# Covering Points by Isothetic Unit Squares

Priya Ranjan Sinha Mahapatra * Partha P. Goswami * Sandip Das †

**Abstract**

Given a set $P$ of $n$ points in $\mathbb{R}^2$, we consider two related problems. Firstly, we study the problem of computing two isothetic unit squares which may be either disjoint or intersecting (having empty common zone) such that they together cover maximum number of points. The time and space complexities of the proposed algorithm for this problem are both $O(n^2)$. We also study the problem of computing $k$ disjoint isothetic unit squares, admitting sliceable $k$-partitions, which maximizes the sum of the points covered by them. Our proposed algorithm for this problem runs in time $O(k^2n^5)$ and uses $O(kn^4)$ space. To solve this problem, we propose an optimal $O(n \log n)$ time and $O(n)$ space algorithm which computes $O(n)$ isothetic unit squares each covering maximum number of points and having one side aligned with a point from $P$.

## 1 Introduction

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of $n$ given points in $\mathbb{R}^2$. Diaz Banez et. al. [1] recently studied the problem of computing two disjoint isothetic unit squares such that the sum of the number of points covered by them is maximum and proposed an $O(n^2)$ time algorithm to solve it. It is also mentioned in the same paper that the problem finds applications in facility location, Pattern Recognition, Classification, etc. In this paper we consider two variations of the problem. So far as we are aware, none of these problems have been studied before. In one variation, we consider the problem of computing a pair of isothetic unit squares together maximizing the number of points covered. The pair of squares may be disjoint or intersecting, in case of later their overlapping zone is empty. The motivation for studying this problem is that there may be a better solution if, in addition of being disjoint, we allow the squares to be intersecting also. We present an algorithm for this generalization that runs using $O(n^2)$ time and space.

In the other variation we want to compute $k$ disjoint isothetic unit squares for a given integer $k$ ($k \leq n$) such that they together cover maximum number of points. For solving this problem, we solve another subproblem. Given a point set $P$, for each point $p \in P$, compute an isothetic unit square covering maximum number of points and having one side aligned with $p$. It is mentioned in [1] that, by reduction to uniform gap problem [2], the lower bound for the time complexity of this subproblem can be proved to be $\Omega(n \log n)$. In this paper, we present an optimal algorithm for the subproblem which runs in $O(n \log n)$ time using $O(n)$ space.

Our algorithm for computing $k$ disjoint isothetic unit squares is based on dynamic programming paradigm and uses the above subproblem as a subroutine. The time and space complexities of the algorithm are $O(k^2n^5)$ and $O(kn^4)$ respectively.

Without loss of generality we assume that no two points have the same $x$- or $y$-coordinates. We pre-sort the points (in non-decreasing order) on their $x$- and $y$-coordinates and store them in lists $L_x$ and $L_y$ respectively. The coordinates of a generic point $p$ are denoted by $(p_x, p_y)$ and those for a specific point $p_i$ are denoted by $(p_{x_i}, p_{y_i})$. In the rest of the paper, a square means an isothetic unit square.

## 2 Candidate Isothetic Unit Square

### 2.1 Characterization

Let $S$ be a square covering maximum number of points from $P$. Let $P' \subseteq P$ be the set of points covered by $S$. Observe that $S$ can always be repositioned, without altering the points covered by it, so that two adjacent sides of $S$ are aligned with two points from $P$ and these points may not belong to $P'$. See Figure 1 for a demonstration. Sometimes the adjacent sides may be aligned to same point, and in that case, the point is at one corner of $S$. For each point $p_i \in P$, we compute the square $S_i$ having one side aligned with $p_i$ and containing maximum number of points. We call each of these squares a *Candidate Isothetic Unit Square* (CIUS). Observe that one point $p_i \in P$ can give rise to a linear number of squares that all have the same number of points. For our case, any one can be taken as a CIUS. The total number of possible CIUS's is thus at most $4n$. For computing the CIUS's, our algorithm performs four similar passes. Below we describe only the pass in which CIUS's, each having bottom side aligned with a point, are computed.

