Smallest k point enclosing rectangle of arbitrary orientation

Sandip Das[*] Partha P. Goswami[†] Subhas C. Nandy[‡]

Abstract Given a set of n points in 2D, the problem of identifying the smallest rectangle of arbitrary orientation, and containing exactly $k(\leq n)$ points is studied in this paper. The worst case time and space complexities of the proposed algorithm are $O(n^2\log n + nk(n-k)(n-k+\log k))$ and $O(n)$ respectively. The algorithm is then used to identify the smallest square of arbitrary orientation containing k points in $O(n^2\log n + nk(n-k)^2\log n)$ time.

1 Introduction

Given a set S of n points in 2D, and an integer k $(\leq n)$, we consider the problem of identifying the smallest rectangle on the plane which encloses exactly k points of S. A restricted variation of this problem has already been investigated, where the desired rectangle is axis-parallel. The first result on this problem appeared in [1] with time and space complexities $O(k^2 n\log n)$ and $O(kn)$ respectively. Both time and space complexity results are finally improved to $(k^2 n + n\log n)$ and $O(n)$ respectively in [2]. In [6], it is mentioned that all the aforesaid algorithms are efficient when k is small. It also proposes an efficient algorithm when k is large (very close to n). The time and space complexities of this algorithm are $O(n + k(n-k)^2)$ and $O(n)$ respectively. In d (> 2) dimensions, the algorithm proposed in [6] runs in $O(dn + dk(n-k)^{2(d-1)})$ time using $O(dn)$ space. In all these variations, the points are assumed to be in general position, i.e., no two points lie on the same horizontal or vertical line, and the desired rectangle is isothetic and closed (i.e., enclosed points may lie on the boundary of the rectangle). A similar problem is studied in [5], where n points are distributed on the plane, and the proposed algorithm identifies the smallest circle containing exactly k points in $O(n\log n + (n-k)^3 n^{\epsilon})$ time for some $\epsilon > 0$. The motivation of studying all these problems come from pattern recognition and facility location, where essential features are represented as a point set, and the objective is to identify a precise cluster (region) containing desired number of features.

We consider the generalized version of this problem, where the desired rectangle may be of arbitrary orientation. We assume that the lines joining pairs of points have distinct slopes. Our proposed algorithm runs in $O(n^2\log n + kn(n-k)(n-k+\log k))$ time and $O(n)$ space. The proposed technique can also identify the smallest k-enclosing square of arbitrary orientation in $O(n^2\log n + kn(n-k)^2\log n)$ time and $O(n^2)$ space. The time complexity result of identifying smallest k-enclosing square is comparable with that of identifying smallest k-enclosing circle proposed in [5].

[*] Indian Statistical Institute, Kolkata 700 108, India
[†] Calcutta University, Kolkata 700 029, India
[‡] Indian Statistical Institute, Kolkata 700 108, India

2 Formulation

Let $S = \{p_1, p_2, \ldots, p_n\}$ be the set of given points. The objective is to identify the smallest area rectangle of arbitrary orientation which contains exactly k points of S. A rectangle R is said to be a *k-rectangle* if it does not contain another rectangle R' containing k points. Let the optimum *k-rectangle* contain a designated subset $S'(\subset S)$ of k points. In other words, this rectangle encloses the convex hull of S'. Thus, each side of a *k-rectangle* contains a member of S, and at least one of its edges will surely contain two points of S.

Lemma 1 *Let $p_i, p_j, p_m \in S$ be three points. The number of* k-rectangles *with one edge passing through both p_i and p_j, and its parallel edge passing through p_m, may be anything from 0 to $k-2$.*

We consider each pair of points $p_i, p_j \in S$. Let L_{ij} denote the line passing through p_i and p_j. We explain the algorithm of identifying all the *k-rectangle*s with p_i, p_j at bottom boundary by sweeping a line L parallel to L_{ij} upwards. The *k-rectangle*s with top boundary containing p_i and p_j can be identified similarly.

