k-Enclosing Axis-Parallel Square

Chapter · June 2011

DOI: 10.1007/978-3-642-21931-3_7

4 authors, including:

Priya Ranjan Sinha Mahapatra
University of Kalyani
21 PUBLICATIONS **54** CITATIONS
SEE PROFILE

Partha P. Goswami
University of Calcutta
19 PUBLICATIONS **88** CITATIONS
SEE PROFILE

k-Enclosing Axis-parallel Square

Priya Ranjan Sinha Mahapatra

***Abstract*— Let P be a set of n points in the plane. Here an optimization technique is used to solve some optimization problems. A simple deterministic algorithm is proposed to compute smallest square containing at least k points of P. The time and space complexities of the algorithm are $O(n \log^2 n)$ and $O(n)$ respectively. For large values of k, the worst case time complexity of the algorithm is $O(n + (n-k)\log^2(n-k))$ using $O(n)$ space which is the best known bound for worst case time complexity. Then an algorithm is designed to locate smallest rectangle containing at least k points of P for large values of k and all values of k.**

***Index Terms*— Optimization Techniques, Computational geometry, Minimum enclosing square and Sweep line algorithm.**

I. Introduction

Given a set P of n points in the plane and an integer k ($\leq n$), we consider the following optimization problem [1]. Locate a minimum area axis-parallel square and rectangle that encloses at least k points of P. From now onwards, a square (rectangle) means an axis-parallel square (rectangle). A k point enclosing square (rectangle) S_k is said to be a k-$square$ (k-$rectangle$) if there does not exist another square (rectangle) having area less than that of S_k and containing k points from P [2].

For both k-square and k-rectangle problems, Aggarwal et al. [3] have presented an $O(k^2 n \log n)$ algorithm using $O(nk)$ space. Later on Datta et al. [4] and Eppstein et al. [5] independently improved the time complexity result to $O(k^2 n + n \log n)$ for k-rectangle problem using $O(kn)$ and $O(n)$ space respectively. Smid [6] also proposes a linear space algorithm to locate k-square in $O(n \log n + nk \log^2 k)$ time. This was followed by the work of Segal et al. [5] that further improves the time complexity result to $O(n + k(n-k)^2)$. They further extended this result to d-space ($d \geq 3$) proposing an $O(dn + dk(n-k)^{2(d-1)})$ algorithm using $O(dn)$ space. Datta et al. [7] proposed an $O(n \log n + n \log^2 k)$ time and $O(n)$ space algorithm to locate k-square. Chan [8] presented a randomized algorithm for computing a k-square. Das et al. [2] considered the generalized version of this problem where k points are enclosed by an arbitrarily oriented square or rectangle. They solved the arbitrary k-rectangle problem in $O(n^2 \log n + kn(n-k)(n-k+\log k))$ time and k-square problem in $O(n^2 \log n + kn(n-k)^2 \log n)$ time using $O(n)$ and $O(n^2)$ space respectively. A similar problem that locates minimum enclosing circle containing exactly k points has also been studied [9].

Deterministic (also known as polynomial-time) as well as probabilistic algorithms can be developed using differents optimization techniques. Deterministic algorithms are most often used if a clear relation between the characteristics of the candidate solutions and their utility for a given problem exists [10], [11], [12]. Then, the search space can efficiently be explored using different techniques such as divide and conquer scheme. If the relation between a potential solution and its fitness [13] are not so simple or too complicated, or the dimensionality of the search space is very high, it becomes difficult to design a deterministic algorithm to solve a problem. For this type of problem, it not possible to frame an exhaustive enumeration of the search space even for relatively small problems. Then, probabilistic algorithms come into action. The initial work in this domain which now has become one of most important research area in optimization was started long time ago. An especially relevant family of probabilistic algorithms are the Monte Carlo6 -based approaches, Evolutionary Computation (EC) [14], Memetic Algorithms, Random Optimization [15], Tabu Search (TS), Genetic Algorithms (GA) [16], [13] etc.

