Pattern Matching for Sets of Segments*

Alon Efrat† Piotr Indyk‡ Suresh Venkatasubramanian§

Abstract

In this paper we present algorithms for a number of problems in geometric pattern matching where the input consists of a collection of orthogonal segments in the plane. Such collections arise naturally from problems in mapping buildings and robot exploration.

We propose a new criterion for segment similarity called the *coverage measure*, and present efficient algorithms for maximizing it between sets of axis-parallel segments under translations. In the general case, we present a procedure running in time $O(n^3 \log^2 n)$, and for the case when all segments are horizontal, we give a procedure that runs in time $O(n^2 \log^2 n)$. Here n is the number of segments. In addition, we show that an ε-approximation to the Hausdorff distance between two sets of horizontal segments under vertical translations can be computed in time $O(n^{3/2} \max(\text{poly}(\log M, \log n, 1/\varepsilon)))$. Here, M denotes the ratio of the diameter to the closest pair of points in the sets of segments (where pairs of points lie on different segments). These algorithms are significant improvements over the general algorithm of Agarwal et al. that required time $O(n^4 \log^2 n)$.

1 Introduction

Traditionally, geometric pattern matching employs as a measure of similarity the directed Hausdorff distance $h(A, B)$ defined as $h(A, B) = \max_{p \in A} \min_{q \in B} d(p, q)$ for two point sets A and B, where $d(p, q)$ is the Euclidean distance between p and q. However, when the patterns to be matched are line segments or curves (instead of points), this distance is less than satisfactory. It has been observed that measures like the Hausdorff measure that are defined on point sets are ill-suited as measures of curve similarity, because they ignore the directionality inherent in continuous curves.

This paper addresses problems in geometric pattern matching where the inputs are sets of axis-parallel line segments. We study two different measures in this context; one is the standard Hausdorff distance, and the other is a novel measure called the *coverage measure*, which captures the similarity between axis-aligned segments.

The motivation for considering instances of pattern matching where the input line segments are orthogonal comes from the domain of *mapping*, in which a robot is required to map the underlying structure of a building by moving inside the building, and "sensing" or "studying" its environment. In one such mapping project at the Stanford Robotics Laboratory[1] the robot is equipped with a laser range finder which measures the distance from the robot to its nearest neighbor in a dense set of directions in a horizontal plane. We call the resulting distance map a *picture*. Figure 1 shows the robot used at Stanford Robotics Laboratory for this purpose.

---

*A preliminary version of this paper appeared in [12].

†Computer Science Department, The University of Arizona **Email:** *alon@cs.arizona.edu*. Work supported in part by a Rothschild Fellowship and by DARPA contract DAAE07-98-C-L027

‡Computer Science Department, MIT. **Email:** *indyk@cs.stanford.edu*.

§AT&T Labs – Research. **Email:** *suresh@research.att.com*.

[1] The interested reader can find more information at the URL `http://underdog.stanford.edu`

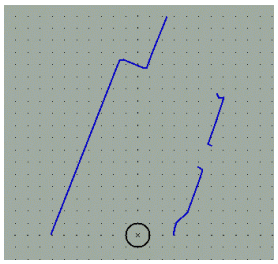


Figure 1: Left: A typical "picture" obtained by the robot of a corridor (after segmentation). Right: The corridor itself, and the robot with the laser range finder installed on it.

During the mapping process, the robot must merge into a single map the series of pictures that it captures from different locations in the building. Since the dead reckoning of the robot is not very accurate, it cannot rely solely on its motion to decide how the pictures are placed together. Thus, we need a matching process that can align (by using overlapping regions) the different pictures taken from different points of the same environment. In addition, we need to determine whether the robot has returned to a point already visited. We make the reasonable assumption that walls of buildings are almost always either orthogonal or parallel to each other, and that these walls are frequently by far the most dominant objects in the pictures. This is especially significant in the case that the robot is inside a corridor, where there is insufficient detail for good registration. In some cases most of the pictures consist merely of two walls with a small number of other segments. See Figure 1 for a typical picture and the real region that the laser range finder senses.

This application suggests the study of matching sets of horizontal and vertical segments. Observe that we may restrict ourself to alignments under *translation*, as it is easy to find the correct rotation for matching sets of orthogonal segments. Formally, let $A = \{a_1 \dots a_n\}$ and $B = \{b_1, \dots b_n\}$ be two sets of axis-aligned line segments in the plane, each consists of pairwise-disjoint segments, and let $\varepsilon > 0$ be a given parameter. A point p of a horizontal (resp. vertical) segment $b \in B$ is *covered* if there is a point of a horizontal (resp. vertical) segment $a \in A$ whose distance from p is at most ε, where the distance is measured using the ℓ_∞ norm. A point on a horizontal (resp. vertical) segment can be covered only by a point of another horizontal (resp. vertical) segment. Let $w(A, B)$ denote the collection of sub-segments of B consisting of covered points. Let $Cov_\varepsilon(A, B)$ be the total length of the segments of $w(A, B)$. The *maximum coverage problem* is to find a translation t^* in the translation plane that maximizes $Cov_\varepsilon(t) = Cov_\varepsilon(A, t + B)$. Here ε is a parameter specified by the user based on the physical model. To the best of our knowledge, this similarity measure is novel. The coverage measure is especially relevant in the case of long segments e.g. inside a corridor, when we might be interested in matching portions of long segments to portions of other segments.

Our Results In Section 2 we present an algorithm that solves the Maximum Coverage problem between sets of axis-parallel segments in time $O(n^3 \log^2 n)$ and the Coverage problem between horizontal segments in time $O(n^2 \log n)$. Note that the known algorithms for matching arbitrary sets of line segments are much slower. For example, the best known algorithm for finding a translation that minimizes the Hausdorff Distance between two sets of n segments in the plane runs in time $O(n^4 \log n)$ [2]. We also show (Section 3) that the combinatorial complexity of the Hausdorff matching between segments is $\Omega(n^4)$, even if all segments are *horizontal*. This strengthens the bounds shown by Rucklidge [20], and demonstrates that our algorithms, much like the algorithms of [10, 11], are able to avoid having to examine each class of combinatorially different translations.

In Section 4 we consider the related problem of matching horizontal segments under vertical translations with respect to the directed Hausdorff distance. It has been observed that if horizontal translations of horizontal segments are allowed, then this problem is 3SUM-hard [4], indicating that finding a sub-quadratic algorithm may be hard. However, we present an ε-approximation algorithm running in time $O(n^{3/2}\max\{\log^c M, \log^c n, 1/\varepsilon^c))\}$, for some fixed constant c, which is sub-quadratic in most cases. Here, M denotes the ratio of the diameter to the closest pair of points in the sets of segments (where pairs of points lie on different segments).