Let p_i be to the left of p_j on the line L_{ij}. L_1 and L_2 are two lines perpendicular on L_{ij}, drawn at p_i and p_j respectively. This splits the half-plane above L_{ij} into three parts, say $LEFT$, MID and $RIGHT$. Following observation lists the possible situations that may take place when a new point p_m is encountered by the sweep line L. Let S_L, S_R and S_M denote the set of points encountered by L in $LEFT$, $RIGHT$ and MID respectively up to and including p_m. We assume that $p_i, p_j \in S_M$. $|A|$ denotes the cardinality of the set A.

Observation 1 **(a)** *If $|S_L| + |S_M| + |S_R| < k$ then no k-rectangle will be reported.*

(b) *If $|S_M| = k$ and $p_m \in MID$, then only one k-rectangle will be reported with p_m at its top boundary, its bottom boundary is the line segment $[p_i, p_j]$, its left and right boundaries are defined by lines L_1 and L_2 respectively. Here, one need not have to consider any point above p_m, and hence the sweep of L stops.*

(c) *If $|S_M| < k$ and $p_m \in MID$, then $min(|S_L|, k - |S_M|, |S_R|) + 1$ k-rectangles will be generated with p_m at top boundary and p_i, p_j at bottom boundary.*

(d) *If $|S_M| < k$ and $p_m \in LEFT$, then if the rectangle with one side aligned with L_{ij} and the line segment $[p_m, p_j]$*

as diagonal contain more than k points then no k-rectangle will be generated with p_m at its top boundary and (p_i, p_j) at its bottom boundary. Moreover, in the further sweep if another point p_ℓ appears whose projection on L_{ij} is to the left of that of p_m, then also no k-rectangle will be generated with p_ℓ at its top boundary and (p_i, p_j) at its bottom boundary.

(e) *If $|S_M| < k$ and $p_m \in LEFT$, then if the rectangle with one side aligned with L_{ij} and the line segment $[p_m, p_j]$ as diagonal contain $r(\leq k)$ points then $min(k - r, |S_L \cup S_M| - r, |S_R|) + 1$ k-rectangles will be generated with p_m at top and p_i, p_j at bottom boundary.*

(f) *Results similar to (d) and (e) holds if $|S_M| < k$, $p_m \in RIGHT$.*

3 Algorithm

Let $\mathcal{A}(H)$ denote the arrangement of the set of lines $H = \{\ell(p_i), i = 1, 2, \ldots, n\}$, where $\ell(p_i)$ is the dual (line) of the point $p_i \in S$ [3]. The vertices in $\mathcal{A}(H)$ are denoted by $\{v_{ij}, i \neq j, i = 1, 2, \ldots, n, j = 1, 2, \ldots, n\}$, where v_{ij} is the dual of the line L_{ij}. We consider each vertex $v_{ij} \in \mathcal{A}(H)$, and compute the *k-rectangle*s in the primal plane with p_i and p_j at its bottom boundary (i.e., all the k points inside the rectangle are above L_{ij}). Our algorithm does not store the entire arrangement in the memory. The vertices of $\mathcal{A}(H)$ are considered by sweeping a vertical line from left to right through the arrangement $\mathcal{A}(H)$. In order to analyze the combinatorial complexity of *k-rectangle*s on the floor, we need the following definition:

Definition 1 [3] A point q in the dual plane is at *level* θ $(0 \leq \theta \leq n)$ if there are exactly θ lines in H that lie strictly below q. The *θ-th level* of $\mathcal{A}(H)$ is the closure of a set of points on the lines of H which are exactly at level θ in $\mathcal{A}(H)$.