In this paper, we use the idea of *prune and search technique* (i.e. an optimization technique) to propose a deterministic algorithm for locating S_k for large values of k ($> \frac{n}{2}$). Each pruning step uses the solution of the corresponding *decision problem* that guides the search process. The decision version of k-$square$ problem asks whether there exists a square of side length α that encloses at least k points where k and α are the input parameters. In Section II, we present some preliminary observations and an algorithm for solving a decision version of the problem. In Section IV, we propose a deterministic algorithm to locate S_k that runs in $O(n+(n-k)\log^2(n-k))$ time using linear space for large values of k. In Section III and IV, we consider $k > n/2$. This algorithm is used to locate S_k for all values of k. In Section V, we also present algorithms that works for locating k point enclosing rectangle S_k that work for large values of k and for all values of k. The conclusions on this work and some related open problems are discussed in Section VI.

The motivation of studying this problem stems from pattern recognition [17], [18], [19] and facility location [20], [21], [22], [23], where essential features are represented as a point set, and the objective is to identify a precise cluster that contains at least k number of features [24], [25], [26], [27], [28], [29], [30]. Moreover, this problem also find application in VLSI physical design for accommodating specified locations like hot spots, power pins into k-$rectangle$ [31].

II. Preliminaries

Let $P = \{p_1, p_2, \ldots p_n\}$ be the set of n points in the plane. Our objective is to locate k-square S_k. Without loss of generality, assume that no two points of P have the same

Priya Ranjan Sinha Mahapatra is with the Department of Computer Science and Engineering, University of Kalyani, Kalyani-741235, India. $Email:$ $priya\_cskly@yahoo.co.in$

x or y coordinates. Let $x(p)$ and $y(p)$ denote the x-coordinate and the y-coordinate of any point p respectively. The size of a square is represented by the length of it's side. We have the following observation.

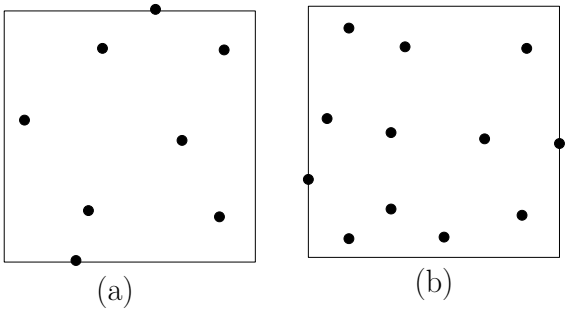

Fig. 1. (a) Type-1 k-square (b) Type-2 k-square.

Observation 1: At least one pair of opposite sides of S_k must contain points from P.

For each optimization problem, there is a corresponding decision problem that asks whether there is a feasible solution for some particular measure [32]. The decision version of the problem of locating S_k can be stated as *"given a length α, does there exist a square of size α that encloses at least k points of P?"*. The following section describes an $O((n-k)\log(n-k))$ algorithm to solve the decision problem for given k and α.

III. An algorithm to solve the decision problem

Let P_b, P_t, P_l, P_r and P_f be five subsets of P such that $P = P_b \cup P_t \cup P_l \cup P_r \cup P_f$ and all the subsets are not necessarily mutually disjoint. We define P_b and P_t as the set of $(n-k)$ bottom most and $(n-k)$ top most points of P respectively; P_l and P_r are the set of $(n-k)$ left most points and $(n-k)$ right most points of P respectively; and $P_f = P - P'$ where $P' = P_b \cup P_t \cup P_l \cup P_r$.

Note that if $k > \frac{3n}{4}$ then P_f must contain at least one point of P. The following observation follows from the above definitions.

Observation 2: For $k > n/2$, S_k must contain all the points of P_f.

Proof: Let p be any point of the set P_f. At least $(n-k)$ elements are on the right side of p. The position of p in the left to right ordering of P are at most k. Therefore there are $(n-k+i)$ number of points of P on the left of p for $0 \leq i \leq 2k-n-1$. Consequently at most $(k-1)$ points are on left of p. Hence right boundary of S_k is on right side of p. Similarly left, top and bottom boundaries of S_k are on left, top and bottom sides of p respectively. Hence the observation follows. ■

Let R_f be the minimum area axis-parallel rectangle enclosing the point set P_f. Suppose the length of the longest side of R_f is λ and the left, right, top and bottom boundaries of the rectangle R_f contain the points p_l, p_r, p_t and p_b respectively (See Figure 2). We define $Max\text{-}square(\alpha)$, $\alpha \geq \lambda$ as an axis-parallel square of size α that includes the point set P_f and the total number of points enclosed from P is maximized. It is