It is interesting to note that the data structure presented in Section 2 can be used to slightly improve the result of Har-Peled *et al* [13], in which the following problem is addressed. Given a set of rectangles $\{R_1 \dots R_n\}$ and another rectangle R, find a translation t for which $t + R$ maximizes $\sum_i \text{area}(R_i \cap t + R)$. Their algorithm runs in $O(n^{3/2}\log^3 n)$ time. Using our data structure in their algorithm the runtime is slightly improved to $O(n^{3/2}\log^2 n)$.

2 Algorithms for maximum coverage

Let $A = \{a_1 \dots a_n\}$ and $B = \{b_1, \dots b_n\}$ be two sets of axis-parallel line segments in the plane, and let $\varepsilon > 0$ be a given parameter. We assume that no two segments of A intersect, and no two segments of B intersect.

2.1 Coverage with axis-parallel segments

The main result of this subsection is the following theorem:

Theorem 2.1. *A translation* t *that maximizes* $Cov_\varepsilon(A, t + B)$ *can be found in time* $O(n^3\log^2 n)$.

For the proof of this theorem, we need several lemmas and definitions. For a geometric object R let $X(R)$, the x-*span* of R, denote the interval of the x-axis between the leftmost and the rightmost point of R', where R' is the orthogonal projection of R on the x-axis.

Let $A^h \subseteq A$ (resp. $B^h \subseteq B$) be the set of horizontal segments of A (resp. B) and let $A^v \subseteq A$ (resp. $B^v \subseteq B$) be the set of vertical segments of B. Note that $Cov_\varepsilon(A, t+B) = Cov_\varepsilon(A^h, t+B^h) + Cov_\varepsilon(A^v, t+B^v)$. Let s be a non-vertical segment. We define the function $s(x) : \mathbb{R} \to \mathbb{R}$ as follows: For $x \notin X(s)$, $s(x) = 0$. For $x' \in X(s)$, $s(x')$ is the value of the y-coordinate of the intersection point of s and the vertical line $x = x'$. To emphasize that s defines a function, we refer to s as a *graph-segment* or *gsegment* for short. Informally speaking, our interest in gsegments results from the fact that the maximum is obtained along translations which lie on a horizontal line in the translation plane. These can be described as a sum of a set of gsegments, and thus tools for manipulating gsegments will be useful for computing the translation achieving maximum coverage.

Lemma 2.1. *Let* $P = \{(x_1, y_1), \dots (x_m, y_m)\}$ *be a point set. We can construct in time* $O(m\log^2 m)$ *a data structure for* P *such that for a query gsegment* s*, the point* $(x_k, y_k) \in P$ *maximizing the set* $\{s(x_i) + y_i \mid x_i \in X(s)$ *and* $1 \le i \le m\}$ *can be found in time* $O(\log^2 m)$.

Proof. If $X(P) \subseteq X(s)$, then (x_k, y_k) is clearly a vertex of the convex hull of P, and once the convex hull is computed, we can find (x_k, y_k) in time $O(\log m)$, as it is a standard linear programming query in a convex polygon. To answer the query in the case that $X(P) \not\subseteq X(s)$, we construct a balanced binary search tree $\Psi(P)$ on the set $\{x_1 \dots x_m\}$. For each node $\mu \in \Psi(P)$ let P_μ denote the points in the subtree of μ, and let X_μ denote the x-span of P_μ. We construct C_μ, the convex hull of P_μ, for each node μ of $\Psi(P)$. Once a

query gsegment s is given, we find a set of $O(\log m)$ nodes μ of $\Psi(P)$ with the property that $X_\mu \subseteq X(s)$, and $X_{\text{parent}(\mu)} \not\subseteq X(s)$. We perform the desired maximization query on every C_μ. The time required is clearly as claimed. □

Let $\mathcal{S} = \{e_1(\tau, x) \dots e_m(\tau, x)\}$ be a set of gsegments, whose locations change as a function of a time parameter τ in the following way: $e_i(\tau, \cdot) = a_i(\tau)b_i(\tau)$, where the location of the points $a_i(\tau), b_i(\tau)$ is given by

$$a_i(\tau) = (a_i^0.x,\ a_i^0.y + \alpha_i\tau) \quad \text{and} \quad b_i(\tau) = (b_i^0.x,\ b_i^0.y + \beta_i\tau)\ ,$$

where $a_i^0.x$, $a_i^0.y$, $b_i^0.x$, $b_i^0.y$, α_i, β_i are given constants. Thinking about τ as a time parameter, the endpoints of the segments move vertically at constant velocities as time elapses. Let $\mathit{max\_sum}(\mathcal{S}, \tau) = \max_{x\in\mathbb{R}} \Sigma_{e\in S} e(\tau, x)$.

Lemma 2.2. *In time $O(m \log m)$, we can construct a data structure $\mathcal{T}(\mathcal{S})$, such that given a time τ_0 and a query value $x \in \mathbb{R}$, $\Sigma_{e\in\mathcal{S}} e(\tau_0, x)$ can be computed in time $O(\log m)$.*

Proof. We construct a segment tree $\mathcal{T}(\mathcal{S})$ (see [7] for details) on the x-projections of the segments of $\mathcal{S}$ (note that their projections do not change in time). With each node μ of $\mathcal{T}(\mathcal{S})$ we maintain the interval I_μ on the x-axis associated with μ, and the subset $S_\mu \subseteq \mathcal{S}$ stored with μ. Let ℓ_L and ℓ_R be the left and right vertical lines passing through the endpoints x_L and x_R of I_μ. We can express the y-coordinate of the intersection point of $e_i(\tau, \cdot) \in S_\mu$ with ℓ_L and ℓ_R (resp.) by $a^L_{\mu,i}\tau + b^L_{\mu,i}$ and $a^R_{\mu,i}\tau + b^R_{\mu,i}$ (for $i = 1, \dots |S_\mu|$), when $a^L_{\mu,i}$, $b^L_{\mu,i}$, $a^R_{\mu,i} b^R_{\mu,i}$ are appropriate constants. Let x be a point on I_μ, and let $\alpha = (x - x_L)/(x_R - x_L)$. Hence at time τ,

$$e_i(\tau, x) = (1 - \alpha)\left(a^L_{\mu,i}\tau + b^L_{\mu,i}\right) + \alpha\left(a^R_{\mu,i}\tau + b^R_{\mu,i}\right)$$