During the sweep of the vertical line through $\mathcal{A}(H)$, the sweep line status is maintained as an array B of size n. It contains the lines of H in the order in which they intersect the sweep line in its current position, from bottom to top. Each element $B[\alpha]$ (representing a line ℓ) is attached with an *id* of the corresponding point in S. B is initialized with the lines of H in increasing order of the ordinates of their intersections with the line $X = -\infty$. The sweep process is guided by an event-queue Q, maintained using min-heap. It stores the $v_{ij} \in \mathcal{A}(H)$ if ℓ_i and ℓ_j are consecutive entries in the array B and intersect at v_{ij} to the right of the sweep line. During the sweep, the next event point v_{ij} is obtained from Q in $O(\log n)$ time. The updating of Q needs another $O(\log n)$ time. The processing of v_{ij} includes swapping of the lines ℓ_i and ℓ_j in the array B, and reporting all *k-rectangles* with p_i and p_j at its bottom boundary.

Remark 1 *If $B[\alpha] = \ell_i$ and $B[\alpha + 1] = \ell_j$, then the points in the primal plane corresponding to all the line stored at $B[\beta], \beta = 1, 2, \ldots, \alpha - 1$ are above the line L_{ij}.*

Lemma 2 *While processing v_{ij} at level θ (say) of $\mathcal{A}(H)$, (i) if $\theta < k$, then no k-rectangle with p_i and p_j at its bottom boundary will be generated, and (ii) if $\theta \geq k$, then the number of reported* k-rectangles *with p_i and p_j at bottom boundary is at most $k(n - k)$.*

While processing the vertex $v_{ij} \in \mathcal{A}(H)$, the upward sweep of L in the primal plane for scanning the points above the line L_{ij} in increasing order of their distance from L_{ij} is equivalent to observing the elements $B[\beta]$, $\beta = \alpha - 1, \alpha - 2, \ldots, 1$.

Let χ_M be the number of points in S_M that are encountered by L up to the current instant of time. It is maintained using an integer variable. The points in S_L and S_R that are encountered during the upward sweep of L, are stored in two link-lists T_L and T_R. At an instant of time, T_L stores at most $k - \chi_M$ points in S_L closest to L_1 in increasing order of their distance from L_1; T_R also stores at most $k - \chi_M$ points in S_R closest to L_2 in increasing order of their distance from L_2. Initially, $T_L = \phi$, $T_R = \phi$ and $\chi_M = 2$, and the upward sweep of L starts. For the first $k - 1$ encountered points, no *k-rectangle* will be reported. For each of them, if it is in *MID* then χ_M is incremented. If it is in *LEFT* or *RIGHT* it is accumulated in T_L or T_R respectively in unordered manner. When the k-th point is encountered, the reporting of *k-rectangle*s starts. We now arrange the elements in T_L in increasing order of their distances from L_1. This needs copying the elements in a temporary array, then sorting the array and finally copying the sorted array in T_L. The elements in T_R are also arranged in similar manner.

Let p be the point faced by the sweep line, which corresponds to $B[\beta]$ (the sweep line status array). $|T_L|$, $|T_R|$ and χ_M indicates the number of points in the respective sets prior to the insertion of p in its appropriate set. The following three cases need to be considered.

$\mathbf{p} \in \mathbf{LEFT}$: If $|T_L| + \chi_M = k$, then the point farthest from L_1 is deleted from T_L. Note that, p is not inserted in T_L just now; it will be inserted in the next step. Next, we execute the following steps to report the *k-rectangles*.

- Perform a linear scan to identify the $(k - |T_L| - \chi_M)$-th element in T_R (from left).
- Next scan T_L from its the leftmost element and T_R from the $(k - |T_L| - \chi_M)$-th element onwards in ordered manner until (i) the proper position of p in T_L is reached or (ii) end of T_R is reached. At each move, a *k-rectangle* is reported. Its bottom and top sides pass through (p_i, p_j) and p respectively; left and right sides are bounded by the respective elements in T_L and T_R.
- In case (i), p is inserted in T_L at its proper position. In case (ii), the scanning in T_L proceeds further to insert p in T_L at its proper position.