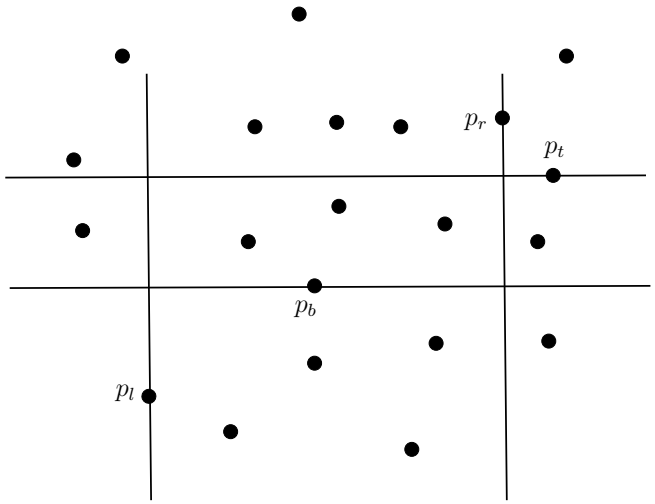

Fig. 2. Proof of Observation 2.

easy to see that the bottom, top, left and right boundaries of $Max\text{-}square(\alpha)$ must lie within the ranges $[y(p_t)-\alpha, y(p_b)]$, $[y(p_t), y(p_b)+\alpha]$, $[x(p_r)-\alpha, x(p_l)]$ and $[x(p_r), x(p_l)+\alpha]$ respectively.

To locate $Max\text{-}square(\alpha)$ among the set $P' \cup \{p_t, p_b, p_l, p_r\}$, we use *sweep line* paradigm combined with *segment tree* [33], [34] as data structure. Our algorithm makes horizontal and vertical sweeps. Below we describe the algorithm for horizontal sweep to compute $Max\text{-}square(\alpha)$ whose bottom side is aligned with a point from P. Look for all squares of size α whose bottom and left boundaries are within the range $[y(p_t)-\alpha, y(p_b)]$ and $[x(p_r)-\alpha, x(p_l)]$ respectively. Now consider possible positions of the left boundary of $Max\text{-}square(\alpha)$ within the above mentioned range such that the left boundary or the right boundary passes through a point of P. Notice that all squares with these restrictions include the point set P_f. Therefore the points in the set $P' \cup \{p_t, p_b, p_l, p_r\}$ are the only points required to be processed for obtaining $Max\text{-}square(\alpha)$ and the number of such points is at most $4(n-k+1)$.

For each point $p_i \in P' \cup \{p_t, p_b, p_l, p_r\}$, compute the interval I_i as the intersection of the intervals $[x(p_r)-\alpha, x(p_l)]$ and $[x(p_i)-\alpha, x(p_i)]$. Let L be the horizontal line at height $y(p_t)-\alpha$. We orthogonally project the endpoints of all intervals I_i onto the horizontal line L. These left to right sorted projected endpoints induces partitioning of L. The partitions thus induced are the *elementary intervals*. We now describe the construction of segment tree for these elementary intervals. The skeleton of the segment tree is a balanced tree T whose leaves correspond to the elementary intervals sorted from left to right. The internal node v of T correspond to an interval that is the union of elementary intervals of the leaves in the subtree rooted at v. (This implies that the interval associated with an internal node is the union of the intervals of its two children.) Each node in T stores two information; (a) the interval that the node covers, and (b) the weight of the node. We define the weight of a node later on. Initially the weight

of each leaf is zero, and the interval covered by each leaf is its elementary interval. A segment tree for a set of n intervals can report the number of intervals that contain the query point in $O(\log n)$ time [33]. Moreover, such a segment tree can built in $O(n \log n)$ time [33]. It should be noted that each internal node of segment tree stores the elementary intervals of the leaves in the subtree rooted at v. But here, each internal node of the segment tree T stores its weight and the span of the interval attached with this internal node. Such a segment tree on a set of n intervals requires $O(n)$ storage space [33].

