Translating a Convex Polygon to Contain a Maximum Number of Points*

EXTENDED ABSTRACT

Gill Barequet† Matthew Dickerson‡ Petru Pau§

1 Introduction

Finding an optimal transformation of a region such that it contains (encloses) a given point set or subset is a problem that has received considerable attention. Applications include optimal object placement (CAD), clustering, and statistical data analysis. As may be expected, there are many variants of this generally stated problem. For example, finding the smallest circle enclosing a given point set S is a famous problem in computational geometry. (See [13, pp. 255–259] for a brief summary.) This problem has been naturally extended to the smallest enclosing triangle [7, 11, 3], square, and rectangle [15], and the smallest enclosing convex polygon.

Another variant of this problem is: Given a set S of planar points and a fixed integer k, find a region that contains a *k-subset* of S and minimizes some measure such as area, radius, or circumference. Efrat, Sharir, and Ziv [6] give algorithms for computing the smallest k-enclosing circle in $O(nk\log^2 n)$ time and $O(nk)$ space, or $O(nklog^2 n\log(n/k))$ time and $O(n\log n)$ space. Eppstein and Erickson [5] provide fast new solutions to a number of these problems including finding k-subsets of a given set S that minimize the following measures: area, perimeter, diameter, and circumradius. Their algorithm for minimizing circumradius (equivalent to finding the smallest k-enclosing circle) requires $O(n\log n + kn\log k)$ time and $O(n\log n + kn + k^2\log k)$ space. Recent work by Datta, Lenhof, Schwartz, and Smid [4] has also provided further improvements and refinements.

A closely related problem is to find a placement of a region that maximizes the size k of the subset contained. That is, instead of fixing k and finding the optimal region enclosing it, we fix the size and shape of the region and try to maximize k. In this paper, we examine the following problem:

Problem 1 *Given a convex polygon P and a planar point set S, find a translation τ that maximizes the number of points contained by $\tau(P)$.*

*Work on this paper by the first author has been supported by the G.I.F., the German-Israeli Foundation for Scientific Research and Development, and by the Israeli Ministry of Science Eshkol grant 0562-1-94. Work by the second and third authors has been supported in part by NATO East Europe grant GER-93-55507. Work by the second author has been supported also by National Science Foundation grant CCR-93-01714.

†School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel

‡Middlebury College, Middlebury Vermont, USA

§University of Timişoara, Timişoara, România

This problem has many of the same applications as the problems mentioned in the previous paragraphs, and has been used as a substep in some of their solutions [5, 6]. Problem 1 also relates to the fixed radius search problems and to problems of optimal object placement.

Chazelle and Lee [2] first solved the problem of placing a fixed radius circle to contain the largest subset of a given set S. Their algorithm requires $O(n^2)$ time. Eppstein and Erickson [5], as a substep of their algorithm to find the minimum L_∞ diameter k-subset of a given set S, note that an algorithm of Overmars and Yap [12] can be modified to find the maximum depth of an arrangement of axis-aligned rectangles. This approach solves in $O(n\log n)$ time the problem of finding an optimal translation of a rectangle to cover the maximum sized subset of S. That is, it solves Problem 1 in $O(n\log n)$ time in the special case when 'polygon' is a rectangle. Efrat, Sharir, and Ziv [6] as a substep in their algorithm for finding the smallest k-enclosing homothetic copy of an m-vertex polygon, claim an *oracle* solving Problem 1. They suggest a line- sweep technique for their oracle, but give no details. For the case when m is a constant, they claim the algorithm to run in $O(nk\log n)$ time (it should be $O(nk\log n + m)$ time) and for general m the complexity is worse by a factor of $O(\log m)$; that is, the algorithm requires $O(nk\log n\log m + m)$ time.