$\mathbf{p} \in \mathbf{RIGHT}$: Similar to the earlier case.

$\mathbf{p} \in \mathbf{MID}$: We increment χ_M.

If $\chi_M = k$, a *k-rectangle* is reported with bottom boundary equal to the line segment (p_i, p_j) and p at top boundary, and the sweep of L stops (by Observation 1(b)).

Otherwise, if $|T_L| + \chi_M = k + 1$, then the point farthest from L_1 is deleted from T_L. Similarly, if $|T_R| + \chi_M = k + 1$, the point farthest from L_2 is deleted from T_R.

Next, *k-rectangles* are reported as stated for $p \in LEFT$. Here the question of inserting p in any set does not arise.

At the end of processing all vertices in $\mathcal{A}(H)$, the desired *k-rectangle* with smallest area is reported.

Theorem 3 *Given a 2D floor containing n points, the number of k-rectangles on the floor is* $O(nk(n-k)^2)$.

Theorem 4 *The time and space complexities of our algorithm are* $O(n^2 \log n + nk(n-k)(n-k+\log k))$ *and* $O(n)$ *respectively.*

4 Smallest k-enclosing square

A square containing k points is called as *k-square* if its size is smallest among all squares containing the same set of points. We classify the *k-squares* with respect to the number of points η present on its boundary. For $\eta = 2$, the only possibility of a *k-square* is that the two points will lie at two end-points of one of its diagonals. For $\eta = 3$, the following cases may arise:

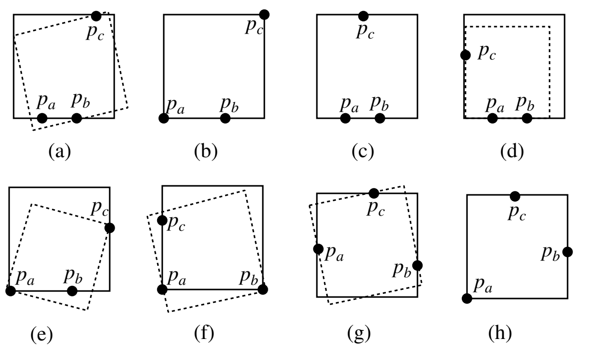

Figure 1: Different cases of *k-square* with $\eta = 3$

Case (i): Two points on one side of the *k-square* and the other point on its parallel side (Figure 1(a)-(c)). In Figure 1(a), a *k point enclosing square* is shown where the projection of p_c on L_{ab} lies outside $[p_a, p_b]$. It is not a *k-square* since its size can be reduced by rotating it around p_c as shown using dotted line. But if p_a and p_c forms the diagonal of the square (Figure 1(b)), its area can not be reduced, and hence it is *k-square*. Such instances are identified while finding the *k-squares* with $\eta = 2$. In Figure 1(c), the projection of p_c on L_{ab} lies inside $[p_a, p_b]$. This is surely a candidate *k-square*. If we translate it along the line joining (p_a, p_b), it remains a *k-square* of same size until it hits a point at one of its vertical boundaries, or any one of p_a, p_b becomes a corner point.

Case (ii): Two points on one side, the other point on one of its adjacent side (see Figure 1(d)-(f)). In each case, the size can be reduced keeping the same set of points inside, as shown using dotted line.

Case (iii): No two points on same side (see Figure 1(g)-(h)). The one shown in Figure 1(g) can be reduced as shown using dotted line. But the square in Figure 1(h) is a *k-square* since it can not be reduced further.