Defining

$$A^L_\mu = \Sigma_i a^L_{\mu,i},\ \ B^L_\mu = \Sigma_i b^L_{\mu,i},\ \ A^R_\mu = \Sigma_i a^R_{\mu,i}\ ,\ A^R_\mu = \Sigma_i a^R_{\mu,i}\ ,$$

where the sum is taken over all i's for which $e_i \in S_\mu$. We have that

$$\Sigma_{e\in\mathcal{S}_\mu} e(\tau, x) \ =\ (1 - \alpha)\left(\tau A^L_\mu + B^L_\mu\right)\ +\ \alpha\left(\tau A^R_\mu + B^R_\mu\right) \tag{2.1}$$

We store $A^L_\mu, B^L_\mu, A^R_\mu, A^R_\mu$ with each node μ. Once a point x and a time τ are given, we find the $O(\log m)$ nodes μ of $\mathcal{T}(\mathcal{S})$ for which $x \in I_\mu$ (there is at most one such node at each level of $\mathcal{T}'$). For each, we evaluate the expression (2.1), and sum the results. Clearly this can be done in time $O(\log m)$, and the construction of $\mathcal{T}(\mathcal{S})$ can be done in time $O(m \log m)$. □

Lemma 2.3. *Let $\mathcal{S}$ be a set of m gsegments, as above. We can construct the data structure $\mathcal{D}(\mathcal{S})$, so that given a query time τ and a query gsegment $e'(x)$ (fixed in time), we can find $\max\{e'(x) + \sum_{e\in S} e(\tau, x) \mid x \in X(e')\}$. The time for constructing the structure and answering k queries is $O((m + k)\log^2 m)$, provided that the time parameter τ of each query is no smaller than the time of the previous query.*

Proof. We first explain how to construct the data structure for a fixed time τ_1. Before each query is posed to the data structure, we modify the data structure according to the time τ_i of that query.

Using a simple divide-and-conquer technique we construct in time $O(m \log m)$ the graph of the function $s(x) = \Sigma_{e\in\mathcal{S}} e(\tau_1, x)$. It is piecewise linear. Let $V(\mathcal{S})$ denote the set of vertices of this graph. Note that every such vertex has the same x-coordinate as an endpoint of one of the gsegments of $\mathcal{S}(\tau_1, \cdot)$, thus $|V(\mathcal{S})| = O(m)$. We construct the data structure $\Psi(V(\mathcal{S}))$ of Lemma 2.1 for $V(\mathcal{S})$. Since the convex hulls C_μ used by this data structure would change as a function of τ, we denote them as $C_\mu(\tau)$. We also construct the data structure $\mathcal{T}(\mathcal{S})$ of Lemma 2.2. Let $\mathcal{D}(\mathcal{S})$ denote the combined structure.

Clearly for every query gsegment $e'(\cdot,\cdot)$ the maximum $\max\{e'(\tau_1,x)+\Sigma_{e\in\mathcal{S}}\,e(\tau_1,x)\,|\,x\in X(e')\}$ is obtained at the x-projection of either a vertex of $V(\mathcal{S})$, or an endpoint of $e'(x)$. We use the data structure $\mathcal{T}(\mathcal{S})$ of Lemma 2.2 to find the value of $e'(\tau_1,x)+\Sigma_{e\in\mathcal{S}}\,e(\tau_1,x)$ at the endpoints of $e'(\tau,x)$. In order to handle the former case, we perform a query in $\Psi(V(\mathcal{S}))$. Thus answering a query is done (for fixed τ_1) in time $O(\log^2 m)$.

Once the next query is submitted (with a larger time $\tau_2\geq\tau_1$), we need to efficiently modify $\Psi(V(\mathcal{S}))$ to create $\Psi(V(\mathcal{S}))$ at τ_2. We increase τ gradually, while keeping track of the changes the data structure goes through. Note that as τ is increased from τ_1 to τ_2, the vertices of the graph of the function $s(x)$ move vertically at constant speeds, as the speed of each of them is the sum of $\leq m$ values which change linearly. We keep track of the changes that each convex hull $C_\mu(\cdot)$, stored at a node of the tree of $\Psi(V(\mathcal{S}))$, goes through.

It was shown ([5]) that the convex hull of such a set of k points moving vertically at constant speeds can go through $O(k)$ combinatorial changes. These changes can be tracked in a total of $O(k\log k)$ time by a simple divide-and-conquer algorithm: Split the vertices into two equal-cardinality subsets to the left and right of a vertical line, maintain recursively the convex hull of each subsets, and show that the common tangents to these hulls can go through $O(k)$ combinatorial changes which is trivial to tackle.

Since the total sizes of convex hulls in $\Psi(V(\mathcal{S}))$ is $O(m\log m)$, we need $O(m\log^2 m)$ time to maintain $\Psi(V(\mathcal{S}))$ as τ decreases from the first to the last query. □

Lemma 2.4. *Let $\mathcal{S}$ be as in Lemma 2.3. Then we can maintain max_sum$(\mathcal{S},\tau)$ under gsegment insertions or deletions in amortized time $O(\sqrt{m'}\log^2 m')$ per operation. In addition, we can maintain max_sum$(\mathcal{S},\tau)$ under a* time-increasing *step ($\tau\leftarrow\tau+\Delta$) (for $\Delta>0$) in $O(\sqrt{m'}\log m')$ time per update. Here m' is the maximum of m and the total number of operations performed on the set.*

Proof. A deletion of a gsegment e is resolved by adding the negation of e, so we direct our attention to the insertion of gsegments. We partition $\mathcal{S}$ into $\mathcal{S}_1$ and $\mathcal{S}_2$. The set $\mathcal{S}_1$ contains at most $\sqrt{m'}$ of the gsegments of $\mathcal{S}$. We define $\mathcal{S}_2=\mathcal{S}\setminus\mathcal{S}_1$. Each time a gsegment is inserted into $\mathcal{S}$, it is inserted into $\mathcal{S}_1$. Once the cardinality of $\mathcal{S}_1$ exceeds $\sqrt{m'}$, we set $\mathcal{S}_2$ to be $\mathcal{S}$, and empty $\mathcal{S}_1$. We construct the data structure $\mathcal{D}(S_2)$ of Lemma 2.3 for the gsegments of $\mathcal{S}_2$. Increasing τ is obtained as in Lemma 2.3.

