( \sigma \in P \) be the point that maximizes \( \langle u, p \rangle \) over all \( p \in P \). Suppose the ray emanating from \( \sigma \) in direction \( u \) hits \( S \) at a point \( x \). We know that there exists a point \( y \in I \) such that \( ||x - y|| \leq \delta \).

If \( \varphi(y) = \sigma \), then \( \sigma \in Q \) and

\[
\max_{p \in P} \langle u, p \rangle - \max_{q \in Q} \langle u, q \rangle = 0.
\]

Now suppose \( \varphi(y) \neq \sigma \). Let \( B \) be the \( d \)-dimensional ball of radius \( ||y - \sigma|| \) centered at \( y \). Clearly, \( \varphi(y) \in B \). Let us denote by \( z \) the point on the sphere \( \partial B \) that is hit by the ray emanating from \( y \) in direction \( -u \). Let \( w \) be the point on \( \partial B \) such that \( zw \parallel u \) and \( h \) the point on \( \partial B \) such that \( yh \parallel \sigma u \); see Figure 1.

![Figure 1. Correctness of the algorithm.](image)

Moreover, we have

\[
\langle u, \sigma \rangle - \langle u, \varphi(y) \rangle \leq \langle u, \sigma \rangle - \langle u, z \rangle = ||z - w|| = ||u - w|| \tan(\angle u \sigma w) = ||u - w|| \tan(\angle u \sigma z/2) \leq (1/2) \cdot ||z - y|| \cdot \tan(\angle u \sigma z/2) \leq (1/2) \cdot ||z - y|| \cdot ||y - h|| \leq (1/2) \cdot ||z - y||^2 ||\sigma - h|| \leq (1/2) \cdot ||z - y||^2 \cdot ||\sigma - h|| \leq (1/2) \cdot ||z - y||^2 ||\sigma - z|| \leq (1/2) \cdot \delta^2 (1 - \delta) ||\sigma - z|| \geq 1 \leq \delta^2 = \alpha \varepsilon (\delta \leq 1/2).
\]

Thus, we can write

\[
\max_{p \in P} \langle u, p \rangle - \max_{q \in Q} \langle u, q \rangle \leq \langle u, \sigma \rangle - \langle u, \varphi(y) \rangle \leq \alpha \varepsilon.
\]

Similarly, we have \( \min_{p \in P} \langle u, p \rangle - \min_{q \in Q} \langle u, q \rangle \geq -\alpha \varepsilon \).

The above two inequalities together imply that \( \omega(u, P) \geq \omega(u, Q) \geq 2\alpha \varepsilon \). Hence \( \omega(u, Q) \geq (1 - \varepsilon) \omega(u, P) \), for any \( u \in S^{d-1} \). This implies that \( Q \) is an \( \varepsilon \)-kernel of \( P \).

We next sketch how to argue that \( Q \) is a \((2\varepsilon/\delta)\)-kernel of \( P \) if \( \varphi(y) \) is a \( \delta \)-approximate nearest-neighbor of \( y \) in \( P \), where \( \delta = \varepsilon/2(\sqrt{d} + 1) = O(\varepsilon) \). Let \( B' \) be a \( \delta \)-expansion of \( B \), i.e., a ball of radius \( (1 + \delta) ||y - \sigma|| \) centered at \( y \), and let \( z' \) be the point on \( \partial B' \) that is hit by the ray emanating from \( y \) in direction \( -u \). Since \( \varphi(y) \in B' \), it follows from our above discussion that

\[
\max_{p \in P} \langle u, p \rangle - \max_{q \in Q} \langle u, q \rangle \leq \langle u, \sigma \rangle - \langle u, z' \rangle = \langle u, \sigma \rangle - \langle u, z \rangle + ||z - z'|| \leq \alpha \varepsilon + \delta ||y - \sigma|| \leq 2\alpha \varepsilon.
\]

The last inequality follows from the fact that \( ||y - \sigma|| \leq 2(\sqrt{d} + 1) \). Similarly, \( \min_{p \in P} \langle u, p \rangle - \min_{q \in Q} \langle u, q \rangle \geq -2\alpha \varepsilon \). Therefore, following a similar argument as given above, we have \( \omega(u, Q) \geq (1 - 2\varepsilon) \omega(u, P) \) as desired.

