A Study on Planar 2-Center Problem

A Dissertation submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Technology in Computer Science

by

SHINCE K BABY

[Roll no: CS2322]

Supervisor: Dr. Sandip Das

Professor

Advanced Computing and Microelectronics Unit

Indian Statistical Institute

Kolkata-700108, India

June, 2025

CERTIFICATE

This is to certify that the dissertation titled "**A Study on Planar 2-Center problem**" submitted by **Shince K Baby** to the Indian Statistical Institute, Kolkata in partial fulfillment of the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has met all the requirements as per the regulation of this institute and, in my opinion, has reached the standard needed for submission.

Sandip Das 11/6/2025

Dr. Sandip Das

Professor,

Advanced Computing and Microelectronics Unit,

Indian Statistical Institute,

Kolkata-700108, India.

ACKNOWLEDGEMENT

I want to sincerely thank my supervisor, Prof. Dr. Sandip Das, for allowing me to work under him, for correcting my proofs when I was mistaken, and for offering me insightful advice when I needed it. I wish I had had more time to enjoy his valuable company.
Additionally, I want to thank my good friends Sai Srujan and Dhruv for spending time with me to relax when I worked too much. I would always cherish those moments of friendship.
Finally, I want to thank my mom and dad for their prayers and blessing. I could never ask anything more from them.

Shince K Baby
Indian Statistical Institute
Kolkata-700108, India

ABSTRACT

The Planar-k-Centre problem is an important problem in the class of Optimal Facility Location problems, where given n points in the planar, the objective is finding the smallest k congruent discs such that their union encompasses all points. This dissertation examines a variant of this problem in which the L^1 metric is used to determine the distances rather than the standard Euclidean metric, which we call L^1P2C and a closely related problem which we call Undirected k Square Coverage where there is no directional constraint for the k squares which contain the given set of points.

We have found two deterministic algorithms for the L^1P2C problem, one operating in $O(n \log n)$ time and the other in $O(n)$ time. Additionally, for $k = 1$, we have found an $O(n \log n)$ time method for the Undirected k Square Coverage problem.

Contents

Chapter 1

Introduction to Planar-2-Center Problem

In computational geometry, researchers have been trying to group points into regions or clusters for several decades. Many approaches for such grouping have been researched, and one of them was to group them into circular regions. The problem of Planar$-k-$Center was widely studied in this regard.

Given a set P of points in $\mathbb{R}^d$, the $k-$center problem is to find k congruent hyperspheres in $\mathbb{R}^d$ of the smallest radius such that all the points of P lie within any of the k hyperspheres. The $k-$center problem in $\mathbb{R}^d$ is known to be NP-complete if $d \geq 2$. [19]. In dimension $d = 2$, the problem is called the Planar-k-Center problem which we shall abbreviate as PkC for our purposes. PkC is known to be NP-hard and the best known algorithm for the problem takes $n^{O(\sqrt{k})}$ time [16].

For $k = 1$, the problem simply translates to finding the smallest circle that contains the given set of points. This could be solved deterministically in $O(n)$ time [10], although the actual algorithm is difficult to implement. It turns out that for $k = 1$ this problem could be solved in linear time for any dimension [20].

For $k = 2$, the problem becomes to find two smallest circles of equal radius that contain all given points. This problem is significantly more difficult than P1C. In 1997, Eppstein [11] found that P2C requires at least $\Omega(n \log n)$ time. A long-standing open problem was to find an $O(n \log n)$ time algorithm to match the lower bound by Eppstein.

There were several algorithms and results on the P2C problem over the years. The

first published work on P2C came out in 1991 by Hershberger and Suri [14] which gave an $O(n^2 \log n)$ time algorithm for the decision version of the problem. An improved version which ran in $O(n^2)$ time was published by Hershberger[13] in 1993.
Using Herberger's 1993 result and a technique called parametric search, Agarwal and Sharir[3] gave an $O(n^2 \log^3 n)$ time algorithm for the original problem. After a series of small improvements from Agarwal and Sharir a significant milestone was reached when Eppstein [11] gave a randomized algorithm that ran in $O(n \log^2 n)$ expected time and Chan [6] gave an $O(n \log^2 n \log \log n)$ time deterministic algorithm for the problem in 1999. This was the best result for more than two decades until in 2022 Wang [22] gave a deterministic algorithm that ran in $O(n \log^2 n)$ time. Then two years later in 2024 Cho et al.[8, 7] gave a deterministic algorithm that runs in $O(n \log n)$ finally solving the long-standing open problem of matching the lower bound by Eppstein once and for all.
Several variations of the P2C problem were also studied. Like the *discrete 2-center problem* where the goal is to find 2 circles that contain the given set of points P and their centers are a pair of points in P [4, 23]. The *colored k-center problem* is where each point in the point set P is given a color from $\{1, 2, 3, \ldots, k\}$ and the aim is to find k minimum radius circles, each containing at least one point of each color [1, 15, 24]. The constrained Planar-$k-$center problem is to find $k-$circles whose union covers all points and their centers lie on a given line L. The work of Brass et al [5] and A. Karmakar et al [17] are notable in this regard. The interested author should look into 2-centre problem in higher dimensions too [2]. Containment of points with other structures are also studied [18].
In this thesis we are going to investigate a lesser known variant of the problem which is *Planar-2-Center Problem in* L^1 *domain*. More about this will be given in the next chapter.