In initial step of sweep line algorithm, we want to locate $Max\text{-}square(\alpha)$ when its lower side is constrained to coincide with L. Here $Max\text{-}square(\alpha)$ will lie on the horizontal slab S of height α and the lower side of S coincides with L. To define the weight of an interval associated with an internal node, we first define the weight of an elementary interval e as the number of intervals I_i corresponding to the points of $(P' \cup \{p_l, p_r, p_t, p_b\}) \cap S$ that contain interval e. Then for each internal node v of T, we determine the interval associated with v as the union of the elementary intervals of the leaves in the subtree rooted at v. This can be done in $O(n-k)$ time using bottom-up traversal in T. The weight of an internal node v is the maximum of the weights of its children plus the weight of the interval associated with v. For each point $p_i \in (P' \cup \{p_l, p_r, p_t, p_b\}) \cap S$, insert interval I_i into T. At each insertion, at most $O(\log(n-k))$ internal nodes have to updated. Consequently weight information of at most $O(\log(n-k))$ nodes are updated (see details in [35], [33]). Note that the root of T stores the maximum weight for $Max\text{-}square(\alpha)$ when it is constrained to touch L. It is also easy to see that the best location for the left boundary of $Max\text{-}square(\alpha)$ is within an elementary interval e^* where the weight of e^* is maximum among all elementary intervals. To locate the left boundary of $Max\text{-}square(\alpha)$, move from the root towards the leaf, each time picking the child with larger weight. The leaf thus reached is the elementary interval where the $Max\text{-}square(\alpha)$ attains the maxima.

In the next phases of the algorithm we sweep the horizontal slab S vertically inducing updates in the segment tree T whenever a point p_i disappears from lower side of S or a point p_i appears on the upper side of S. Lastly when the top side of S coincides with the horizontal line at height $y(p_b)+\alpha$, the algorithm terminates. Each of these events can be easily handled using segment tree either by deleting an interval I_i from T or by inserting an interval I_i into T. Correspondingly the nodes in T, which are affected due to deletion or insertion of intervals I_i, are accordingly updated with new weights. Storing the maximum weight of the root of T so far found during this sweep line algorithm, the best location of $Max\text{-}square(\alpha)$ can be achieved. Each insertion or deletion of an interval takes $O(\log(n-k))$ time and there are $O(n-k)$ events. Thus, we obtain the following theorem.

Theorem 1: For given $\alpha > \lambda$, the axis-parallel square $Max\text{-}square(\alpha)$ containing maximum number of points from P and enclosing point set P_f can be computed in $O((n-k)\log(n-k))$ time using $O(n)$ space.

IV. An efficient algorithm to locate k-square for large values of k

In this section, an optimization technique is used to design an efficient algorithm for locating S_k for large values of k ($> \frac{n}{2}$). The algorithm to find $Max\text{-}square(\alpha)$ described in the previous section is used as a subroutine to locate S_k for $k > \frac{n}{2}$. From Observation 1, we can conclude that either top and bottom sides of S_k contain points of P or left and right sides of S_k contain points of P (See Figure 1). Here we call these two configurations of S_k as *Type-1* and *Type-2*. Without loss of generality, assume that top and bottom sides of S_k contain points from P (i.e. Type-1). The other case where left and right sides of S_k contain points of P, can be handled in similar manner (i.e. Type-2). Let $Q = <p_1, p_2, \ldots, p_m>$ be an ordering of points of the set $P' \cup \{p_l, p_r, p_t, p_b\}$ in increasing order of their y-coordinate values. Consider Δ to be the list of $O((n-k)^2)$ vertical distances $(y(p_j) - y(p_i))$, $j > i$ for each pair of points p_i and $p_j \in Q$.

Our objective is to locate S_k for a given value k such that $Max\text{-}square(\alpha)$ contains k points of P and the value $\alpha \in \Delta$ is minimized. To locate S_k for the given value of k, an efficient optimization technique is proposed. This technique works in similar with median finding algorithm [36]. We iteratively reduce the size of Δ by *prune and search technique* without explicitly computing $O((n-k)^2)$ elements of Δ. Let Δ_i represent the list of vertical distances at i^{th} iteration. At i^{th} iteration we reduce the size of Δ_i by $\frac{1}{4}$. Initially $\Delta_0 = \Delta$. Observe that for any $p_i \in Q$, $(y(p_j) - y(p_i)) < (y(p_{j+1}) - y(p_i))$ for $m > j > i$. Without loss of generality, let the indices of the points p_l, p_r, p_t and p_b remain same in Q also. Let us denote the set of vertical distances generating Δ_0 as follows.