A variant of Problem 1 is:

Problem 2 (Bichromatic Coverage) *Given a convex polygon P and two planar point sets A and B, find a translation τ such that the number of points in A contained by $\tau(P)$ minus the number of points in B contained by $\tau(P)$ is maximized.*

1.1 Overview of New Results

We provide two general solutions to Problem 1 for arbitrary convex polygons. Our first algorithm requires $O(nk\log(mk) + m)$ time and $O(m + n)$ space, which is asymptotically faster than that of [6]. We also give details of a line-sweep algorithm, similar to that suggested by [6], and show that the algorithm runs in $O(nk(\log n + \log m) + m) = O(nk\log(nm) + m)$ time rather than $O(nk\log n\log m)$ time. We also show that the *bichromatic* version, Problem 2, can be solved in the same running time with only a slight modification of the algorithm. In fact, our algorithms solve a more general problem where all points have given weights, and the goal is to maximize the total weight of contained points.

Our first algorithm is based on a lemma that limits the number of possible translations to certain *transla-*

tion stable placements. A naive algorithm based solely on this lemma requires $O(n^2 m \log(mn))$ time. We show how to improve the complexity to output-sensitive $O(nk \log(mk) + m)$ time at no further cost to space. Our improvements are based on two techniques. The first technique relies on a simple property that relates translation stable placements to pairwise intersections of convex polygons, which may then be computed efficiently. The second technique is bucketing. Let A_P be the area of the smallest rectangle enclosing P, and let A_S be the area of the smallest rectangle enclosing S. In the case where the ratio A_S/A_P is $O(n)$, then a simple bucketing approach using buckets of size A_P achieves the $O(nk \log(mk))$ time bound at no further space cost. If A_S/A_P is $\omega(n)$, then we use either a hash table to explicitly store only those buckets actually containing points from S, or a degraded grid approach as suggested in various papers by Lenhof and Smid [10]. Both of the algorithms make use of a method for computing in $O(\log m)$ time the intersections between two translated copies of an m-vertex convex polygon. This method is based on prune-and-search techniques presented by Kirkpatrick and Snoeyink [9].

2 Geometric Preliminaries

We now present some geometric results necessary for our latter algorithms. We begin with some definitions and notation that will be used throughout the paper.

We use q_i to represent the ith point in our input set S. We assume that the polygon P is represented as a list of its vertices $p_1, \ldots, p_m$ given in clockwise order with p_1 located at the origin. Thus given a translation τ represented as a vector v, we can in constant time compute the position of the ith vertex of $\tau(P)$ as $p_i + v$ without explicitly computing the entire polygon.

We use the standard notation ∂P to represent the boundary of the polygon P. That is, ∂P is the union of the edges and vertices of P. Likewise, $\partial\tau(P)$ is the boundary of the translated polygon $\tau(P)$.

In addition to this notation, we also use the following definitions. Given two polygons P and Q such that P contains Q. Chazelle [1] defines a *contact point* between P and Q as an intersection of a vertex of Q with an edge of P. When translations and rotations of P are allowed, a *stable placement* of polygons P and Q is one with three contact points. (Note that if a vertex of Q lies on a vertex of P, it intersects *two* edges of P and thus contributes two contact points.) The same definition may be extended to a polygon P and a contained set S. For this paper, however, we need a slightly different notion of stable placement which applies to translations only. We define a *translation stable placement* as follows:

Definition 1 *Let $\tau(P)$ be a translation of polygon P containing a set of points S. We say that $\tau(P)$ is in* translation stable placement *if at least 2 points in S are on ∂P.*

Note that unlike Chazelle's original general definition, a translation stable placement is not defined by contact points but by two distinct points in S both on the boundary of $\tau(P)$. Using these definitions, we may now proceed with the preliminary results.

2.1 Limiting the Search Space

Chazelle [1] showed that if a polygon P contains a polygon Q, then there exists a rigid motion (translation and rotation) of P containing Q and in a stable placement. That is, at least one of the following conditions holds: 1) At least 3 points in S lie on ∂P; or: 2) At least 2 points in S lie on ∂P and at least one point lies on a vertex of P. We may easily extend (or rather simplify) this result to show the following:

Lemma 1 *Let S be a planar point set and let P be a convex polygon. If there is a translation τ such that $\tau(P)$ contains $k \geq 2$ points, then there exists a translation τ^* such that $\tau^*(P)$ contains at least k points and is in translation stable placement.*

Actually, this result is stronger than we need. In our first algorithm, we limit our search space to translations τ with at least one point of S on $\partial\tau(P)$. However we use the idea of stable placement to find these translations. The following lemmas show that given two points q_i and q_j and a polygon P, translation stable placements can be determined efficiently. (For Lemmas 2 and 3, see Figure 1. For simplification, in the figure we let q_2 be on the origin $(0, 0)$. That is, τ_2 is the null translation.)