Chapter 2

Planar$-2-$Center in L^1 Domain

2.1 What is Planar-2-Center in L^1 Domain

We have already seen that in the entire premise of the P2C (Planar-2-Center) problem is that we have taken euclidean norm, also called the L^2 norm, to calculate distances in the plane. We are also covering all the points using two circles, i.e. structures which have constant L^2 distance with respect to a center point. What if we work with the L^1 norm? How does the problem change?
A structure that has a constant distance from a center point with respect to the L^1-norm would be as in Figure 2.1. As you can see, this is a square that is rotated 45° with an axis in the center. Immediately we can see that a simpler variation of the original PkC problem is to take circles in the L_1 norm. Observe that if we rotate the plane by 45° then this particular version of the PkC can be written as:
Given a set of n points in the x-y plane find k congruent axis aligned squares of the smallest side length that cover all the points.
We shall call this problem as Planar-k-Center problem in L^1 Domain or L^1PkC.

Remark. It is easy to see that the L^1P1C problem could be solved simply by taking the axis-aligned bounding rectangle and then enclosing that rectangle in an axis-aligned square with side length equal to the longer side of the rectangle. This could be solved easily in $O(n)$ time.

The subsequent sections of this chapter discusses the approach for k=2. We shall

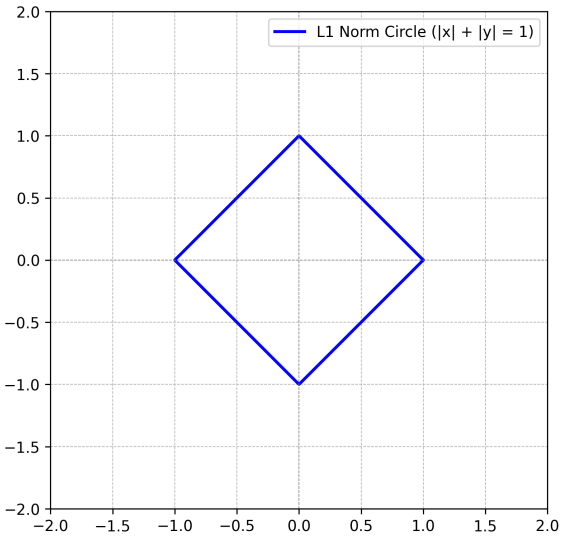


Figure 2.1: L^1 norm circle with radius 1

discuss 2 different algorithms for solving this problem, one runs in $O(n \log n)$ time and the other in $O(n)$ time.

2.2 Solving L^1P2C

We have found 2 algorithms that solves this problem efficiently. For both the algorithms the first order of business is to find out the axis aligned bounding rectangle for the given n points. This can be done in $O(n)$ time. Without loss in generality assume that the length of the bounding box parallel to x-axis is longer than the y-axis parallel side. If this is not the case we can simply rotate the plane by 90° to get this. We denote the longer side length as l and the shorter side length as b.

Definition 2.2.1. We shall call the points that touch the 'width' sides as length-bounding points, and the points that touch the 'length' sides as breadth-bounding points.

It is evident that two squares that encompass all n points will be located to the left of the bounding box's right-width side. This is so because there cannot be any points in the area outside the bounding box if any square extends to the right of the right-width side of the box. As a result, we can move the square's edge to the right-

width side. This also applies to the bounding box's left-width side. This reasoning can be distilled into the following observation.

Observation 2.2.1. For any n-points, there exist solutions of the L^1P2C problem such that both squares lie in between the vertical lines subtended by the 'breadth' sides of the axis aligned bounding rectangle.

We also have the following observation:

Observation 2.2.2. For any $n-$points, there exist solutions of the L^1P2C problem such that both squares have a corner located at diagonally opposite corners of the axis-aligned bounding rectangle.