Maintaining *max_sum*$(\mathcal{S},\tau)$ under insertions is obtained as follows: Assume the insertion happens at time τ_0. Once a gsegment is inserted into $\mathcal{S}_1$, we explicitly compute the graph of the function $s(x)=\sum_{e\in\mathcal{S}_1}e(\tau_0,x)$ which is piecewise linear of complexity $O(\sqrt{m'})$. With each gsegment $f(\tau_0,x)$ of this graph (not to be confused with the segments of $\mathcal{S}$) we perform a query in $\mathcal{D}(\mathcal{S}_2)$. The maximum obtained is *max_sum*$(\mathcal{S},\tau)$. This operation is doable in time $O(\sqrt{m'}\log^2 m')$. □

We next turn our attention to conclude the proof of Theorem 2.1.

Proof. (of Theorem 2.1) The algorithm consists of sweeping the translation plane from bottom to top, using a horizontal sweep line $\ell(y)$. Here y equals the time parameter τ used by the data structures. We maintain a set $\mathcal{S}$ of gsegments, initially empty, and we maintain its maximum *max_sum*$(\mathcal{S},y)$ using the data structure $\mathcal{D}(\mathcal{S}(y))$ of Lemma 2.3. As shown later, the maximum value obtained is equal to $\max_t Cov_\varepsilon(A,t+B)$.

Let D_ε be a square of edge-length 2ε, and consider the Minkowski sum

$$D_\varepsilon\oplus A^h=\{d+a\,|\,d\in D_\varepsilon\text{ and }a\text{ is a point of a segment of }A^h\}.$$

Note that $D_\varepsilon\oplus A^h$ can be expressed as the union of n rectangles, all of height 2ε, so the boundaries of each two intersect in at most two points, and by [17] the complexity of their union is $O(n)$. We impose a

horizontal decomposition on $D_\varepsilon \oplus A^h$ to obtain a set of $O(n)$ rectangles $R^h = \{\gamma_1, \gamma_2 \dots\}$ and a vertical decomposition on $D_\varepsilon \oplus A^v$ to obtain a set of $O(n)$ rectangles $R^v = \{\rho_1, \rho_2 \dots\}$. There are two types of events that we handle in the line sweep process, called *horizontal segment events* and *vertical segment event*. Upon encountering encountering an event, say for $\ell(y_0)$, we modify the data structures (described below) and compute $\max_{t \in \ell(y_0)} Cov_\varepsilon(A, t + B)$. The events are computed in the preprocessing stage, and stored in the line-sweep queue. The events are described as follows:

Horizontal segment events. For every $b_i \in B^h$ and rectangle $\gamma_j \in R^h$ we create a set of events, as follows: Assume

$$\gamma_j = ((c, d), (c + \Delta_x, d), (c, d + \Delta_y), (c + \Delta_x, d + \Delta_y)) ,$$

and the segment $b_i = ((a, b), (a+\delta, b)) \in B$. Assume that $\delta \leq \Delta_x$. The case that $\delta > \Delta_x$ is treated analogously.

The first event at which the pair (b_i, γ_j) is involved happens when $y = d + \Delta_y - b$ (i.e. when $(x, y) + b_i$ is aligned with the upper edge of γ_j). Upon this event, we insert the following gsegments into $\mathcal{S}$. These gsegments are not changing with y.

1. $r_1 = \overline{(c - a - \delta, 0)\ (c - a, \delta)}$. The x-span of this gsegment corresponds to all translations on $\ell(y)$ for which $(x, y) + b_i$ intersects γ_j, but the left endpoint of b_i is outside γ_j.
2. $r_2 = \overline{(c - a, \delta)\ (c + \Delta_x - a - \delta, \delta)}$. The x-span of this gsegment corresponds to all translations on $\ell(y)$ for which $(x, y) + b_i$ is fully contained in γ_j.
3. $r_3 = \overline{(c + \Delta_x - a - \delta, \delta)\ (c + \Delta_x - a, \delta)}$. The x-span of this gsegment corresponds to all translations on $\ell(y)$ for which $(x, y) + b_i$ intersects γ_j, but the right endpoint of b_i is outside γ_j.

The second event at which the pair (b_i, γ_j) is involved happens when $y = d - b$ (i.e. when $(x, y) + b_i$ is aligned with the lower edge of γ_j.) Upon this event, we delete r_1, r_2, r_3 from $\mathcal{S}$. Note that for every $y \in [d + \Delta_y - b, d - b]$ the function $r_1(x) + r_2(x) + r_3(x)$ equals the length of the portion of $(x, y) + b_i$ inside γ_j.

Vertical segment events. For every $b_i \in B^v$ and rectangle $\rho_j \in R^v$ we create a set of events, as follows: Assume that

$$\rho_j = ((c, d), (c + \Delta_x, d), (c, d + \Delta_y), (c + \Delta_x, d + \Delta_y)) ,$$

and $b_i = ((a, b), (a, b+\delta))$. Again assume that $0 < \delta \leq \Delta_y$ (the other case is treated analogously). Note that the pair (b_i, ρ_j) is involved in several events.

- Once $y = d + \Delta_y - b$ (i.e., the lower endpoint of $(x, y) + b_j$ is aligned with the upper edge of ρ_j) we insert the gsegment $s_1(y) = \overline{(c - a, y' - y)\ (c + \Delta_x - a, y' - y)}$ into $\mathcal{S}(y)$, where y' is the current value of y.
- Once $y = d + \Delta_y - \delta - b$ (i.e. the upper point of $(x, y) + b_i$ is aligned with the upper edge of ρ_j) we delete $s_1(y)$ from $\mathcal{S}(y)$ and insert $s_2(y') = \overline{(c - a, \delta)\ (c + \Delta_x - a, \delta)}$. This is a static horizontal gsegment.

- Once $y = d - b - \delta$ (i.e. the lower point of $(x, y) + b_i$ is aligned with the lower edge of ρ_j) we delete $s_2(y)$ from $\mathcal{S}(y)$, and insert the gsegment

$$s_3(y) = \overline{(c - a, \delta + d - b + y)\,(c + \Delta_x - a, \delta + d - b + y)}$$

- Once $y = d - b - \delta$ (i.e. the upper point of $(x, y) + b_i$ is aligned with the lower edge of ρ_j) we delete $s_3(y)$ from $\mathcal{S}(y)$.

Note that $s_1(y), s_2(y)$ and $s_3(y)$ represent, each in its y-span, the function which is the length of the portion of b_i inside ρ_j.

Observe that for any given y, the function $\sum_{e \in \mathcal{S}} e(y, x)$ represents the total length of the portion of the segments of $(x, y) + B$ which are contained inside $D_\varepsilon \oplus A$, i.e. $Cov_\varepsilon(A, (x, y) + B)$. Since the maximum value of these functions must be obtained at one of the events listed above, and at each such event we check this maximum, the correctness of the algorithm follows.