$$\Psi_1 = \{y(p_t) - y(p_1), y(p_{t+1}) - y(p_1), \ldots, y(p_m) - y(p_1)\}$$

$$\Psi_2 = \{y(p_t) - y(p_2), y(p_{t+1}) - y(p_2), \ldots, y(p_m) - y(p_2)\}$$

$$\vdots$$

$$\Psi_i = \{y(p_t) - y(p_i), y(p_{t+1}) - y(p_i), \ldots, y(p_m) - y(p_i)\}$$

$$\vdots$$

$$\Psi_b = \{y(p_t) - y(p_b), y(p_{t+1}) - y(p_b), \ldots, y(p_m) - y(p_b)\}$$

Note that the elements in each sequence Ψ_i are in nondecreasing order. At j^{th} iterative step of the algorithm the current search space Δ_j is reduced by pruning the Ψ_i's. Here, either upper or lower portion of Ψ_i is pruned. Therefore, each Ψ_i sequence can be represented by lower and upper indices of the original sequence. For any point $p_i \in Q$, median element of the corresponding sequence Ψ_i is $|y(p_i) - y(p_{\lfloor \frac{l_1+l_2}{2} \rfloor})|$ where l_1 and l_2 are the lower and upper indices of the sequence Ψ_i. We denote the median element of Ψ_i as $med(\Psi_i)$. So computing the median of the sequence of vertical distances corresponding to any point $p_i \in Q$ requires only a constant time arithmetic operation on the array indices.

We represent each Ψ_i as a vertical strip parallel to the y-axis (see Figure 3). All the vertical strips (Ψ_i's) are arranged along the x-axis such that $med(\Psi_i)$'s fall on the x-axis and the median values are in nonincreasing order along the x-axis. Again the elements of each Ψ_i are arranged in nondecreasing order parallel to the y-axis. At initial step of iteration, all medians $med(\Psi_1), med(\Psi_2), \ldots, med(\Psi_b)$ are in nonincreasing order. This ordering may change in subsequent iterations due to pruning of Ψ_i's. Therefore at each iteration, we need to rearrange Ψ_i's such that $med(\Psi_i)$'s are in nonincreasing order. Let $\Psi_1, \Psi_2, \ldots, \Psi_b$ be an arrangement of the sequences in Δ_j such that $med(\Psi_1) \geq med(\Psi_2) \geq \ldots \geq med(\Psi_b)$. At j^{th} iteration we find an index c such that $\sum_{i=1}^{c} |\Psi_i|$ is half of the size of Δ_j. Observe that the size of Δ_j is at most $\frac{3}{4}$ of the size of Δ_{j-1}. Consider $med(\Psi_c)$ as α and compute $Max\text{-}square(\alpha)$. If $Max\text{-}square(\alpha)$ encloses at least k points of P, then size of S_k is less than or equal to α and we can ignore the elements in Δ_j greater than $med(\Psi_c)$. Note that all the $med(\cdot)$ values corresponding to $\Psi_1, \Psi_2, \ldots, \Psi_{c-1}$ are greater than $med(\Psi_c)$. Therefore for each $i,\ 1 \leq i < c$ we can delete upper half of Ψ_i. In case, $Max\text{-}square(\alpha)$ encloses less than k points, we similarly delete lower half of each Ψ_i for $c \leq i \leq b$. Now continue with the subsequent iterations until we end up at an iteration z such that size of Δ_z is constant.

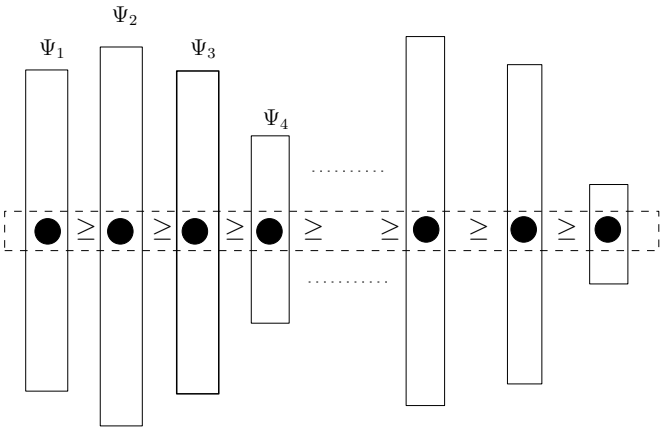

Fig. 3. Arrangement of Ψ_i's.

Lemma 1: At every iterative step the size of the current solution space is reduced by a factor of $\frac{1}{4}$.