Lemma 2 *Let P be a convex polygon, q_1, q_2 points, and τ_1 and τ_2 the translations mapping the origin to points q_1 and q_2 respectively. For any point x on ∂P, define $\tau_x = q_2 - x$ as the translation that maps x to q_2. Then $\tau_1(x)$ is a point of intersection between $\partial\tau_1(P)$ and $\partial\tau_2(P)$ if and only if q_1 is on $\partial\tau_x(P)$.*

Lemma 2 states that $\tau_x(P)$ is in translation stable placement with q_1 and q_2 on the boundary of $\partial\tau_x(P)$ if and only if $\tau_1(x)$ is a point of intersection between $\partial\tau_1(P)$ and $\partial\tau_2(P)$. The proof of this lemma follows from elementary geometry and vector arithmetic. In fact, the lemma easily generalizes to the following:

Lemma 3 *Let P be a convex polygon, q_1, q_2 points, and τ_1 and τ_2 the translations mapping the origin to points q_1 and q_2 respectively. For any point x, define $\tau_x = q_2 - x$ as the translation that maps x to q_2. Then $x \in (\tau_1(P) \cap \tau 2(P))$ if and only if $\tau_x(\tau_1(P))$ contains both q_1 and q_2.*

2.2 Computing Events Quickly

Both of our algorithms are based on event queues of some sort. In the second algorithm, we use a standard line-sweep technique. In the first algorithm we use a different technique, that of an anchored sweep: sweeping the polygon around a point in S, keeping the polygon edges in contact with that point as we process events in clockwise order around the polygon.

In both cases, as was shown in Lemmas 2 and 3, the events are determined by intersection points between polygons. Fortunately, we do not need to compute intersections between two arbitrary polygons but only between two translated copies of the same convex polygon. That is, we must solve the following problem.

Problem 3 *Let P be a convex polygon with m vertices, and let τ_1 and τ_2 be translations. Compute intersections between $\partial\tau_1(P)$ and $\partial\tau_2(P)$.*

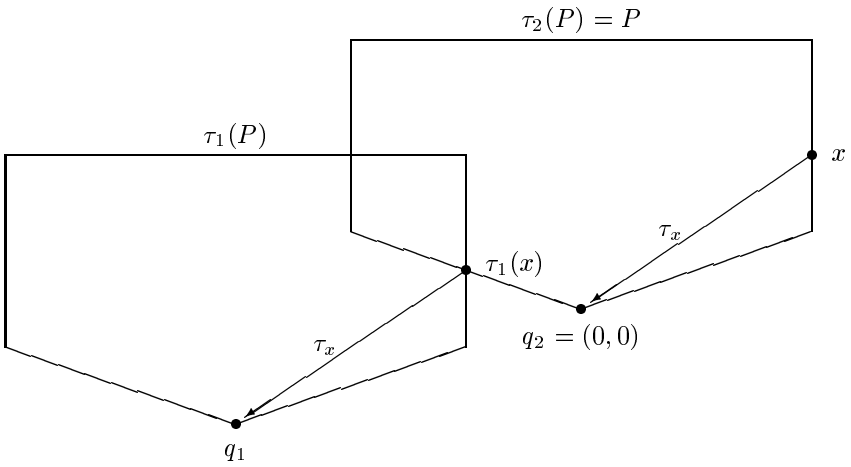

Figure 1: Stable Placements from Intersecting Polygons

We show that this problem can be solved efficiently, as is stated in the following lemma:

Lemma 4 *Problem 3 can be solved in $O(\log m)$ time for translates of an m-vertex convex polygon.*

The proof of this lemma follows from recent results of Kirkpatrick and Snoeyink [9] on tentative prune-and-search techniques for computing fixed-points. The intersections between $\partial\tau_1(P)$ and $\partial\tau_2(P)$ can be found from the two parallel chords of P of length and direction $\tau_2 - \tau_1$. It is shown in [9] that the fixed-point of the composition of a monotone increasing piecewise-basic function and a monotone decreasing piecewise-basic function can be computed in $O(\log m)$ time using tentative prune-and-search techniques. As an application, it is shown how to compute the a pair of chords of a polygon of a given length and direction, also in $O(\log m)$ time, solving Problem 3.