Running time. Note that a straightforward implementation (the one that answers a nearest-neighbor query by comparing the distances to all the points) of the algorithm runs in \( O(n/\varepsilon^{(d-1)/2}) \) time. By first computing an \((\varepsilon/2)\) kernel \( Q' \) of \( P \) of size \( O(1/\varepsilon^{d-1}) \), as in Agrawal et al. [5], and then applying the above algorithm to compute an \((\varepsilon/3)\)-kernel \( Q \) of \( Q' \), we compute an \( \varepsilon \)-kernel \( Q \) of \( P \) in \( O(n + 1/\varepsilon^{(d-1)/2}) \) time. Although this running time is the same as the one in Agrawal et al. [5], our algorithm is more practical than theirs, which, for each \( y \in I \), computes the nearest point on the convex hull of \( Q' \). Each query in their algorithm required an invocation of an abstract optimization problem [16]. In practice our algorithm may take advantage of known approximate nearest-neighbor searching data structures, such as BBT-trees [8], and the running time would be bounded by \( O(n \log n + \log n/\varepsilon^{(3d-1)/2}) \). In fact, using data structure described in [14], which appears in this proceedings, the running time can be improved to \( O(n + 1/\varepsilon^{d-3/2}) \).

Experimental results. We implemented our \( \varepsilon \)-kernel algorithm and tested its performance on a variety of inputs. We measure the quality of an \( \varepsilon \)-kernel \( Q \) of \( P \) as the maximum relative error in the directional width of \( P \) and \( Q \). Since it is hard to compute the maximum error over all directions, we sampled a set \( \Delta \) of 1000 directions in \( S^{d-1} \) and
computed the maximum relative error with respect to those directions, i.e.,
\[
\text{err}(Q, P) = \max_{u \in \Delta} \frac{\omega(u, P) - \omega(u, Q)}{\omega(u, P)}.
\] (3)

We performed two sets of experiments:

(i) We fixed a real value \( \varepsilon \), and checked how the kernel size increased, as a function of input size and dimension, to ensure that \( \text{err}(Q, P) \leq \varepsilon \). We fixed \( \varepsilon \) to be 0.05 in our experiments.

(ii) We fixed the input size and checked how \( \text{err}(Q, P) \) decreased as the kernel size increased.

We implemented the constant-factor approximation algorithm by Barequet and Har-Peled [9] for computing the minimum-volume bounding box to convert \( P \) into an \( \varepsilon \)-fat set. In order to generate \( I \), we use the standard coordinate frame on \( S^{d-1} \), choose a parameter \( k \), and draw a grid \( \times \cdots \times k \) on \( S^{d-1} \). There are two weaknesses of this approach — the algorithm is not regular and the size of \( I \) is of the form \( \varepsilon^{-d} \). Finally, we used the ANN library [22] for answering approximate nearest-neighbor queries, and we set its relative error to \( 0.01 \).

We used three different types of synthetic inputs in dimensions 2–8 and a few large 3D geometric models [21]:

(i) points uniformly distributed on a sphere (sphere);

(ii) points uniformly distributed on a cylindrical surface (cylinder);

(iii) clustered point sets (clustered), consisting of 20 equal-sized clusters whose centers are uniformly distributed in the unit square and radii uniformly distributed between \([0, 0.2]\);

(iv) 3D geometric models: bunny \((\sim 36K \text{ points})\), dragon \((\sim 438K \text{ points})\), and buddha \((\sim 544K \text{ points})\).

Since the algorithm chooses mostly convex hull vertices as kernels, we used convex sets for synthetic data sphere and cylinder. Input points lying on a sphere can be regarded as the worst-case example. The relative size of the kernel would be smaller for arbitrary point sets (see, e.g., clustered).