Proof: At j^{th} iteration, either we discard upper half of $\Psi_1, \Psi_2, \ldots, \Psi_{c-1}$ or lower half of $\Psi_c, \Psi_{c+1}, \ldots, \Psi_b$. As the total number of elements in the sequences $\Psi_1, \Psi_2, \ldots, \Psi_{c-1}$ is $\frac{1}{2}$ of size of Δ_j, we can discard at least $\frac{1}{4}$ elements of Δ_j. Similar amount of elements is discarded for pruning of lower half.

■

Now we have the following theorem.

Theorem 2: Given a set P of n points in the plane and an integer k ($> \frac{n}{2}$), the smallest area square containing at least k points of P can be located in $O(n + (n-k)\log^2(n-k))$ time using linear space.

Proof: Partitioning the set P to generate subsets P_b, P_t, P_l, P_r and P_f requires $O(n)$ time. Sorting the points of the sets P_b and P_t with respect to their y-coordinates requires $O((n-k)\log(n-k))$ time. We do not store the Ψ_i's explicitly. Instead, for all Ψ_i's, we maintain an array $\mathcal{A}$ whose each element $\mathcal{A}[i]$ contains the index information l_1 and l_2 for Ψ_i at each iteration. So for each Ψ_i we need only an additional constant amount of space. Altogether in linear amount of space we can execute our algorithm. Time complexity can be established from the following algorithmic steps at iteration j.

i) Computation of $med(\Psi_i)$ for each i requires constant amount of time.
ii) Sorting the set of all medians $med(\Psi_1), med(\Psi_2), \ldots, med(\Psi_b)$ takes $O((n-k)\log(n-k))$ time.
iii) Determining c such that $\sum_{i=1}^{c} |\Psi_i|$ is half of the size of Δ_j, needs $O(n-k)$ time.
iv) Computation of $Max\text{-}square(med(\Psi_c))$ takes $O((n-k)\log(n-k))$ time (see Theorem 1).
v) We maintain the index structure of the arrays Ψ_i. This involves updating of l_1 and l_2 for each Ψ_i when half of it's elements are discarded. This step requires constant amount of time for each Ψ_i.

From Lemma 1, we get that at j^{th} iterative step at least $\frac{M}{4}$ elements are discarded where M denotes the size of Δ_j. This leads to the following recurrence relation. $T(M) = T(3M/4) + O((n-k)\log(n-k))$
$= O((n-k)\log^2(n-k))$
Hence the theorem. ■

Now we prove the following result using Theorem 1 and the optimization technique described in this section.

Theorem 3: Given a set P of n points in the plane and an integer k ($\leq n$), the smallest area square containing at least k points of P can be located in $O(n\log^2 n)$ time using linear amount of space.

Proof: Let $Q = <p_1, p_2, \ldots, p_n>$ be the ordering of points of P in nondecreasing order of their y-coordinate values. As described in Section IV, the algorithm locates S_k such that each of it's bottom and top side contain at least one point of P. For large values of k, a restriction was imposed on the vertical distances to compute Δ. In this case, this restriction is waived in the following fashion. For any point $p_i \in P$, the sequence Ψ_i of vertical distances is as below.

$$\Psi_i = \{y(p_{i+1}) - y(p_i), y(p_{i+2}) - y(p_i), \ldots, y(p_n) - y(p_i)\}$$

Thus we have the following $(n-1)$ sequences of vertical distances.

$$\Psi_1 = \{y(p_2) - y(p_1), y(p_3) - y(p_1), \ldots, y(p_n) - y(p_1)\}$$

$$\Psi_2 = \{y(p_3) - y(p_2), y(p_4) - y(p_2), \ldots, y(p_n) - y(p_2)\}$$

$$\vdots$$

$$\Psi_i = \{y(p_{i+1}) - y(p_i), y(p_{i+2}) - y(p_i), \ldots, y(p_n) - y(p_i)\}$$

$$\vdots$$

$$\Psi_{n-2} = \{y(p_n) - y(p_{n-2}), y(p_{n-1}) - y(p_{n-2})\}$$

$$\Psi_{n-1} = \{y(p_n) - y(p_{n-1})\}$$

Therefore, the number of elements in all Ψ's is $\frac{n(n-1)}{2}$. Observe that $\Delta = \bigcup_{i=1}^{n-1} \Psi_i$. We now design an deterministic algorithm to solve the decision problem in similar way (as described in section III) on the point set P instead of $P' \cup \{p_t, p_b, p_l, p_r\}$. This algorithm runs in $O(n \log n)$ time. Here the vertical distances are taken from the sequences $\Psi_1, \Psi_2, \ldots, \Psi_{n-2}, \Psi_{n-1}$ and the pruning process on the vertical distances is performed in similar manner as described in Section IV. Therefore, we have the following recurrence relation.
$T(M) = T(3M/4) + O(n \log n)$
$= O(n \log^2 n)$; M denotes the number of vertical distances on j-th iteration. ■