2.3 Limiting the Search Space More

We have described how our search space can be limited to translation stable placements. We now show that it may be limited further to a number of events which is *output sensitive*. The method for doing this varies with the algorithm. In our first algorithm, we use bucketing to limit the number of pairs of points that need to be explicitly examined. To prove this method is efficient, we will use the following two lemmas, which are not difficult to prove.

Lemma 5 *Let P be a convex polygon. There exist two rectangles R_P and R_I such that R_P encloses P and is no more than 2 times the area of P, and R_I is inscribed in P, is orthogonal to R_P, and is at least $1/2$ as long as R_P and at least $1/4$ as wide as R_P.*

Lemma 6 *Let S be a point set and P a convex polygon. Then there exists a rectangle R_P enclosing P with the following property: If there exists a translation $\tau(R_P)$ of R_P containing k points of S, then there exists a translation $\tau^*(P)$ of P containing $\Omega(k)$ points of S.*

The following result, proven by Sharir [14], will be used in the analysis of our second algorithm.

Lemma 7 (Sharir) *Let $\mathcal{A}$ be an arrangement formed by n shapes in the plane, having the property that the boundaries of each pair of shapes intersect at most twice. If the maximal depth of the arrangement is $\leq k$ then the number of intersections of pairs of the boundaries is $O(nk)$.*

3 The First Approach

We now present the first algorithm for the solution to Problem 1. From Lemma 1, we see that we may limit our search to translations of the polygon which are in translation critical positions. A naive approach based on this Lemma is, for every edge e_i in P and every point q_j in S, translate e_i onto q_j, and then slide e_i along q_j in discrete intervals determined by the translation stable placements. For each of the $\Theta(nm)$ edge-point pairs, we need to compute the distance of every other point in S to the boundaries of the current translated polygon in the direction determined by e_i, and keep an updated event queue. This requires $O(\log(nm))$ time per point for each of the $n-1$ other points for a total of $O(n^2 m \log(nm))$ time. Based on the results presented in Section 2, however, we can speed this approach considerably. First, we use bucketing as follows to limit the number of pairs of points examined. Let R_P be a rectangle enclosing P and of area proportional to P (as described in Lemma 5). Let R_S be the smallest area rectangle orthogonal to R_P and enclosing S. We use R_P to partition R_S into a grid of "buckets", and then place each point of S into its appropriate bucket. Note that for a given point $q_i \in S$, there are at most 9 buckets intersected by all polygons $\tau(P)$ with q_i on its boundary. We define the *neighborhood* B_i of point q_i as the bucket that q_i is in plus its 8 adjacent buckets, including those diagonally adjacent. Our search from point q_i will be limited to points in B_i.

Secondly, we avoid recomputing distance information for all points for every edge e_i of P, but instead for all $j \neq i$ and $q_j \in B_i$ we compute at one time all stable placements between q_i and q_j using Lemmas 4 and 6. The resulting algorithm is given in Figure 2. We use τ_j for the translation mapping the origin to the point $q_j \in S$. We let Q be a priority queue of pairs (x, j) where x is