For $\eta = 4$, let the points appearing on the boundary of the square are p_a, p_b, p_c and p_d. Considering general position assumption, the possible subcases are

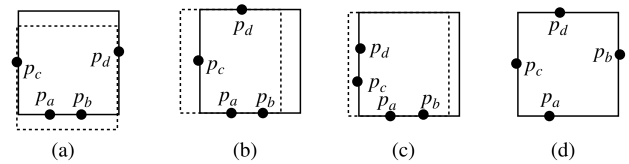

Figure 2: Different cases of *k-square* with $\eta = 4$

Case (i): Two points on one side and two other points on its two adjacent sides respectively. Translating the square as shown in Figure 2(a), this case can be reduced to a case where two points are on two mutually parallel sides respectively. This case does not yield a *k-square* since size of this square can surely be reduced with the same set of points inside the square.

Case (ii): Two points on one side, one point on its parallel side, and the fourth point is on one of the two remaining sides. This case can also be reduced to a case with $\eta = 3$ as shown using dotted line in Figure 2(b), and can be tackled as described earlier. If the point p_b is at the corner, then it is a candidate for *k-square* (see Figure 1(h)).

Case (iii): Two points on one side, and two other points on its parallel side. This violates the general position assumption, i.e., lines joining pairs of points do not have distinct slope. However, our algorithm can tackle this situation.

Case (iv): Two points on one side, and two other points on one of its adjacent sides. This case can be reduced to a case with two points on one side, and no other side containing any point (see Figure 2(c)).

Case (v): No two points lie on the same side of the square. This situation is shown in Figure 2(d). Following lemma says that for a given set of four points, at most 2 such *k-squares* are possible.

Lemma 5 *Given four points* $p_a, p_b, p_c, p_d \in S$, *the number of square whose each of the four sides contains one of these four points is at most 2.*

Proof. Let the lines which form the required square are ℓ_a, ℓ_b, ℓ_c and ℓ_d, where ℓ_i pass through the point p_i, $i = a, b, c, d$. The gradient of the lines ℓ_a and ℓ_c be μ and the gradient of the lines ℓ_b and ℓ_d be $-\frac{1}{\mu}$. Let A, B C and D denote respectively the point of intersection of (ℓ_a and ℓ_b), (ℓ_b and ℓ_c), (ℓ_c and ℓ_d) and (ℓ_d and ℓ_a) respectively. Equating the lengths $\overline{AB}$, $\overline{AD}$, we get a quadratic equation of μ. Hence the result follows. $\square$

For $\eta = 2$, we consider each pair of points $p_i, p_j \in S$, and test whether the square with $[p_i, p_j]$ as diagonal contains exactly k points or not using the method of simplex range searching. If *yes*, we compare its area with the existing optimum. The *k-square*s with $\eta = 3$ and $\eta = 4$ are identified using the method for identifying *k-rectangles* with minor modification as described below.

As in the earlier problem, we consider the vertices of the arrangement $\mathcal{A}(H)$. At each vertex (representing a line L_{ij}), the upward (resp. downward) sweep of a line L parallel to L_{ij} is also performed as earlier and using the same data structure. We execute the following two procedures: the first one identifies the smallest *k-square* with both p_i, p_j at its bottom boundary, and the second one identifies the smallest *k-square* with either p_i or p_j at its bottom boundary, and each of the other three sides contains a point of S.

Procedure A: Let the length of the line segment $[p_i, p_j] = r$. Initially, we consider the $r \times r$ square with one side aligned with the line segment $[p_i, p_j]$. Using simplex range searching, we count the number of points inside that square. If it exceeds k, then this procedure terminates without reporting any *k-square*. Otherwise, the following steps are executed.

Step A1 Our upward sweep proceeds, and the points in *LEFT* and *RIGHT* that encountered by the sweep line L are accumulated in T_L and T_R. For the points in *MID*, χ_M is incremented. Sweep continues untill it hits a point p_m whose distance from L_{ij} is greater than or equal to r.

Step A2 Now, the median find algorithm is invoked for the points in T_L to collect only k points (if available) which are closest to L_1. They are also orderly stored in T_L. Similarly, among the points in T_R only k points are stored in increasing order from L_2.