All experiments of this paper were conducted on a Dell PowerEdge 650 server with a 3.00 GHz Pentium IV processor and 3GB memory, running Linux 2.4.20. Running time is measured in seconds. Tables 1–3 show our results on the first experiment, and Figure 2 shows the results of the second experiment. In Table 1, we have decomposed the running time into two components: (i) preprocessing time and (ii) query time. The former includes the time spent in converting \( P \) into a flat set and in preprocessing \( P \) for approximate nearest-neighbor queries, and it increases linearly with the size of the input. The latter includes the time spent in computing \( \phi(x) \) for \( x \in I \), and it depends on the size of \( I \). Our algorithm worked extremely well in low dimensions \((d \leq 4)\) both in terms of size and running time. To achieve an error below 0.05, the size of the \( \varepsilon \)-kernel is usually few hundreds and the running time is just a few seconds (except for the case of sphere). As Table 3 illustrates, the performance is even better on 3D geometric models. In higher dimensions, both our algorithm and the ANN library ran into the curse of dimensionality: the size and running time grow exponentially in \( d \). However, we expect that it might still be useful as a first prune in practice, and the running time can be improved by a better tradeoff between the preprocessing and query time. Our algorithm still performed well on the input clustered in dimensions 6 and 8. Figure 2 shows that, as the size of the computed \( \varepsilon \)-kernels becomes larger, its quality improves drastically while the kernel size is small, e.g., in the range \([0, 200]\). This phenomenon suggests that in practice it is most beneficial to keep the size of \( \varepsilon \)-kernels small.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Input Size</th>
<th>d = 2</th>
<th>d = 4</th>
<th>d = 6</th>
<th>d = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>sphere</td>
<td>10,000</td>
<td>10</td>
<td>994</td>
<td>7,113</td>
<td>6,985</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>10</td>
<td>1,824</td>
<td>22,202</td>
<td>57,276</td>
</tr>
<tr>
<td>cylinder</td>
<td>10,000</td>
<td>6</td>
<td>367</td>
<td>3,834</td>
<td>6,350</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>6</td>
<td>859</td>
<td>8,857</td>
<td>49,203</td>
</tr>
<tr>
<td>clustered</td>
<td>10,000</td>
<td>8</td>
<td>235</td>
<td>718</td>
<td>2,402</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>12</td>
<td>336</td>
<td>1,483</td>
<td>7,614</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>12</td>
<td>140</td>
<td>1,554</td>
<td>13,781</td>
</tr>
</tbody>
</table>

Table 2. Sizes of \( \varepsilon \)-kernels for various synthetic data sets, \( \varepsilon < 0.05 \).

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Input Size</th>
<th>Running Time</th>
<th>Kernel Size</th>
<th>Diameter Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>bunny</td>
<td>35,941</td>
<td>0.17</td>
<td>0.01</td>
<td>67</td>
</tr>
<tr>
<td>dragon</td>
<td>437,645</td>
<td>2.44</td>
<td>0.01</td>
<td>69</td>
</tr>
<tr>
<td>buddha</td>
<td>543,652</td>
<td>2.87</td>
<td>0.01</td>
<td>68</td>
</tr>
</tbody>
</table>

Table 3. Performance on various 3D geometric models with \( \varepsilon < 0.05 \). We also computed the relative error in \( \text{diam}(Q) \) and \( \text{diam}(P) \) as another measure of the quality of \( Q \). Running time is measured in seconds.

3. Shape Fitting

Let \( P \) be a set of points in \( \mathbb{R}^d \). We describe a very simple incremental algorithm for fitting a simple shape through a point set. We show its versatility and efficiency by applying it to: (i) minimum enclosing cylinder in \( \mathbb{R}^3 \), (ii) minimum-width annulus in \( \mathbb{R}^2 \), and (iii) minimum-volume bounding box in \( \mathbb{R}^3 \).

3.1 An incremental algorithm

Let \( \Pi \) be an infinite family of simple shapes, e.g., set of all cylinders, set of all boxes, or set of all annuli. Let \( \mu: \Pi \to \mathbb{R} \) be a measure function, and let \( A_{\text{opt}}(R, \Pi) \) be an algorithm that returns an optimal shape in \( \Pi \) enclosing a set \( R \) of points, i.e., \( A_{\text{opt}}(R, \Pi) = \arg\min_{\tau \in \Pi} \mu(\tau) \). Assume that \( \Pi \) is closed under translation and uniform scaling. For a shape \( \pi \in \Pi \), let \( (1 + \varepsilon)\pi \) denote the scaled version of \( \pi \). We here assume that each shape in \( \Pi \) has a center with respect to which we perform the scaling.