I. Preprocessing: Preprocess points into buckets. Initialize Q.

II. Iteration

1. Set max := 0; {Maximum # of points contained so far}
2. **FOR** each point $q_i \in S$ **DO BEGIN** {Anchored sweep from every point}
3. Set $c := 1$; {Points contained by current translation}
4. **FOR** each $j \neq i$ and $q_j \in B_i$ **DO BEGIN** {Examine nearby points for containment}
5. Compute intersections of $\partial\tau_i(P)$ and $\partial\tau_j(P)$. {Compute stable placements with q_i, q_j}
6. Let $\tau_j(x)$ be a discrete intersection point; ADD (x, j) to Q.
7. **IF** q_j is contained by $\tau_i(P)$
   THEN Mark q_j "IN"; Set $c := c + 1$;
   ELSE Mark q_j "NOT IN".
   END IF
8. **END FOR**
9. **WHILE** $Q \neq \emptyset$ **DO BEGIN** {Sweep with stable placements as events}
10. Delete (x, j) from front of Q. {Update structures and counters}
11. **IF** q_j is not "IN"
    THEN Set $c := c + 1$; Mark q_j "IN".
    ELSE Set $c := c - 1$; Mark q_j "NOT IN".
    END IF
12. **IF** $c >$ max, **THEN** Set max := c; Store translation. **END IF**
13. **END WHILE**
14. **END FOR**

Figure 2: Algorithm 1

a point on ∂P and the points are ordered in clockwise order around P. We can represent x by the edge number and the distance along the edge. The proof of correctness follows from the results of Section 2.

3.1 Analysis

We now present an asymptotic analysis of the time required by Algorithm 1, given in Figure 2.

The outer loop beginning at Step 2 is iterated n times. It follows from Lemma 6 that for each iteration of the outer loop, the inner loop beginning at Step 4 is iterated $O(k)$ times. So steps 5 through 7 in the inner loop are iterated $O(nk)$ times. From Lemma 4, we see that the intersections at Step 5 can be computed in $O(\log m)$ time. Likewise, the polygon inclusion queries of Step 7 may be answered in $O(\log m)$ time.

Assume general position, such that we have at most two discrete intersections per polygon pair. The priority queue for each point thus has at most $2k$ events. It follows that the number of queue events in the loop beginning at Step 9 is also $O(k)$ and that each queue operation requires $O(\log k)$ time. The algorithm therefore requires a total of $O(nk(\log m + \log k)) = O(nk \log(mk))$ time if an appropriate bucketing strategy is used.

It is important to note that we are analyzing our use of bucketing in a deterministic way. That is, our running time is sensitive to k. Though the number of points in a particular bucket may grow as large as $\Theta(n)$, k is asymptotically as large as the number of points in the densest bucket. We also note that only minor modifications are required if we allow arbitrary position. When two polygons intersect along an edge, only the initial point on the first edge and the final point on the second edge need be added to the queue. Both of these are polygon vertices.

3.1.1 A Note on Bucketing

The drawback to this approach is that the number of buckets does not depend on n, m, or k but on A_S/A_P, the ratio of the area of the smallest rectangle enclosing S to the area of P. The initialization step requires $O(n + m + \frac{A_S}{A_P})$ time and space. There are two different approaches to resolve the problem. The first is instead of explicitly storing all $\Theta(A_S/A_P)$ buckets, we only store those that contain points. This may be accomplished using a hash table of $O(n)$ buckets. Every bucket reference is resolved with a lookup to this $O(n)$ sized hash table, which can be done in $O(1)$ expected case time.

A second approach is that of so-called *degraded* grids introduced by Lenhof and Smid [10]. The idea is to use varying sized buckets, whose size is the same as R_P if they contain a point, but which grow maximally long and maximally wide if empty. This ensures that the total number of buckets is still $O(n)$, accomplishing in the worst case the same time bounds accomplished by the hash table in the expected case, at the cost of a slightly more complicated preprocessing step. Readers are referred to the paper for details of this very elegant approach.

4 The Second Approach

We now provide an alternative and conceptually quite different algorithm for the solution to Problem 1. This algorithm is based on a technique now standard in computational geometry: the line sweep. This more common and slightly simpler technique comes at the cost of a $\log(nm)/\log(km)$ factor in the running time. It is based on the following simple observation: if you reflect a polygon P around a point p_i to form a new polygon P^R, then for any translation τ, $\tau(p_i)$ is contained by P^R if and only if p_i is contained by $\tau(P)$. Thus computing an optimal translation of P (the translation containing the maximum number of points) is equivalent to computing a location of maximum depth in the arrangement of polygons formed by translating a copy of the reflected P^R to every point $q_i \in S$. Efrat, Sharir, and Ziv [6] suggested this approach as an *oracle* in a parametric algorithm for computing the minimum area homothetic copy of a polygon containing k points for some fixed k. Without giving details, they claimed the algorithm has a running time of $O(nk \log n)$ if m is constant, with an additional $\log m$ factor otherwise. As noted earlier, there is also an additional $O(m)$ term for preprocessing of the polygon P. We give details of this approach now, and prove a tighter bound of $O(nk \log(nm) + m)$ rather than $O(nk \log n \log m + m)$.