Time analysis: Overall, we add and delete four (moving) gsegments for each pair (b_i, γ_j) (for $b_i \in B^h, \gamma_j \in R^h$) or a pair (b_i, ρ_i) (for $b_i \in B^v, \gamma_j \in R^v$), thus a the total number of events is $O(n^2)$. We compute $max\_sum(\mathcal{S}, y)$ for each of these events, in time $O(\sqrt{n}^2 \log^2 n)$, hence the overall running time of the algorithm is $O(n^3 \log^2 n)$. It is not hard to show that this bound also bounds the time needed to construct the data structures. This concludes the proof of Theorem 2.1. □

2.2 Maximum coverage for horizontal segments

We present a faster algorithm for the case that all segments in A and B are horizontal. This is a line-sweep algorithm influenced by the Chew-Kedem algorithm [10], and Chew *et al.* [11] algorithm for computing the Hausdorff distance under translation between point-sets in the plane, under the L_∞ norm. This time, we sweep the plane from left to right, using a vertical sweep line. Let $D_\varepsilon \oplus A$ and $R^h = \{\gamma_1, \gamma_2 \dots\}$ be as in the proof of Theorem 2.1. For every horizontal segment $b_i \in B$ and $\gamma_j \in R^h$ let β_{ij} denote the rectangle in the translation plane consisting of all translations in which b_i intersects γ_j (i.e. $\beta_{ij} = \{t \mid (t + b_i) \cap \gamma_j \neq \emptyset\}$). Let $\mathcal{E}$ denote the set of the vertical edges of all rectangles β_{ij}.

Let $\mathcal{T}$ be a segment tree constructed on the projections of the segments of $\mathcal{E}$ on the y-axis. During the algorithm we sweep the translation plane using a vertical sweep line ℓ. Once ℓ meets an edge $e \in \mathcal{E}$, we insert e into $\mathcal{T}$. No edge is deleted.

Let μ be a node of $\mathcal{T}$. Let I_μ be the y-span corresponding to μ, and let $S_\mu \subseteq \mathcal{E}$ denote the edges of $\mathcal{E}$ corresponding to μ, i.e. the edges of $\mathcal{E}$ whose y-span contains I_μ but not $I_{parent(\mu)}$. Let $\mathcal{T}_\mu$ denote the subtree rooted at μ. For $x \in \mathbb{R}$, let $S_\mu(x) \subseteq \mathcal{S}_\mu$ denote the segment of S_μ whose x-coordinate is $\leq x$, and let

$$\mathcal{L}_\mu(x) = \{\, (b_i, \gamma_j) \mid \text{a vertical edge of } \beta_{ij} \text{ is stored in } S_{\mu'}(x), \text{ for some } \mu' \in \mathcal{T}_\mu \,\}\ .$$

We define the *maximal coverage associated with a node* μ at x_0, denoted by $Cov_\mu(x_0)$ as the maximal total length of segments

$$\{\gamma_j \cap ((x_0, y) + b_i) \mid (b_i, \gamma_j) \in \mathcal{L}_\mu(x_0)\}$$

where the maximum is taken over all translations $(x_0, y), y \in I_\mu$. For example,

$$Cov_{root(\mathcal{T})}(x_0) \;=\; \max_{y \in \mathbb{R}} Cov_\varepsilon(A, (x_0, y) + B)\ .$$

Let $\pi^*_\mu(x)$ denote the path in $\mathcal{T}$ from μ to the leaf that contains the translation that maximizes the coverage associated with μ at x. Thus for example, $\pi^*_{root(\mathcal{T})}(x)$ is the path from $root(\mathcal{T})$ to the leaf μ' such that $y^* \in I_{\mu'}$, where $Cov_\varepsilon(A,(x,y^*)+B) = \max_y\{Cov_\varepsilon(A,(x,y)+B)\}$.

We maintain the following fields with each node μ of $\mathcal{T}$. All of these, excluding $MaxMul_\mu$ are set to zero at the beginning of the algorithm. $MaxMul_\mu$ is set to 1.

- Pos_μ: the number of edges of $\mathcal{E}$ currently in $S_\mu(x)$ resulting from the right (resp. left) endpoint of a segment $b \in B$ meeting a left (resp. right) vertical edge of some rectangle γ_j. We call such an event a *Positive event*.

- Neg_μ: the number of edges of $\mathcal{E}$ currently in $S_\mu(x)$ resulting from the left (resp. right) endpoint of a segment $b \in B$ meeting a left (resp. right) vertical edge of some rectangle γ_j. We call such an event a *Negative event*. Observe that $Pos_\mu - Neg_\mu$ is

$$\begin{aligned}&|\{(b_i,\gamma_j) \in \mathcal{L}_\mu(x) \mid \gamma_j \text{ contains the right endpoint of } b_i, \text{ but not the left endpoint}\}|\\ -&|\{(b_i,\gamma_j) \in \mathcal{L}_\mu(x) \mid \gamma_j \text{ contains the left endpoint of } b_i, \text{ but not the right endpoint}\}|\end{aligned}$$

- x_last_μ — the last x at which we inserted an edge into the subtree of μ.

- $Max\_Tot\_at\_event_\mu$. We will show in Lemma 2.5 that this parameter stores $Cov_\mu(x\_last_\mu)$. We describe below how this variable is updated.

- $MaxMul_\mu$ — a multiplicative factor specifying the rate of increase of the maximum coverage, as the horizontal distance increases. The maximum is taken over all translations (x,y) for which $y \in I_\mu$. That is, if x_1 and x_2 are two close points in $\mathbb{R}$, then the difference in the coverage $Cov_\mu(x_2) - Cov_\mu(x_1) = (x_2 - x_1) * MaxMul_\mu$. Thus

$$MaxMul_\mu = \Sigma_{\mu' \in \pi^*_\mu(x)}(Pos_{\mu'} - Neg_{\mu'})$$

Handling an event.

During the algorithm we encounter two types of events. An **edge event** happens when the line sweep hits a vertical edge of $\mathcal{E}$. A **dominance event** happens at time x and node μ if $MaxMul_{left(\mu)} \neq MaxMul_{right(\mu)}$ and

$$\begin{aligned}&MaxMul_{left(\mu)} * (x - x\_last_{left(\mu)}) + Max\_Tot\_at\_event_{left(\mu)} =\\ &MaxMul_{right(\mu)} * (x - x\_last_{right(\mu)}) + Max\_Tot\_at\_event_{right(\mu)}\end{aligned}$$

This event occurs at x' if the translation (x,y) that maximizes $\{Cov(A,(x,y)+B) | y \in I_\mu\}$ is in $I_{left(\mu)}$ for x slightly smaller than x', and occurs at $I_{right(\mu)}$ for x slightly larger, or vice versa.