*Department of Computer Science and Engineering, University of Kalyani, Kalyani, India

†Indian Statistical Institute, Kolkata, India

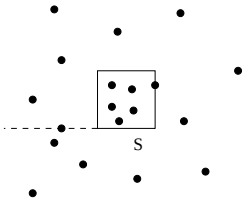


Figure 1: Candidate isothetic unit square

### 2.2 Algorithm for computing CIUS's

Let the function $left(p_i)$ output the minimum entry in $L_x$, say $p_j$, such that $p_{x_i} - p_{x_j}$ is less than or equal to unity. In a similar way, we define $right(.)$, $bottom(.)$, and $top(.)$.

We construct a balanced binary search tree $T$ with search key as the $x$-coordinate values of the points in $P$. We attach positive integral variables $\mathcal{M}$ and $\mathcal{C}$ with each node $v$ of $T$.

The algorithm processes all points $p_1, p_2, \ldots, p_n$ from bottom to top one at a time. For this, we use two sweep lines named bottom sweep line $BS$ and top sweep line $TS$.

To start with, the variables $\mathcal{M}$ and $\mathcal{C}$ corresponding to all nodes are initialized with zero and both the sweep lines are aligned with the bottom most point.

The sweep line $TS$ is moved up one point at a time. For each point $p$ encountered by the sweep line, if the distance of $p$ from the current position of the sweep line $BS$ is less than or equal to unity then $p$ is inserted into $T$ in the following manner.

For interval $[left(p)_x, p_x]$, locate the split node $v$ in $T$. During the search for the leaf node containing $left(p)_x$ along the left subtree of $v$, if the search path goes left at a node $\gamma$, then increment $\mathcal{C}$ of the right child of $\gamma$. In case the right child is a leaf node, increment its $\mathcal{M}$ instead of $\mathcal{C}$. Similar action is performed while searching for the leaf node containing $p_x$ on the right subtree of the split node $v$. Finally, the $\mathcal{M}$ values of the two leaf nodes containing $left(p)_x$ and $p_x$ are increased by one. Then update the $\mathcal{M}$ values associated with the internal nodes by the sum of its $\mathcal{C}$ and the maximum value of $\mathcal{M}$ of its children along the paths from $left(p)_x$ and $p_x$ towards the split node $v$ and then towards root.

When the distance of the current point $p$ from $BS$ becomes greater than unity, the sweep line $TS$ stops advancing. The sweep line $BS$ is then moved up one point at a time and for each point $q$ encountered by $BS$, if the distance of $TS$ from $q$ is greater than or equal to unity then report a $CIUS$ and delete the point $q$ as follows.

Let $S_q$ be the $CIUS$ with bottom boundary aligned with the point $q$. Observe that the number of points covered by $S_q$ is equal to the $\mathcal{M}$ value at the root of the tree $T$. To locate the left boundary of $S_q$, start from the root of the tree $T$. Compare the $\mathcal{M}$ values of the two children of the root and go to that node having larger $\mathcal{M}$ values. Continue the process till a leaf node $\alpha$ is reached. Then the left boundary of $S_q$ is aligned with the point stored in $\alpha$ and report $S_q$ along with the number of points inside it.

The deletion operation is same as insertion operation with the following exception. During searching for the nodes containing $left(q)_x$ and $q_x$, instead of incrementing, we decrement $\mathcal{C}$'s and $\mathcal{M}$'s by one as necessary. The subsequent updation is exactly same as that in the insertion operation.

When the distance of $q$ from $TS$ becomes smaller than unity and $TS$ is not at the top most point then the sweep line $BS$ stops advancing. Above process is continued till all the points are inserted and deleted from the tree $T$.

For a point $p$, each of insertion, deletion and reporting operation takes $O(\log n)$ time. Since for each point in $P$, these operations are executed only once, they together takes $O(n \log n)$ time. It is also clear that total space requirement is $O(n)$.

**Theorem 1** *Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of $n$ points on a plane. Then all $CIUS$'s, as defined above, can be computed in $O(n \log n)$ time using $O(n)$ space.*

## 3 Optimum $k$-disjoint squares

In this section we study the problem of computing $k$ disjoint squares, admitting sliceable $k$-partitions, such that they together cover maximum number of points. Our algorithm for this problem is based on dynamic programming paradigm and it partitions the point set recursively.