Figure 3 describes the incremental approximation algorithm that computes \( A_{\text{opt}}(P, \Pi) \) for an input point set \( P \), i.e., it returns a shape \( \pi \in \Pi \) such that \( P \subseteq \pi \) and \( \mu(\pi) \leq \mu((1 + \varepsilon)A_{\text{opt}}(P, \Pi)) \). Figure 4 for an illustration of a few iterations of the algorithm. In the algorithm, \( c \) is a constant depending on \( \Pi \) that, roughly speaking, is the number
Table 1. Running time for computing \(\varepsilon\)-kernels of various synthetic data sets, \(\varepsilon < 0.05\). \(\text{Prepr}\) denotes the preprocessing time, including converting \(P\) into a fat set and building ANN data structures. \(\text{Query}\) denotes the time for performing approximate nearest-neighbor queries. Running time is measured in seconds.

![Table Image](image_url)

Figure 2. Approximation errors under different sizes of computed \(\varepsilon\)-kernels. (a) sphere, (b) cylinder, (c) clustered, and (d) various geometric models. All synthetic inputs had 100,000 points.

![Figure Image](image_url)

Figure 3. An incremental algorithm for shape fitting.

**APPROX\_SHAPE\_FITTING** \((P, \Pi, \varepsilon)\)  
\(R \subseteq P\): arbitrary set of size \(\min\{|P|, c\}\)  
while \((\pi = (1 + \varepsilon)A_{\text{opt}}(R, \Pi)) \not\supseteq R\)  
\(p = \arg\max_{q \in P} d(q, \pi)\)  
\(R = R \cup \{p\}\)  
end while

of parameters required to specify an element in \(\Pi\). For example, if \(\Pi\) is a set of annuli in \(\mathbb{R}^2\) (resp. boxes, cylinders, slabs in \(\mathbb{R}^3\)), then \(c\) is 4 (resp. 6, 5, 4). The third line in the algorithm means that \(p \in P\) is the point that requires \(\pi\) to be scaled the most in order to be covered.

### 3.2 Analysis

Assume that \(\varepsilon \leq 1/2\). We prove that if computing \(A_{\text{opt}}(P, \Pi)\) can be formulated as computing the minimum extent of a set of \(k\)-variate linear functions (over an appropriate subset of the domain), the algorithm will terminate in \(O(1/\varepsilon^{(k-1)/2})\) steps. Instead of proving the claim in general, we prove it for the case in which \(\Pi\) is the set of all slabs — region lying between two parallel hyperplanes. As shown by Agarwal et al. [5], the other cases of the smallest enclosing box, cylinder, spherical or cylindrical shells can be reduced to computing...
the smallest slab containing a set of points in higher dimensions.

Clearly, when the algorithm terminates, it returns an \( \varepsilon \)-approximation of the optimal minimum-width slab. Next we show that the algorithm indeed terminates in \( O(1/\varepsilon^{(d-1)/2}) \) steps, by exploiting our argument for the correctness of our algorithm for computing \( \varepsilon \)-kernels.

Let \( \sigma \) be the affine transform from Lemma 2.1 so that \( \tau(P) \) is an \( \alpha \)-fat point set in \( \mathbb{R}^d \) satisfying \( \alpha C \subseteq \text{conv}(\tau(P)) \subseteq C \) where \( C = [-1,1]^d \). For any slab \( \sigma \in \mathbb{R}^d \), \( \tau(\sigma) \) is also a slab in \( \mathbb{R}^d \), and if \( \tau(\sigma) \) is the point farthest away from \( \sigma \) in \( P \), then \( \tau(\sigma) \) is the point farthest away from \( \tau(\sigma) \) in \( \tau(P) \).