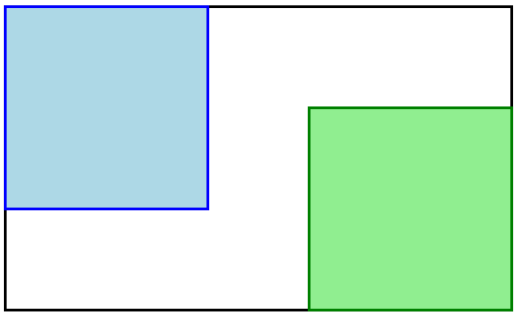
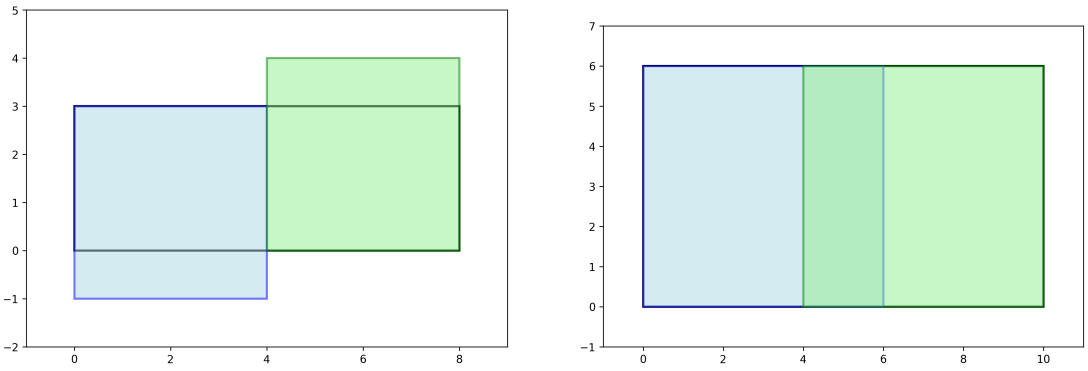


Figure 2.2: Solution to L^1P2C looks something like this - above figure drawn for illustrative purposes

The lemma below is arguably the cornerstone of both the algorithms we are to discuss.

Lemma 2.2.1. For any instance of the L^1P2C problem, the length-bounding points on opposite sides cannot be on the same square.

Proof. On the contrary, if, for some instance of L^1P2C, the length bounding points were on the same square, then the length of that square will be greater than or equal to l. But for the case where the bounding rectangle has dimensions l and b such that $l/2 \geq b$, we have a covering as shown in figure 2.3a. Also for the case where the bounding rectangle has dimensions l and b such that $l/2 < b$ we have a covering as



(a) $l/2 \geq b$ (b) $l/2 < b$

Figure 2.3: Two congruent squares that will cover the bounding box in each of the cases

shown in figure 2.3b.

By the above figures, the solution of any L^1P2C problem will have solutions whose squares' side-lengths are less than or equal to $\max\{l/2, b\}$, which is strictly less than l. This proves that the length bounding points cannot be on the same square. □

With all the above knowledge, let's now look towards the methodology of the first algorithm.

2.3 Contracting Squares Algorithm

Starting with a known set of squares that will cover every point, the objective behind this method is to compress the squares until the ideal value is reached. A detailed methodology of this approach is given in the following subsection.

2.3.1 Contracting Squares Methodology

First of all, draw the axis-aligned bounding rectangle for the given set of points. Both are known to lie between the breadth-sides of the bounding rectangle by observation 2.2.1. We do not know at which two diagonal corners does the solution squares share their corners. So we shall run our algorithm on both the diagonal corner pairs and choose whichever returns smaller squares.

For description we shall choose main-diagonal corners of the bounding rectangle.

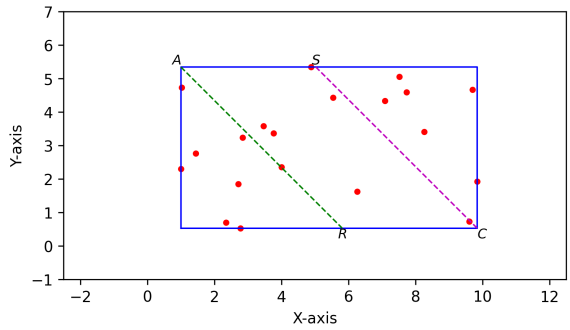


Figure 2.4: Case 2; Step 1

There are two possible cases to consider.

Case 1: $l/2 \geq b$: Here the solution squares, if they intersect, would do so only at its boundary. This is because as in figure 2.3a even the extreme case of coverage of such a rectangle would not have their individual square side lengths greater than $l/2$. Also, if the covering squares were any smaller, they would not even touch each other's boundaries.