The x-coordinate of this event is computed and inserted into the queue of the line sweep, once the values of the fields of $left(\mu)$ or $right(\mu)$, or the fields of any of their descendants are modified. We explain next how we handle each such event.

We will show in Lemma 2.5 that the following claim is an invariant of the algorithm: For every $x \in \mathbb{R}, \mu \in \mathcal{T}$, $Cov_\mu(x) = Max\_Tot\_at\_event_\mu + (x - x\_last_\mu) * MaxMul_\mu$. In order to maintain this invariant, we use the following procedure, which we call for every node μ once a new edge of $\mathcal{E}$ is inserted into $S_\mu(x)$.

Function `UpdateNode`(μ)

$\text{Max\_Tot\_at\_event}_\mu = \text{Max\_Tot\_at\_event}_\mu + (x - \text{x\_last}_\mu) * \text{Max\_Tot\_at\_event}_\mu$

$\text{x\_last}_\mu = x$

Let $\delta > 0$ be an infinitely small constant

If $\text{Max\_Tot\_at\_event}_{\text{left}(\mu)} + \text{MaxMul}_{\text{left}(\mu)} * (x + \delta - \text{x\_last}_{\text{left}(\mu)}) >$
$\text{Max\_Tot\_at\_event}_{\text{right}(\mu)} + \text{MaxMul}_{\text{right}(\mu)} * (x + \delta - \text{x\_last}_{\text{right}(\mu)})$

Then

$\text{MaxMul}_\mu = \text{Pos}_\mu - \text{Neg}_\mu + \text{MaxMul}_{\text{left}(\mu)}$

Else

$\text{MaxMul}_\mu = \text{Pos}_\mu - \text{Neg}_\mu + \text{MaxMul}_{\text{right}(\mu)}$

If μ is not the root of $\mathcal{T}$

Then `UpdateNode`(parent(μ)).

Handling edge events at node μ. Let x be the current x-value of the line sweep. Once ℓ hits a new edge $s \in \mathcal{E}$, we first find all nodes μ for which $s \in S_\mu$ as in a standard segment tree. Next, for each such node μ, we increase either Pos_μ or Neg_μ by one, according to the type of s, and perform `UpdateNode(`μ`)`.

Handling dominance events at node μ. Once a dominate event occurs at a node μ, we call `UpdateNode`(μ).

Lemma 2.5. *The invariant*

$$\text{Cov}_\mu(x) = \text{Max\_Tot\_at\_event}_\mu + (x - \text{x\_last}_\mu) * \text{MaxMul}_\mu \quad \textit{for every } x \in \mathbb{R}, \mu \in \mathcal{T}$$

holds at any stage of the algorithm.

Proof. The proof is by a double induction on the height of a node μ and the sequence of events in which μ was updated. Assume first that μ is a leaf. Assume that x_1 is an event at which the fields of μ are updated, and that no event happened between x_1 and $x_2 > x_1$. x_2 is not necessarily an event. Also assume that in x_1 the invariant holds.

Let y_0 be a point in I_μ (since μ is a leaf, the choice of y_0 is not relevant). Let $(b_i \gamma_j) \in \mathcal{L}_\mu(x_2)$. Then $|((x_2, y_0) + b_i) \cap \gamma_j| - |((x_1, y_0) + b_i) \cap \gamma_j|$ is equal (resp.) $x_2 - x_1$, $x_1 - x_2$ or 0, according to whether γ_j contains only the right endpoint of $(x_2, y_0) + b_i$, only the left endpoint of b_i, or neither or both endpoints. In the first case, (b_i, γ_j) contributes 1 to Pos_μ. In the second case, (b_i, γ_j) contributes 1 to Pos_μ and 2 to Neg_μ, and in the last case, (b_i, γ_j) contributes the same amount to Pos_μ and Neg_μ. Summing over all pairs $(b_i, \gamma_j) \in \mathcal{L}_\mu(x_2)$, we obtain that $\text{Cov}_\mu(x_2) - \text{Max\_Tot\_at\_event}(x_1)$ equals $(\text{Pos}_\mu - \text{Neg}_\mu)(x_2 - x_1)$, from which the claim follows.

Next assume that μ is an internal node. Consider the pairs $(b_i, \gamma_j) \in \mathcal{L}_\mu(x_2)$ for which the vertical edges of β_{ij} are stored in S_μ. The contribution of these pairs to $\text{Cov}_\mu(x_2)$ is counted as in the case of a leaf node μ. Moreover, one can show that MaxMul at x_2 equals $\sum_{\mu'}(\text{Pos}_{\mu'} - \text{Neg}_{\mu'})$ where the sum is taken over all nodes μ' on the path leaving from μ to the leaf containing the translations that maximizes $\text{Cov}_\mu(x_2)$. This is because of the dominance events mechanism that guarantees that MaxMul_μ would take into account the contribution from the child μ' of μ from which $\text{Cov}_{\mu'}$ is larger. □

Theorem 2.2. *Let A and B be two sets of n horizontal segments in total. Then we can find in time $O(n^2 \log^2 n)$ a translation t at which $\text{Cov}_\varepsilon(A, t + B)$ is maximum.*

Proof. The number of edge events is clearly $O(n^2)$. Each edge event occurring at a node $\mu \in \mathcal{T}$ can cause $O(\log n)$ dominance events, one at each of the ancestor nodes of μ, thus there are $O(n^2 \log n)$ dominance events. Each event is handled in $O(\log n)$ time. Thus the bound of the running time follows.

To find the optimal translation, we monitor the maximal value of $Max\_Tot\_at\_event_{root(\mathcal{T})}$. Clearly the maximal coverage must be obtained at an edge event, and Lemma 2.5 guarantees that the maximum coverage must equal $Max\_Tot\_at\_event_{root(\mathcal{T})}$ at this event. □

3 A lower bound

Rucklidge [20] showed that for given a parameter $\varepsilon > 0$, and two families A and B of segments in the plane, the combinatorial complexity of the regions in the translation plane (TP) of all translations t for which $h(A, t+B) \leq \varepsilon$ is in the worst case $\Omega(n^4)$, where $h(A, B)$ is the directed Hausdorff distance from A to B. This bound is tight, since the number of intersection points created by n^2 rectangles in the plane is $O(n^4)$. We next show that the $\Omega(n^4)$–bound holds also in the case that the segments are horizontal. That is, we show a construction of sets A and B of n horizontal segments each, such that the combinatorial complexity of the regions R of all translations t for which $h(A, t+B) \leq \varepsilon$ is $\Omega(n^4)$. This is relevant for the previous section since R is exactly the region of all translations t for which $Cov_\varepsilon(A, t+B)$ is equal to the total length of the segments of A.