Let \( S \subseteq \mathbb{R}^d \) be the sphere of radius \( \sqrt{d} + 1 \) centered at the origin. Let \( \delta_1 = \sqrt{2\alpha/2} \leq 1/2 \) and \( \delta_2 = \delta_1/2 \). We deploy a set of \( O(1/\delta_2^{d-1}) = O(1/\varepsilon^{(d-1)/2}) \) sentiment points \( I \) on the sphere such that for any point \( x \) on \( S \), there exists \( y \in I \) satisfying \( ||x - y|| \leq \delta_2 \). At the beginning, all sentiment points are unmarked. For any point \( q \) on the sphere \( S \), define \( \mathcal{N}(q) = \{w \in I : ||w - q|| \leq \delta_2\} \).

At some stage of the algorithm, suppose \( R \subseteq P \) is the current subset of points in consideration. The algorithm now computes a slab \( \sigma^* \) containing \( R \). Suppose that the slab does not terminate, i.e., the \( \varepsilon \)-expansion of \( \sigma^* \) does not contain \( P \). Let \( p \in P \cap \sigma^* \) be the point farthest away from \( \sigma^* \), then \( \tau(p) \in \tau(P) \setminus \tau(\sigma^*) \) is also the point farthest away from \( \tau(\sigma^*) \). Let \( u \) be one of the two directions normal to \( \tau(\sigma^*) \) so that \( \langle u, \tau(p) \rangle = \max_{q \in P \cap \sigma^*} \langle u, \tau(q) \rangle \). Let \( x \in S \) be the point that is hit by the ray emanating from \( \tau(p) \) along \( u \). Since \( \tau(p) \) is the extreme point in direction \( u \), the ball of radius \( ||x - \tau(p)|| \) centered at \( x \) does not contain any point of \( \tau(p) \). Therefore \( \tau(p) \) is the nearest neighbor of \( x \) in \( \tau(P) \).

We now mark all the sentinel points in \( \mathcal{N}(x) \). Let us remark that marking sentinel points is just for the analysis.

Clearly, \( \mathcal{N}(x) \neq \emptyset \) by our choice of \( I \). Moreover, we claim that no points in \( \mathcal{N}(x) \) were previously marked. Indeed, otherwise let us pick any marked point \( \xi \) from \( \mathcal{N}(x) \). Then at some previous step, the algorithm added a point \( p' \in P \) to \( R \) such that \( \tau(p') \) is the farthest point in \( \tau(P) \) along some direction \( u' \) and \( \xi \in \mathcal{N}(x) \), where \( x' \) is the point on the sphere \( S \) hit by the ray emanating from \( \tau(p') \) along \( u' \) (see Figure 5). Thus we have \( \tau(p') \) is the nearest neighbor of \( x' \) in \( \tau(P) \), and

\[
||x - x'|| \leq ||x - \xi|| + ||\xi - x'|| \leq \delta_1.
\]

Now proceeding as in the proof of (2) in Section 2, we have

\[
\langle u, \tau(p) \rangle - \langle u, \tau(p') \rangle \leq \delta_1^2 = \alpha/2.
\]

Since \( \alpha C \subseteq \text{conv}(\tau(P)) \), we have \( \omega(u, \tau(P)) \geq \alpha/2 \). Let \( x \) be any point on the center hyperplane of \( \tau(\sigma^*) \). Since \( \tau(p) \in \tau(P) \) is the point farthest away from \( \tau(\sigma^*) \), we obtain

\[
\langle u, \tau(p) \rangle - \langle u, x \rangle \geq \omega(u, \tau(P))/2 \geq \alpha.
\]

It follows from (4) and (5) that

\[
\langle u, \tau(p) \rangle - \langle u, x \rangle \geq \alpha/2.
\]

In view of (4) and (6), we see that the \( \varepsilon \)-expansion of \( \tau(\sigma^*) \) contains \( \tau(p) \) and hence \( \tau(P) \). In other words, the \( \varepsilon \)-expansion of \( \sigma^* \) has already contained \( P \), a contradiction.

We can then conclude that at least one previously unmarked sentinel point is marked in each iteration of the algorithm, which implies that the algorithm terminates within \( O(1/\varepsilon^{(d-1)/2}) \) iterations.