V. An algorithm to locate k-rectangle

In this section, we outline an algorithm to locate k-rectangle S_k for large values of k ($> \frac{n}{2}$). Observe that each side of S_k must contain a point from the set $P' \cup \{p_l, p_r, p_t, p_b\}$. Let $Q' = <p'_1, p'_2, \ldots, p'_m>$ be an ordering of points of the set $P' \cup \{p_l, p_r, p_t, p_b\}$ in increasing order of their y-coordinate values and Δ' be the list $O((n-k)^2)$ horizontal distances $(y(p'_j) - y(p'_i))$, $j > i$ for each pair of points p'_i and $p'_j \in Q'$. As the area of S_k is determined by the length of its horizontal and vertical sides, we need to consider both the list of vertical distances in Δ and the list of horizontal distances in Δ'. Here a rectangle of *size* $\alpha \times \beta$ means a rectangle having a pair of horizontal sides each of length α, other pair of vertical sides each of length β. We now define $Max\text{-}rectangle(\alpha, \beta)$, $\alpha, \beta \geq \lambda$ as a rectangle of size $\alpha \times \beta$ that include point set P_f and the total number of points enclosed from P is maximized. Observe that Theorem 1 can be extended to compute $Max\text{-}rectangle(\alpha, \beta)$, $\alpha, \beta \geq \lambda$ in $O((n-k)\log(n-k))$ time using $O(n)$ space. This observation leads to the following theorem.

Theorem 4: Given a set P of n points in the plane and an integer k ($> \frac{n}{2}$), the smallest area rectangle containing at least k points of P can be located in $O(n + (n-k)^3 \log^2(n-k))$ time using $O(n)$ space.

Proof: Our algorithm to locate k-rectangle S_k works in two passes. In first pass, for each horizontal distance $\alpha \in \Delta'$, we identify a minimum vertical distance $\beta \in \Delta$ such that the $Max\text{-}rectangle(\alpha, \beta)$ contains k points of P. It should be mentioned that β can be found by similar iterative method as described in Section IV. As horizontal distances in the set Δ' is $O((n-k)^2)$ and β can be found in $O((n-k)\log^2(n-k))$ time, we conclude that S_k can be located in $O(n+(n-k)^3 \log^2(n-k))$ time. Note that we are considering one horizontal distance at a time and therefore space requirement remains linear. ■

We conclude this section with the remark that a similar approach can also find $k\text{-}rectangle$ for all values of k in $O(n^3 \log^2 n)$ time and $O(n)$ space.

VI. Conclusions

Given a set P of n points in two dimensional plane and an integer k ($\leq n$), we have considered the problem of locating a minimum area square that encloses at least k points of P. A k point enclosing square (rectangle) S_k is said to be a $k\text{-}square$ ($k\text{-}rectangle$) if there does not exist another square (rectangle) having area less than that of S_k and containing k points from P. We first propose a simple deterministic algorithm to locate $k\text{-}square$ for large values of k ($> \frac{n}{2}$). Then it is shown that this algorithm can be used to find $k\text{-}square$ for all values of k. Moreover, we outline an algorithm to locate $k\text{-}rectangle$ for large values of k ($> \frac{n}{2}$) and all values of k.

One may hope faster algorithm for each of the above optimization problem. Moreover, it is interesting to compute at least two squares (respectively rectangles) that together contains at least k points of P. In this case, the area of the largest square (respectively rectangle) is required to be minimized.

References

[1] B. Korte and J. Vygen, *Combinatorial Optimization Theory and Algorithms*, 2nd ed. Berlin, Heidelberg, New York: Springer-Verlog.

[2] S. Das, P. P. Goswami, and S. C. Nandy, "Smallest k-point enclosing rectangle and square of arbitrary orientation," *Inf. Process. Lett.*, vol. 94, no. 6, pp. 259–266, 2005.