Given any non-empty subset $Q \subseteq P$, the optimum square covering maximum number of points from $Q$ can be computed in time $O(n \log n)$ in the worst case. Now, using this result and the dynamic algorithm for partitioning a point set by Mukherjee et. al. [3], we can have the following result.

**Theorem 2** *Given a set $P$ of $n$ points in $\mathbb{R}^2$, $k$ disjoint isothetic unit squares, admitting sliceable $k$-partitions, together covering maximum number of points can be found using $O(k^2n^5)$ time and $O(kn^4)$ space.*

## 4 Optimum pair of squares

In this section we consider the generalized problem of computing a pair of squares $S1$ and $S2$ such that the sum of the number of points covered by them is maximum and $S1 \cap S2$ does not contain any point from $P$.

If we allow the pair of squares to be intersecting with empty overlapping zone, then the squares of the optimum pair may not be members of the set of optimum $CIUS$'s as computed in Section 2.2.

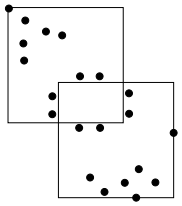

Figure 2: Optimum intersecting pair

Figure 2 demonstrate this. Here none of the squares of the pair is an optimum $CIUS$ though they together constitute an optimum pair.

**Lemma 3** *Let $S$ be the square containing maximum number of points. Then the overlapping zone of an optimum intersecting pair of squares lies inside $S$.*

However, unfortunately, we can not exploit this Lemma for developing efficient algorithm for finding the optimum intersecting squares having empty intersection zone. In the following, we present a general algorithm which report the optimum pair $S1$ and $S2$ where $S1 \cap S2$ does not contain any point from $P$.

We use the implicit grid obtained by drawing vertical and horizontal lines through each point of the given set $P$ (see Figure 3). Observe that the resulting $n^2$ grid points can be traversed by using the previously defined sorted lists $L_x$ and $L_y$ in appropriate manner. We denote, by $(i, j)$, the grid point generated by the intersection of the vertical line through the point which is the $i$-th entry in $L_x$ and the horizontal line through the point which is the $j$-th entry in $L_y$. If, for some grid point $(i, j)$, the corresponding $x$- and $y$-coordinates are of the same point, then the grid point is occupied by a point from $P$. For brevity, we sometimes specify a grid point by its coordinates also. Before describing the algorithm in detail, we first give a brief overview.

### 4.1 Theme of the algorithm

Let $S_{i,j}$ be a square having bottom right corner at the grid point $(i, j)$. From the points contained by $S_{i,j}$, we discard those points $p_s$ for which there exists a point $p_t$ such that

$$p_{x_s} < p_{x_t} \text{ and } p_{y_s} > p_{y_t} \quad (1)$$

Effectively, we thus get a stair case ($AB$ in Figure 3) which is a rectilinear $xy$- monotone chain defined by the points after removal. Observe that the portion of $S_{i,j}$ lying between its bottom right corner and the stair case does not contain any point from $P$. Let $p_{q_1}, p_{q_2}, \ldots, p_{q_{r-1}}, p_{q_r}, \ldots, p_{q_m}$ be the points which form the stair case as we traverse it from left to right. We form a set $V_{q_1,q_m}$ of grid points as follows. For each

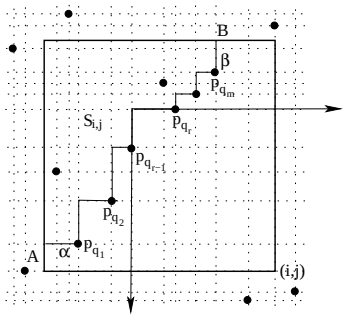


Figure 3: Stair case within unit square $S_{i,j}$

consecutive pair of points $p_{q_{r-1}}$ and $p_{q_r}$, the grid point having coordinates $(p_{x_{q_{r-1}}}, p_{y_{q_r}})$ is a member of $V_{q_1,q_m}$. We consider, for each member $v \in V_{q_1,q_m}$, the square $S_v$ containing maximum number of points from the subset of points inside the rectangle defined by the grid points $v$ and $(p_{x_{max}}, p_{y_{min}})$ where $p_{x_{max}}$ is the maximum of the $x$-coordinates occurred in $P$ and $p_{y_{min}}$ is the minimum $y$-coordinate. Let $C_v$ be the corresponding count of the number of points contained by $S_v$. Compute $C_{V_{q_1,q_m}} = \max_v(C_v)$ and let the corresponding square be $S_{V_{q_1,q_m}}$.