**Theorem 3.1.** Let \( P \) be a set of \( n \) points in \( \mathbb{R}^d \), and let \( \varepsilon > 0 \) be a parameter. The incremental algorithm computes an \( \varepsilon \)-approximation of the minimum-width slab containing \( P \) in at most \( O(1/\varepsilon^{(d-1)/2}) \) iterations.

A similar statement bounding the number of iterations by a term independent of \( n \) can be made for minimum enclosing cylinder, bounding box, annulus, cylindrical shell, etc. We omit the proof from this extended abstract. Note that if the number of iterations is \( k \), then the running time is \( O(nk) \) plus the time needed to solve \( k \) instances of problems of size at most \( O(k) \).

**Remark.** A similar incremental algorithm was proposed by Badoiu et al. [12] for computing the smallest enclosing ball of a point set and by Kumar and Yildirim [20] for computing the smallest enclosing ellipsoid. Note that both these problems can be formulated as convex programs, unlike the problems we address here. Thus our proof of convergence is quite different from the proofs in these papers.
Table 4. A comparison of the performance of various approximating algorithms for computing the smallest enclosing cylinder. The numbers of iterations performed by the incremental algorithm are in parenthesis. Running time is measured in seconds.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Input Size</th>
<th>Running Time</th>
<th>Output Radius</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>APRR</td>
<td>CORR</td>
</tr>
<tr>
<td>$h = 2.0$</td>
<td>100,000</td>
<td>100,000</td>
<td>20,358</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>225,799</td>
<td>1.14</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>12.19</td>
</tr>
<tr>
<td>$h = 300.0$</td>
<td>100,000</td>
<td>16,220</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>12.19</td>
</tr>
<tr>
<td>$h = 200.0$</td>
<td>100,000</td>
<td>16,456</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>12.19</td>
</tr>
<tr>
<td>bunny</td>
<td>36,547</td>
<td>65,058</td>
<td>0.40</td>
</tr>
<tr>
<td>dragon</td>
<td>437,645</td>
<td>800,777</td>
<td>2.04</td>
</tr>
<tr>
<td>buddha</td>
<td>543,652</td>
<td>1,126,27</td>
<td>3.58</td>
</tr>
</tbody>
</table>

3.3 Experimental Results

In this section, we study the empirical performance of our incremental approach for shape fitting by applying it to (i) smallest enclosing cylinder, (ii) minimum-width annulus, and (iii) minimum-volume box.

Smallest enclosing cylinder. In the incremental algorithm described in Figure 3, we need to implement the $A_{opt}$ algorithm for computing the smallest enclosing cylinder. The exact algorithm for computing the smallest enclosing cylinder are not only expensive (the best known exact algorithm requires near-cubic time [2]), they also involve complicated machineries such as parametric searching and computing the roots of high-degree polynomials. Therefore, we implemented a simpler variant of the approximation algorithm by Aragwal et al. [2], which computes an $e$-approximation of the smallest enclosing cylinder in $O(n/e^2)$ time. This algorithm requires a procedure for computing smallest enclosing disks, and we have used Gärtner’s code [17] for this purpose. We omit the implementation detail of this algorithm from here.

We performed experiments on both synthetic inputs and real geometric models. The synthetic input is generated by uniformly sampling on cylindrical surfaces of fixed radius $r = 1.0$ and various heights $h$. We ran four types of algorithms on the inputs:

(i) approximation algorithm directly (APPR);
(ii) computing $e$-kernels as described in Section 2 and applying APPR to kernels (CORE);
(iii) incremental algorithm with APPR on the small problems for each iteration (INCR);
(iv) computing $e$-kernels first and applying INCR to kernels.

We observed that the fourth algorithm improves very little over CORE and performs worse than INCR. We therefore compare the performance of the first three algorithms, and summarize our results in Table 4. We paid special attention to the number of iterations required by INCR and included it in Table 4 (numbers in parenthesis).

As expected, CORE is much faster than APPR, and the output quality of the two algorithms is roughly the same. More interestingly, we observed that INCR always outperforms CORE. Intuitively, the reason might be explained as follows: in practice, the size of an $e$-kernel can be much smaller than its theoretical bound, and INCR can find such a kernel on the fly, instead of computing it in advance using the algorithm for computing kernels. In fact, as shown in Table 4, the number of iterations of the incremental algorithm never exceeded 10 in all experiments we have run.