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

[4] A. Datta, H.-P. Lenhof, C. Schwarz, and M. H. M. Smid, "Static and dynamic algorithms for k-point clustering problems," in *WADS*, 1993, pp. 265–276.

[5] D. Eppstein and J. Erickson, "Iterated nearest neighbors and finding minimal polytopes," *Discrete & Computational Geometry*, vol. 11, pp. 321–350, 1994.

[6] M. Smid, "Finding k points with a smallest enclosing square," in *Report MPI-92-152*, 1995.

[7] A. Datta, H.-P. Lenhof, C. Schwarz, and M. H. M. Smid, "Static and dynamic algorithms for k-point clustering problems," *J. Algorithms*, vol. 19, no. 3, pp. 474–503, 1995.

[8] T. M. Chan, "Geometric applications of a randomized optimization technique," *iscrete Comput. Geom.*, vol. 22, no. 4, pp. 547–567, 1999.

[9] J. Matoušek, "On geometric optimization with few violated constraints," *Discrete Comput. Geom.*, vol. 14, pp. 365–384, 1995.

[10] T. Weise, *Global Optimization Algorithms: Theory and Application*, 2nd ed., 2009.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Intoduction To algorithms*, 5th ed. PHI, 2001.

[12] J. Kleinberg and E. Tardos, *Algorithm Design*, 1st ed. Pearson Education, 2006.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. New York: Addison-Wesley, 1989.

[14] K. A. D. Jong, *Evolutionary Computation: A Unified Approach*. MIT Press.

[15] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*. Englewood Cliffs, NJ 07632: Prentice-Hall, 1994.

[16] L. Davis, Ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[17] Bishop and Christopher, *Pattern Recognition and Machine Learnin*. Berlin, New York: Springer, 2006.

[18] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading: Addison-Wesley, 1974.

[19] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[20] Z. Drezner and H. W. Hamacher, *Facility location: applications and theory*, 1st ed. Berlin, Heidelberg, New York: Springer, 2001.

[21] R. Z. Farahani and M. Hekmatfar, *Facility Location: Concepts, Models, Algorithms and Case Studies*. Berlin, Heidelberg: Springer-Verlog, 2009.

[22] R. L. Francis and L. Mirchandani, *Discrete Location Theory*. New York: Wiley, 1990.
[23] R. L. Francis, F. Leon, J. McGinnis, and J. A. White, *Facility Layout and Location: An Analytical Approach*. New York: Prentice-Hall, 1992.
[24] H. C. Andrews, *Introduction to mathematical techniques in pattern recognition*. R.E. Krieger Pub. Co, 1983.
[25] T. Asano, B. K. Bhattacharya, J. M. Keil, and F. Yao, "Clustering algorithms based on minimum and maximum spanning trees," in *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, 1988, pp. 252–257.
[26] J. A. Hartigan, *Clustering Algorithms*. New York: John Wiley & Sons, 1975.
[27] L. Kaufman and P. Roussenw, *Finding Groups in Data: An Introduction to Cluster Analysis*. NY, US: John Wiley & Sons, 1990.
[28] B. S. Everitt, *Cluster Analysis*. Halsted Press, third edition, 1993.
[29] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[30] R. C. Dubes and A. K. Jain, "Clustering techniques : The user's dilemma," *Pattern Recognition*, vol. 8, pp. 247–260, 1976.
[31] S. Majumder and B. B. Bhattacharya, "Density or discrepancy: A vlsi designers dilemma in hot spot analysis," *Information Processing Lette*, vol. 107, pp. 353–37, 2008.
[32] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. McGraw-Hill, 2006.
[33] M. d. Berg, O. Cheong, M. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, April 2008.
[34] H. Edelsbrunner, *Algorithms in Computation Geometry*. Berlin, Heidelberg, New york, London, Paris, Tokyo: Springer-Verlog, 1987.
[35] K. Mehlhorn, *Data structures and algorithms 3: multi-dimensional searching and computational geometry*. New York, NY, USA: Springer-Verlag New York, Inc., 1984.
[36] M. Blum, R. W. Floyd, V. Pratt, R. Rivest, and R. Tarjan, "Time bounds for selection," *Journal of Computing System Sciences*, vol. 7, pp. 448–461, 1973.