In addition, we consider two more grid points $\alpha$ and $\beta$, where $\alpha$ is on the bottom part of the stair case and $\beta$ is on its top part. The grid point $\alpha$ may coincide with the boundary or closest to it among all those grid points lying on the bottom segment ($[A, p_{q_1}]$ in Figure 3) of the stair case. The grid point $\beta$ can be identified in a similar manner. Observe that any one or both of these two additional grid points may coincide with a point from $P$. Let $S_\alpha$ (respectively $S_\beta$) be the square containing maximum number of points from the subset of points inside the rectangle defined by the grid points $\alpha$ ($\beta$) and $(p_{x_{max}}, p_{y_{min}})$; $C_\alpha$ ($C_\beta$) be the corresponding count. Let $C_V = \max(C_{V_{q_1,q_m}}, C_\alpha, C_\beta)$ and the corresponding square be $S_V$.

Observe that the pair $S_{i,j}$ and $S_V$ may be either disjoint or intersecting with empty overlapping zone. So $S_{i,j}$, $S_V$ forms a candidate pair. Each grid point $(i, j)$ gives one such candidate pair and our algorithm finds all such candidate pairs and computes the optimum among them.

In the above, we have identified candidate pairs by constructing the stair case relative to the bottom right corner position of $S_{i,j}$. Similar candidate pairs may also be obtained by considering stair cases relative to other corner positions of $S_{i,j}$. However, since they are similar we describe only the bottom right corner case.

In order to reduce the over all complexity, instead of processing each $S_{i,j}$ separately we process the entire point set in the following three phases.

### 4.2 Phase I

Here we consider $S_{i,j}$ to be the square whose top left corner is at the grid point $(i, j)$ and let $C_{i,j}$ be the number of points from $P$ inside $S_{i,j}$. Let $R^1_{i,j}$ denote the rectangle defined by grid points $(i, j)$ and $(p_{x_{max}}, p_{y_{min}})$. We denote the square containing maximum number of points within $R^1_{i,j}$ by $S^{opt}_{i,j}$ and the number of points within it by $C^{opt}_{i,j}$.

In this phase we compute $C_{i,j}$, $S^{opt}_{i,j}$ and $C^{opt}_{i,j}$ for each grid point $(i, j)$. Here we need to consider only those grid points $(i, j)$ such that $S_{i,j}$ lies properly within the rectangle defined by $(p_{x_{min}}, p_{y_{min}})$ and $(p_{x_{max}}, p_{y_{max}})$.

**Lemma 4** *$C_{i,j}$, $S^{opt}_{i,j}$ and $C^{opt}_{i,j}$ can be computed for all grid points $(i, j)$ in $O(n^2)$ time.*

**Proof:** For a particular $j$, we can compute $S_{i,j}$ and the corresponding $C_{i,j}$ for all $i$ by using two vertical unit sticks and traversing the horizontal grid line corresponding to this $j$ from right to left. This takes linear time. To compute $C^{opt}_{i,j}$, we note that $C^{opt}_{i,j} = \max(C_{i,j}, C^{opt}_{i,j-1}, C^{opt}_{i+1,j})$ and $S^{opt}_{i,j}$ is the square for which maximum occurs. The Lemma follows since we have to consider all $j$. □

### 4.3 Phase II

From the notion of stair case defined above we observe the following.

**Observation 1** *For any two points $p_i$ and $p_j$ in some square, if $p_{x_i} < p_{x_j}$ and $p_{y_i} < p_{y_j}$, the stair case defined by these two points is unique.*

Hence, from the above observation, we can conclude that, the number of stair cases is at most $O(n^2)$.