Minimum-width annulus. We implemented a simple $O(n^3)$ algorithm for computing $A_{opt}$ exactly; the detail is omitted from here. We ran our incremental algorithm on point sets uniformly sampled from annuli of fixed inner radius $r = 1.0$ and various widths $w$. The experimental results are shown in Table 5. Note that when the point set is almost circular, which is probably the interesting case in practice, the algorithm has very few iterations. This is what we would expect for the incremental algorithm.

Table 5. Performance of the incremental algorithm for computing the minimum-width annulus. The numbers of iterations performed by the algorithm are in parenthesis. Running time is measured in seconds.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Input Size</th>
<th>Running Time</th>
<th>Output Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>w = 0.05</td>
<td>$10^4$</td>
<td>0.01 (2)</td>
<td>0.004</td>
</tr>
<tr>
<td>w = 0.50</td>
<td>$10^4$</td>
<td>0.03 (2)</td>
<td>0.004</td>
</tr>
<tr>
<td>w = 5.00</td>
<td>$10^6$</td>
<td>0.07 (9)</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Minimum-volume box. We used an approximation algorithm (BBH) by Barouqet and Harkoo [9] for computing...
A method. We compared our incremental algorithm (INCR) with BBX on various inputs. The input box \((a, b, c)\) is generated by uniformly sampling from the boundary of a 3D box of side lengths \(a, b, c\). The results are summarized in Table 6. Note that on inputs that are closer to being spherical, INCR is sometimes slightly better than INCR, whereas INCR is clearly faster on inputs that are far from spherical.

### 4. Kinetic Data Structures

In this section, we apply the \(\varepsilon\)-kernel algorithm to maintain an approximation to the minimum orthogonal bounding box and the convex hull of moving points in \(\mathbb{R}^2\), and show empirically how this technique impacts the efficiency of kinetic data structures.

**Maintaining minimum orthogonal bounding box.** We maintain the minimum orthogonal bounding box of a point set by keeping track of the two extremal points along each dimension using kinetic tournament trees \([10]\) we have implemented. Let \(p_x\) (resp. \(p_y\)) denote the trajectory, a curve in \(\mathbb{R}^2\), of the \(x\)-coordinate (resp. \(y\)-coordinate) of the point \(p \in P\). Let \(F_x = \{p_x \mid p \in P\}\) and \(F_y = \{p_y \mid p \in P\}\). We compute \(\varepsilon\)-kernels \(G_x\) and \(G_y\) of \(F_x\) and \(F_y\) respectively, using linearization and duality as described in \([5]\) and the algorithm described in Section 2. We set

\[
Q = \{p \mid p_x \in G_x \text{ or } p_y \in G_y\},
\]

and maintain a bounding box of \(Q\) using kinetic data structures.

Our experiments were conducted on two types of synthetic motions: (i) linear motion, and (ii) quadratic motion. For a linear motion \(x = v \cdot t + x_0\) along a specific dimension, \((v, x_0) \in \mathbb{R}^2\) is uniformly sampled from \(S^1\), with small perturbations. For a quadratic motion along a specific dimension, it is generated by randomly shifting a fixed quadratic function, i.e., \(x = a \cdot (t - \beta)^2 + v \cdot (t - \beta) + x_0\), where \((a, v, x_0)\) is fixed and \(\beta\) is random, and introducing small perturbations. These synthetic motions tend to have a large number of external events (i.e., topological changes in the structure maintained) compared to the input size. We also fine-tuned the parameters so that most of the external events happened during the time interval \([-10, 10]\).

We present our experimental results for the 2D case only; similar results were also observed in higher dimensions. We studied two measures in our experiments: (i) relative error in the size of the bounding box as a function of time, and (ii) number of kinetic events.

---

**Figure 6.** Quality of kernels for the minimum orthogonal bounding box of 100,000 moving points over the time interval \([-10, 10]\). (a) Linear motion, (b) quadratic motion.