Our algorithm makes use of only two data structures, both of which are simple, standard, and easy to implement. The first structure is the event queue itself, used for the line sweep. It returns the next event ordered by x-coordinate. This is implemented as a standard priority queue. The **Add**(e, Q) operation adds an event e to the queue Q, and the **DeleteMin**(Q) operation returns the next event e from Q. Both operations require $O(\log q)$ time, where q is the current size of the queue. The second structure keeps track of the current polygon chains. These are stored in a balanced binary tree, ordered by the y-coordinate at the current x position in the line sweep. We call this structure a *chain tree*. The **Add**(C, CT) operation adds a polygon chain C to a chain tree CT. The **NextEvent**(C, CT) operation returns the next intersection between a chain C and a neighboring chain in the tree CT. There is also a depth associated with the region between each pair of chains in the tree. The goal of the algorithm is to find the region of greatest depth.

4.1 Events

We describe the algorithm by first specifying the events in the queue. There are three main types of events described below:

Type 1. First Vertex The first (leftmost) vertex in a polygon is the first type of event. There are n of these events, all computed and added to the queue in the initial stage. For every event of this type, we compute two chains C_l and C_u, the lower and upper chains leading from the leftmost to the rightmost vertex of the polygons. Both chains are added to the chain tree (where they are initially consecutive) and each chain's first intersection event is computed using **NextEvent** and added to the event queue. The depth of the new region between C_l and C_u is one greater than the depth of the region into which the initial point of the consecutive chains was inserted. The original region is split into two regions.

Type 2. Last Vertex Likewise, the final (rightmost) vertex of a polygon also forms an event. This event ends two polygon chains C_l and C_u. Here a region ends, and the neighbor regions on both sides are merged to form a single new region. In doing so, a new pair of chains—call them C_{ll} and C_{uu}—becomes adjacent in the tree. We therefore must check **NextEvent**(C_{ll}, CT) and **NextEvent**(C_{uu}, CT) and add them to the event queue.

Type 3. Chain Intersection A third and final type of event is the intersection of two polygon chains C_1 and C_2. In this case, the two chains swap order in the tree, and each must be checked using **NextEvent**. What happens to the depth of the regions is a little complicated and may be divided into three subcases. If two upper or two lower chains intersect, then the depths of the regions remain the same. If an upper chain intersects with a lower chain, then the depth of the region increases or decreases by 2: in case the upper chain had lower y-value before the intersection and a higher y-value after the intersection, the depth of the region increases by 2; and in case the lower chain had lower y-value before the intersection and a higher y-value after the intersection, then the depth of the region decreases by 2.

The second algorithm is presented in Figure 3.

I. Preprocessing Let p_i be the rightmost vertex of P. Form P^R by reflecting P around p_i. Initialize a priority queue Q by adding every point $q_i \in S$ to the queue as an event of type 1.

II. Line Sweep
1. **WHILE** not Empty(Q) **DO**
2. Remove and process events as described in Section 4.1.

Figure 3: Algorithm 2

4.2 Analysis

By Lemma 7, the number of events of is $O(nk)$. The event queue can be managed at a cost of $O(\log(nk)) = O(\log n)$ time per event. There are at most $2n$ active chains at any time, so operations on the trees of chains require $O(\log n)$ time also. We saw in Lemma 4 that intersections of two translations of a convex polygon can be computed in $O(\log m)$ time. We modify this algorithm so that it only reports intersections on the specified subchains of the polygons. We thus have $O(nk)$ events requiring $O(\log n + \log m)$ time per event for a total running time

of $O(nk \log(nm))$ plus $O(m)$ time to preprocess the polygon P.