**Observation 2** *Let $S_{i,j}$ be a square and its corresponding stair case be $AB$. Then the stair case $CD$ for the square $S_{i-1,j}$ can be obtained from $AB$ by adding at most one point to the left of $AB$ and deleting zero or more points from its right.*

**Observation 3** *Let the points $p_{q_1}, p_{q_2}, \ldots, p_{q_{m-1}}, p_{q_m}$ form a stair case. Then the optimum count $C_{V_{q_1,q_m}}$ is equal to $\max(C_{V_{q_1,q_r}}, C_{V_{q_r,q_m}})$ for $1 < r < m$ and the corresponding square is $S_{V_{q_1,q_m}}$.*

Consider the stair cases defined by pairs of points $(p_i, p_j)$ for all possible $i$ and $j$. Let $M_{ij}$ of the matrix $M$ store the optimum square $S_{V_{i,j}}$ and the corresponding count $C_{V_{i,j}}$ for the stair case defined by the pair $(p_i, p_j)$. Note that no stair case is defined for an entry $M_{ij}$ for which $p_{x_i} < p_{x_j}$ and $p_{y_i} > p_{y_j}$.

**Lemma 5** *The matrix $M$ can be computed in $O(n^2)$ time.*

### 4.4 Phase III

In this phase, we compute the candidate pair of squares and the optimum pair using the results of Phase-I and Phase-II.

We use two vertical unit sticks $\mathcal{L}$ and $\mathcal{R}$ whose bottom points are initialized at the right most grid point of a horizontal grid line corresponding to $j$. Let this grid point be $(n, j)$. Let the $n$-th entry in $L_x$ be the point $p_k$ $(\in P)$. The stick $\mathcal{L}$ is moved towards left, one grid point at a time, till it reaches the grid point $(left(p_k)_x, p_{y_j})$. The first point found is taken as the right most point of the stair case. Subsequent points are collected satisfying Equation 1. The points thus collected forms the stair case within the square defined by the current positions of the unit sticks $\mathcal{L}$ and $\mathcal{R}$. We insert them in a list in the order as they are found. Let the list of points be $V_{q_1,q_m} = \{p_{q_1}, p_{q_2}, \ldots, p_{q_m}\}$. From Phase-II, $C_{V_{q_1,q_m}} = M_{q_1,q_m}$ and the corresponding square is $S_{V_{q_1,q_m}}$. Additionally compute two grid points $\alpha$ and $\beta$, where $\alpha$ is on the bottom part of the stair case and $\beta$ is on its top part as explained previously. If they are present, we lookup the corresponding $S_\alpha$, $S_\beta$ and $C_\alpha$, $C_\beta$ from the result of Phase-I. Let $C_V = \max(C_{V_{q_1,q_m}}, C_\alpha, C_\beta)$ and $S_V$ be the corresponding square. The optimal square $S_V$ together with the square defined by the sticks $\mathcal{L}$ and $\mathcal{R}$ is a candidate pair containing total number of points, say $C$, from $P$.

Now move the square implicitly towards left by one grid point and update the stair case within it by moving $\mathcal{L}$ and $\mathcal{R}$ towards left. Then find the candidate pair in the similar way as above and update $C$ and the corresponding pair when current $C$ is less than the value of $C$ in the previous step. This is done for all grid points on the grid line corresponding to $j$. Finally, we repeat this process for all grid lines. We thus have the result.

**Theorem 6** *Given a set $P$ of $n$ points in $\mathbb{R}^2$, two disjoint or intersecting (with empty overlapping zone) isothetic unit squares covering maximum number of points can be found using $O(n^2)$ time and space.*

### References


[1] J. M. Diaz-Banez, C. Seara, J. A. Sellares, J. Urrutia, and I. Ventura, Covering Points Sets with Two Convex Objects, *Proceedings of the twenty first EWCG*, 2005.

[2] D. T. Lee and Y. F. Wu, Geometric complexity of some location problems, *Algorithmica*, 1, 193-211, 1986.

[3] M. Mukherjee and K. Chakraborty, A polynomial-time optimization algorithm for a rectlinear partitioning problem with applications in VLSI design automation, *Information Processing Letters*, 83, 41-48, 2002.

[4] M. de Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf, Computational Geometry, Algorithms and Applications, *Springer*, 1997.