**Figure 7.** Distribution of kinetic events for maintaining the minimum orthogonal bounding box of 100,000 moving points over the time interval \([-10, 10]\). (a) Exact algorithm, (b) using \(\varepsilon\)-kernels with quality \(> 0.96\).
The quality of a kernel at a time $t$ is measured by
\[ \min \left\{ \frac{\|C_G(t)\|}{\|C_R(t)\|}, \frac{\|C_Q(t)\|}{\|C_R(t)\|} \right\}, \]
where $C$ is as defined in (1) of Section 1. We plotted the quality of the computed kernel over time in Figure 6. It was observed that a kernel of size at most 40 is sufficient to guarantee the quality above 0.98 over time for a set of 100,000 moving points with linear or quadratic trajectories. Figure 7 compares how kinetic events were distributed over time for the input and its kernel. It can be seen that the number of kinetic events was significantly reduced (by more than 99.8% on inputs of size 10,000) with very little sacrifice ($< 0.034$ on the quality of the box maintained by choosing a kernel of size $\leq 27$).

Maintaining convex hull. The algorithm of Basch et al. [10] for maintaining the convex hull of a point set involves a set of sophisticated certificates and maintenance operations. Instead of using their algorithm, we maintain the convex hull by maintaining a simple triangulation of the point set over time, which is much simpler and relatively efficient in practice. We compute an $\epsilon$-kernel $Q$ of the moving point set $P$ as described in [5], using the algorithm of Section 2 as a subroutine.

We tested the algorithm for linear motion; quadratic motion would entail solving high-degree polynomial equations. The initial position of a point is randomly chosen on the unit circle, and the velocity is a random unit vector. We also tried various other synthetic motions, e.g., the speeds or the directions of the velocities are clustered. They yielded similar experimental results as reported below.

We measured the quality of a kernel at time $t$ using (3) where $\Delta$ is the set of 200 random directions in $\mathbb{R}^2$. We also computed the ratios between the width and the diameter of the kernel and of the input over time. We plotted our results in Figure 8. It was observed that a kernel of size at most 130 is sufficient to keep the quality above 0.9 over time for input of size 10,000. The approximation of the width and the diameter is even better. We also counted the number of kinetic events for maintaining $\conv(P(t))$ and $\conv(Q(t))$. Since we weren’t using the best known KDS for maintaining the convex hull and the focus was on demonstrating that the KDS for $\conv(Q)$ processes much fewer events that that for $\conv(P)$, we only count the number of external kinetic events, i.e., the events at which the combinatorial structure of the convex hull changes. We plotted the distribution of external kinetic events in Figure 9 for both the input and its kernel. As can be seen, we were able to process significantly fewer kinetic events while maintaining a good approximation of the convex hull as the points move.

5. Conclusions

In this paper we presented a simple and practical algorithm for computing the $\epsilon$-kernel of a set of points in $\mathbb{R}^d$ and studied its empirical performance on various inputs. We then described an incremental algorithm for fitting various shapes on a point set. Although we can only prove a weaker bound on the worst-case running time of this algorithm, our empirical studies show that it performs very well in practice. We also applied our $\epsilon$-kernel algorithm to maintain various extent measures of a set of moving points, which significantly improved the performance of kinetic data structures.

An interesting open problem is to understand why the incremental algorithm for shape fitting works so well in practice. For example, in the smallest enclosing cylinder problem, can one prove a better bound on the number of iterations of the incremental algorithm that does not depend exponentially on $d$ (the $1/\epsilon^2 \log d$ bound is not hard to prove)?

Acknowledgments

The authors would like to thank Sariel Har-Peled and the anonymous referees for helpful comments.

References

[14] T. M. Chan, Faster core-set constructions and data stream algorithms in fixed dimensions, this proceedings.
Figure 8. Quality of kernels for the convex hull of 10,000 moving points over the time interval $[-2,2]$. (a) Quality over 200 random directions, (b) quality of width, (c) quality of diameter.

Figure 9. Distribution of external kinetic events for maintaining the convex hull of 10,000 moving points over the time interval $[-2,2]$. (a) Exact convex hull, (b) using $c$-kernels.