5 Weighted and Bichromatic Sets

The algorithms presented here can easily be extended to a more general version of the problem. Consider a set S where each point q_i is given a weight $W(q_i)$. Instead of maximizing the number of points covered, we want to maximize the total weights of all points covered. If $\forall i : 1 \leq i \leq n : W(q_i) \geq 0$, then Lemma 1 still applies and the algorithms run with no modifications.

However if $W(q_i) < 0$ for some points in the set, then it is possible that there is no translation that maximizes the total weight of the points covered and is in translation stable placement. However for each translation τ with a negatively weighted point q_i on the boundary, we may look for a "nearby" translation τ_ϵ that contains the same points but does not contain q_i. The same idea can be applied in the degenerate case where multiple points lie on the boundary, though if there are more than one negatively weighted points on the boundary then the ϵ translation will not necessarily exist. Thus with minor modifications, both Algorithms 1 and 2 can be used to solve Problem 2 in the same running times.

6 Summary

We have provided two asymptotically fast solutions to the problem of computing a translation of a given polygon P containing the maximum number of points of a given point set S. The faster of the two algorithms requires $O(nk \log(km) + m)$ time which is asymptotically faster than all previously known solutions. The algorithms are conceptually simple, using either a plane sweep or an anchored sweep. They are also self-contained except for the use of the tentative prune-and-search technique of Kirkpatrick and Snoeyink [9] for computing intersections of the polygons in $O(\log m)$ time, beating the more straightforward nested binary search which would require $O(\log^2 m)$ time. The algorithms also generalize at no cost in running time to the bichromatic variant of the problem, and also to the more general weighted point set problem.

6.1 Extensions and Open Problems

There are three obvious generalizations of Problem 1: we may extend to arbitrary simple polygons, to containment by general rigid motion, or containment by polyhedra in higher dimensions.

6.2 Acknowledgements

This paper benefited from discussions with Scot Drysdale, Dan Halperin, Matthew Katz, and others at the 10th ACM Symposium on Computational Geometry, Stony Brook NY, June 5–8 1994, and also from a discussion with Jack Snoeyink.

References

[1] B. Chazelle "The Polygon Placement Problem," in Advances in Computing Research: volume 1 ed. F. Preparata *JAI Press* (1983) 1–34.

[2] B. Chazelle and D.T. Lee "On a Circle Placement Problem," *Computing* **36** (1986) 1–16.

[3] S. Chandran and D. Mount "A parallel algorithm for enclosed and enclosing triangles," *IJCGA* **2:2** (1992) 191–214.

[4] A. Datta, H.P. Lenhof, C. Schwarz, and M. Smid "Static and dynamic algorithms for k-point clustering problems," *Proc. 3rd Workshop Algorithms and Data Structures* Lect. Notes in Comp. Sci. 709 Springer, NY (1993) 265–276.

[5] D. Eppstein and J. Erickson "Iterated nearest neighbors and finding minimal polytopes," *Disc. and Comp. Geom.* **11** (1994) 321–350.

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

[7] V. Klee and M.L. Laskowski "Finding the smallest triangles containing a given convex polygon," *J. Alg.* **6** (1985) 359– 375.

[8] K. Kedem, R. Livne, J. Pach, and M. Sharir "On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles," *Disc. and Comp. Geom.* **1** (1986) 59–71.

[9] D. Kirkpatrick and J. Snoeyink "Tentative prune-and-search for computing fixed-points with applications to geometric computing".

[10] H.P. Lenhof and M. Smid "Sequential and parallel algorithms for the k closest pairs problem," 1993.

[11] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin "An optimal algorithm for finding minimal enclosing triangles," *J. Alg.* **7** (1986) 258–269.

[12] M.H. Overmars and C.K. Yap "New upper bounds in Klee's measure problem," *SIAM J. Computing* **20** (1991) 1034–1045.

[13] F.P. Preparata and M.I. Shamos Computational Geometry: an Introduction, *Springer* (1985).

[14] M. Sharir "On k-sets in arrangements of curves and surfaces," *Disc. and Comp. Geom.* **6** (1991) 593–613.

[15] G.T. Toussaint "Solving geometric problems with the rotating calipers," *Proc. IEEE MELECON '83.*