Step A3 Upward sweep continues and for each encountered point p_m, we test the existance of a *k-square* with p_i, p_j at bottom and p_m at top boundary as follows. Sweep terminates as soon as a *k-square* is found.

• Let the distance of p_m from L_{ij} be equal to h. We sequentially walk from left to right along T_L. For each element $\pi \in T_L$ (corresponding to a point $p_a \in S_L$) we choose a point ϕ on L_{ij} such that the length of the interval $[\pi, \phi] = h \geq [\pi, p_j]$ and the $h \times h$ square with (p_i, p_j) at its bottom boundary, p_m at top boundary, and p_a at its left boundary contains exactly k points.

Before describing Procedure **B**, we need the following important observation.

Observation 2 *Let R be the optimum k-square containing one designated point of S in each of its boundaries. If we rotate each edge by the same angle keeping the designated point on its boundary, it becomes a rectangle. We continue rotation till one of its boundaries hit another point inside/outside the square. Thus, we have a k-rectangle or a k+1-rectangle whose one side contains two points of S and each of the other three sides contains exactly one point of S. In other words, this k-rectangle or $k+1$-rectangle, on rotation, produces the desired (optimum) k-square.*

Procedure B:

Step B1: Consider each pair of points (p_i, p_j) and identify all *k-rectangle*s with (p_i, p_j) at one sides. For each such rectangle R, execute the following steps:

Step B2: Let the other three sides of R contain p_a, p_b, p_c respectively. We consider two quadruples $\{p_i, p_a, p_b, p_c\}$ and $\{p_j, p_a, p_b, p_c\}$. For each quadruple, we compute the smallest valid square with those four points at its four boundaries respectively using Lemma 5.

Step B3: Next, we use simplex range searching technique to test whether it is a *k-square* or not. If *yes*, it is a candidate for the optimum *k-square*.

Theorem 6 *The smallest k-square can be identified in $O(n^2 \log n + nk(n-k)^2 \log n)$ time using $O(n^2)$ space.*

Proof. Throughout the analysis, we use the fact that the simplex range counting query can be performed in $O(\log n)$ time using $O(n^2)$ time and space [4].

For $\eta = 2$, we consider all possible pair of points $(p_i, p_j) \in S$, and perform simplex range searching to test whether the square with (p_i, p_j) as diagonal is a *k-square*. This needs $O(n^2 \log n)$ time.

Using almost similar argument of proving Theorem 4, the worst case time required of executing Procedure A is $O(n^2 \log n + nk(n-k)(n-k+\log k))$.

In Procedure B, all the *k-rectangle*s and *k+1-rectangles* are considered. Each of them gives birth to two squares. For each square, the simplex range counting query is to be invoked. As the total number of possible *k-rectangles* is $O(nk(n-k)^2)$ (see Theorem 3), the total time required for execution of Procedure B for all the vertices of $\mathcal{A}(H)$ is is $O(nk(n-k)^2 \log n)$. □

References

[1] A. Aggarwal, H. Imai, N. Katoh and S. Suri, *Finding k point with minimum diameter and related problems*, Journal of Algorithms, vol 12, pp. 38-56, 1991.

[2] A. Datta, H. -P. Lenhof, C. Schwarz and M. Smid, *Static and dynamic algorithms for k-point clustering problems*, Journal of Algorithms, vol. 19, pp. 474-503, 1995.

[3] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer, Berlin, 1987.

[4] P. P. Goswami, S. Das and S. C. Nandy, *Simplex range searching and k nearest neighbors of a line segment in 2D*, Computational Geometry, in Press, 2004.

[5] J. Matousek, *On geometric optimization with few violated constraints*, Discrete Computational Geometry, vol. 14, pp. 365-384, 1995.

[6] M. Segal and K. Kedem, *Enclosing k points in the smallest axis parallel rectangle*, Information Processing Letters, vol. 65, pp. 95-99, 1998.