Assume for the construction that $\varepsilon = 1/2$. The first component in the construction (see Figure 2) is the set A'_1 consisting of $2n$ points, which are

$$\{(i, 1/2 + i/n^2) \text{ and } (i, -1/2 + i/n^2 - \delta), \text{ for } i =, 1 \ldots n\}$$

where δ is a small enough parameter. Let q^+ denote the Minkowski sum of a point q and a unit square. Thus the pair $(i, 1/2 - i/n)^+$ and $(i, -1/2 - i/n - \delta)^+$ forms two very close vertically aligned squares, where the gap between them is of unit width and height δ, and located at vertical distance i/n below the y-axis. We add the segment A''_1, which is the long horizontal segment between the points $(-n, 0)$ and $(0, 0)$ and the segment A'''_1 between $(n, 0)$ and $(2n, 0)$. Let $A_1 = A'_1 \cup A''_1 \cup A'''_1$.

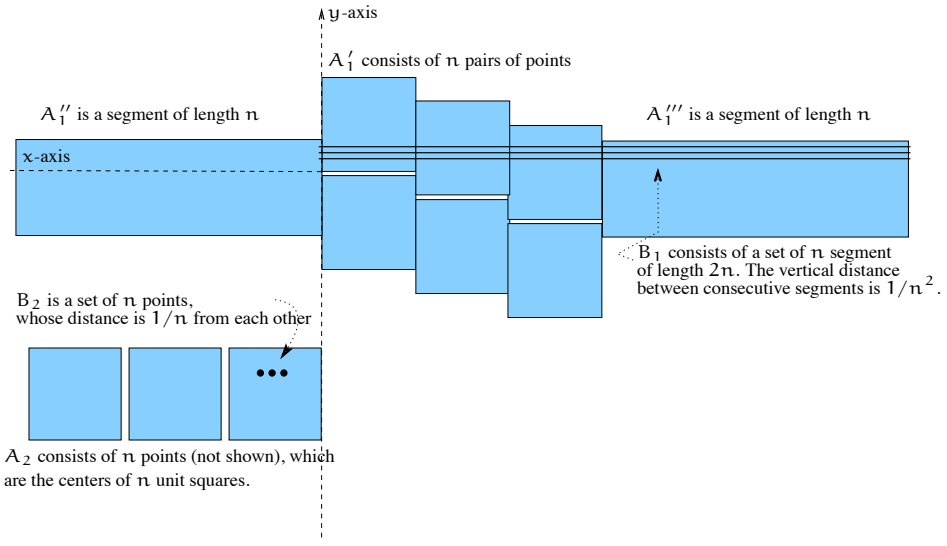

Figure 2: The lower bound construction. The set A is not shown explicitly — only A^+ is shown.

The set B_1 consists of n horizontal segments of length $2n$, whose vertical distance is $\frac{1}{n}^2$. The left endpoint of all of them is on the y-axis, and the middle one is on the x-axis. By shifting them vertically, each segment in turn is not completely covered at some time, when it passes between the gaps between one of the pairs of A_1. In all other cases, all the segments are completely covered. The region in TP corresponding to all translations t for which $h(A_1, t + B_1) \leq 1$ consists of $\Omega(n^2)$ horizontal strips, each of length n.

The set A_2 consists of the n points $(-(1 + 1/n^2)i, -5)$ (for $i = 1 \ldots n$). Thus A_2^+ (The Minkowski sum of A_2 and a unit square) creates n unit squares along the line $y = -5$, with a gap of $1/n^2$ between them. The set B_1 consist of n points along the horizontal line $(-1/2n, -5)$ (for $i = 1 \ldots n$). Observe that B_1 fits completely into each of the squares of A_2^+. However, by sliding B_1 horizontally, along $y = -5$ or anywhere at distance $\leq 1/2$ from the line $y = -5$, each of the points of B_1 "falls" at some stage into each of the gaps between each of the squares of A_2^+, The region $S_2 = \{t \mid h(A_2, t + B_2) \leq 1\}$ consists of $\Omega(n^2)$ vertical strips, each of height 2. Letting $A = A_1 \cup A_2$ and $B = B_1 \cup B_2$, the region $S = \{t \mid h(A, t+B) \leq 1\}$ is merely the intersection of S_1 and S_2, which is clearly of complexity $\Omega(n^4)$, thus proving our claim.

4 Matching Horizontal Segments Under Vertical Translation

In this section we describe a sub-quadratic algorithm for the Hausdorff matching between sets A and B of horizontal segments, when translations are restricted to the vertical direction.

Let $\rho^* = \min_t h(A, t + B)$ where t varies over all vertical translations, and $h(\cdot, \cdot)$ is the directed Hausdorff distance. Let M denote the ratio of the diameter to the closest pair of segments in $A \cup B$. Further, let $[M]$ denote the set of integers $\{1 \ldots M\}$.

Theorem 4.1. *Let A and B be two sets of horizontal segments, and let $\varepsilon < 1$ be a given parameter. Then we can find a vertical translation t for which $h(A, t+B) \leq (1+\varepsilon)\rho^*$ in time $O(n^{3/2} \max(poly(\log M, \log n, 1/\varepsilon)))$.*

We first relate our problem to a problem in string matching:

Definition 4.1. (Interval matching): *Given two sequences $t = (t[1] \ldots t[n])$ and $p = (p[1] \ldots p[m])$, such that $p[i] \in [M]$ and $t[i]$ is a union of disjoint intervals $\{a_i^1 \ldots b_i^1\} \cup \{a_i^2 \ldots b_i^2\} \ldots$ with endpoints in $[M]$, find all translations j such that $p[j] \in t[i + j]$ for all i. The* size *of the input to this problem is defined as $s = \sum_i |t[i]| + m$.*

We also define the *sparse* interval matching problem, in which both $p[i]$ and $t[i]$ are allowed to be equal to a special empty set symbol $\emptyset$, which matches any other symbol or set. The size s in this case is defined as $\sum_i |t[i]|$ plus the number of non-empty pattern symbols. Using standard discretization techniques [8, 16], we can show that the problem of $(1 + \varepsilon)$-approximating the minimum Hausdorff distance between two sets of n horizontal intervals with coordinates from $[M]$ under vertical motion can be reduced to solving an instance of sparse interval matching with size $s = O(n)$.

Having thus reduced the problem of matching segments to an instance of sparse interval matching, we show that:

- The (non-sparse) interval matching problem can be solved in time $O(s^{3/2}\text{polylog } s)$.
- The same holds even if the pattern is allowed to consists of unions of intervals.
- The sparse interval matching problem of size s can be reduced to $O(\log M)$ non-sparse interval matching problems, each of size $s' = O(s \text{ polylog } s)$.

These three observations yield the proof of Theorem 4.1. In the remainder of this section, we describe the proofs of the above observations.

The interval matching problem. Our method follows the approach of [1, 19] and [3].

Firstly, we observe that the universe size M can be reduced to $O(s)$, by sorting the coordinates of the points/interval endpoints and replacing them by their rank, which clearly does not change the solution. Then we reduce the universe further to $M' = O(\sqrt{s})$ by merging some coordinates, i.e. replacing several coordinates $x_1 \dots x_k$ by one symbol $\{x_1 \dots x_k\}$, in the following way. Each coordinate (say x) which occurs more than $\sqrt{s}$ times in t or p is replaced by a singleton set $\{x\}$ (clearly, there are at most $O(\sqrt{s})$ such coordinates). By removing those coordinates, the interval $[M]$ is split into at most $O(\sqrt{s})$ intervals. We partition each interval into smaller intervals, such that the sum of all occurrences of all coordinates in each interval is $O(\sqrt{s})$. Clearly, the total number of intervals obtained in this way is $\sqrt{s}$. Finally, we replace all coordinates in an interval by one (new) symbol from $[M']$ where $M' = O(\sqrt{s})$. By replacing each coordinate x in p and t by the number of a set to which x belongs, we obtain a "coarse representation' of the input, which we denote by p' and t'.

In the next phase, we solve the interval matching problem for p' and t' in time $O(nM'\text{polylog}(n, M'))$ using a Fast Fourier Transform-based algorithm (see the above references for details). Thus we exclude all translations j for which there is an i such that $p[i]$ is not included in the *approximation* of $t[i+j]$. However, it could still be true that $p[i] \notin t[i+j]$ while $p'[i] \in t'[i+j]$. Fortunately, the total number of such pairs (i, j) is bounded by the number of new symbols (i.e. M') times the number of pairs of all occurrences of any two (old) symbols corresponding to a given new symbol (i.e. $O(\sqrt{s}^2)$). This gives a total of $O(s^{3/2})$ pairs to check. Each check can be done in $O(\log n)$ time, since we can build a data structure over each set of intervals $t[i]$ which enables fast membership queries. Therefore, the total time needed for this phase of the algorithm is $O(s^{3/2}\text{polylog})$, which is also a bound for the total running time.

The generalization to the case where $p[i]$ is a union of intervals follows in essentially the same way, so we skip the description here.

The sparse-to-non-sparse reduction. The idea here is to map the input sequences to sequences of length P, where P is a random prime number from the range $\{c_1 s \log M \dots c_2 s \log M\}$ for some constants c_1, c_2. The new sequences p' and t' are defined as $p'[i] = \cup_{i':i' \bmod P = i} p[i']$ and $t'[i] = \cup_{i':i' \bmod P = i} t[i']$. It can be shown (using similar ideas as in [8]) that if a translation j *does not* result in a match between p and t, it will remain a mismatch between p' and t' with constant probability. Therefore, all possible mismatches will be detected with high probability by performing $O(\log M)$ mappings modulo a random prime.

Acknowledgements

We would like to thank Helmut Alt, Julien Basch, Mikkel Thorup, Carola Wenk and Li Zhang for fruitful discussion. We also thank Héctor H. González-Baños and Eric Mao for supplying some of the pictures in this paper.

References

[1] K. Abrahamson, Generalized string matching, *SIAM Journal on Computing* 16 (1987), 1039–1051.

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

[3] A. Amir, M. Farach, Efficient 2-dimensional approximate matching of half-rectangular figures, *Information and Computation* 118 (1995), 1–11.

[4] G. Barequet and S. Har-Peled, Some Variants of Polygon Containment and Minimum Hausdorff Distance under Translation are 3sum-Hard, *Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, 862–863.

[5] J. Basch, L. J. Guibas and J. Hershberger, Data Structures for Mobile Data, *J. Algorithms* 31 (1999), 1–28.

[6] M. de Berg and O. Schwarzkopf. Cuttings and applications. *Internat. J. Comput. Geom. Appl.* 5 (1995), 343–355.

[7] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[8] D. Cardoze, L. Schulman, Pattern Matching for Spatial Point Sets, *Proceedings 39th Annual IEEE Symposium on Foundations of Computer Science*, 1998, 156–165.

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

[10] L. P. Chew and K. Kedem, Improvements on geometric pattern matching problems, *Proceedings 3rd Scand. Workshop on Algorithms Theory*, LNCS Vol. #621, 1992, 318–325.

[11] L. P. Chew, D. Dor, A. Efrat, and K. Kedem, Geometric Pattern Matching in d-Dimensional Space, *Discrete and Computational Geometry* 21 (1999) 257–274.

[12] A. Efrat, P. Indyk and S. Venkatasubramanian. Pattern Matching for Sets of Segments, *Proceedings 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001, 295–304.

[13] S. Har-Peled, V Koltun, D. Song, and K. Goldberg, Efficient Algorithms for Shared Camera Control, *Proceedings 19th Annual Symposium on Computational Geometry*, 2003, 68–77.

[14] P. J. Heffernan and S. Schirra, approximate decision algorithms for point set congruence, *Computational Geometry: Theory and Applications* 4 (1994), 137–156.

[15] P. Indyk, R. Motwani, S. Venkatasubramanian, Geometric Matching Under Noise: Combinatorial Bounds and Algorithms, *Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999, 457–465.

[16] K. Kedem, R. Livne, J. Pach, M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete and Computational Geometry* 1 (1986), 59–71.

[17] S. Khanna, R. Motwani, and R. Wilson, On certificates and lookahead in dynamic graph problems, *Algorithmica* 21 (1998), 377–394.

[18] S. R. Kosaraju, Efficient string matching. manuscript, 1987.

[19] W. Rucklidge, Lower bounds for the complexity of the graph of the Hausdorff distance as a function of transformation, *Discrete and Computational Geometry* 16 (1996),135–153.

[20] S. Venkatasubramanian. *Geometric Shape Matching and Drug Design*. PhD thesis, Department of Computer Science, Stanford University, August 1999.