So in this case it suffices to draw a perpendicular bisector line to the length side and determine the bounding squares of the points on either side of the bisector line. This, as we have seen in an earlier remark, could be done in $O(n)$ time.

Case 2: $l/2 < b$: Here the solution squares may intersect. Let the chosen diagonal corners of the bounding rectangle be A and C. Draw line segments AR and CS such that R and S lie on the opposite length edge of A and C, respectively, and both line segments subtend 45° to their corresponding breadth edges. (See fig. 2.4). We shall assume that the bounding rectangle of points is longer along the x-direction. Define the following two sets

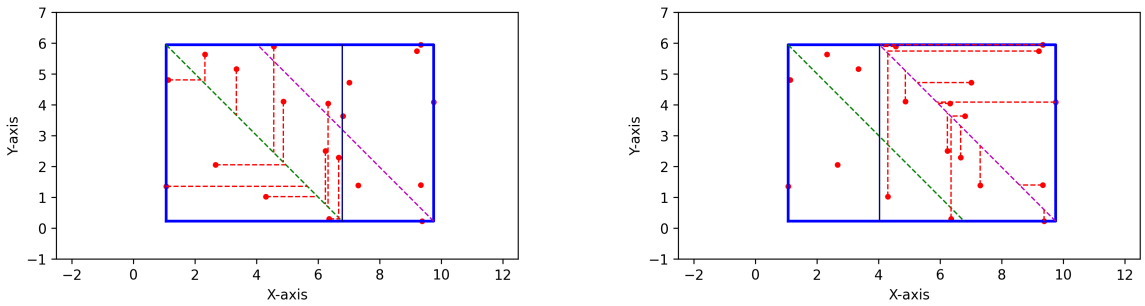
$$P_1 = \{p : p \in P, p_x \leq R_x, \text{where } p = (p_x, p_y) \text{ and } R = (R_x, R_y)\}$$
$$P_2 = \{p : p \in P, p_x \geq S_x, \text{where } p = (p_x, p_y) \text{ and } S = (S_x, S_y)\}$$

For every point draw horizontal projections onto AR if $p \in P_1$ lies to the left of AR, and project vertically onto AR if p lies to the right of AR. We shall refer to these projected points as Square Projections of p onto AR.

Likewise, for every point $p \in P_2$: If p is to the right of CS, project it horizontally

onto CS; if not, project it vertically onto CS. We shall refer to these projected points as Square Projections of p onto CS.

We shall denote it by the square projection function$\Gamma(p, AR)$ and $\Gamma(p, CS)$. See figure 2.5. We will then derive the following sets from P_1 and P_2.



(a) Square projection of P_1 on AR (b) Square projection of P_2 on CS

Figure 2.5: Case 2, Step 2

$$P_1^* = \{\Gamma(p, AR) : p \in P_1\}$$
$$P_2^* = \{\Gamma(p, CS) : p \in P_2\}$$

For all the projected points in P_1^* and P_2^* define

$$M(q) = \begin{cases} d(q, A) & \text{, if } q \in P_1^* \\ d(q, C) & \text{, if } q \in P_2^* \end{cases}$$

where $d()$ denotes the L^2 norm function. Sort the points of $P^* = P_1^* \cup P_2^*$ according to their respective $M(q)$ values in decreasing order.

Now for each projected point $q \in P^*$ draw squares of diagonal length $M(q)$ that is cornered at A and C and check if the point in P that corresponds to the previous projected point in P^* is contained within these two squares. If yes, continue to the next point. If not, then the two squares that correspond to the previous projected point are returned.

A complete pseudocode for this methodology is given below.

Algorithm 1: Contracting Squares Algorithm

Input: A set P of n points in the $x - y$ plane
Output: Two congruent axis aligned squares of the smallest side length that contains P.

```
1 W ← BoundingRect(P) ;
  /* W is expected to have side aligned along x-axis to be
     longer.  If this not the case rotate the frame by 90°, get
     results and rotate the results back by 90°                  */
2 A, B, C, D = Corners of W in cyclic order;
3 S_A, S_C, l_AC ← MinSquaresContractive(P, BR, A, C);
4 S_B, S_D, l_BD ← MinSquaresContractive(P, BR, B, D);
5 if l_AC < l_BD then
6 |   return S_A, S_C, l_AC;
7 else
8 |   return S_B, S_D, l_BD;
```

2.3.2 Proof of Correctness

The proof of correctness follows from theorem 2.3.1. Considering the fact that both squares have a fixed corner at some corner of the axis-aligned bounding rectangle, we shall give the following definitions.

Definition 2.3.1. Let S be an axis square that has one of its corners fixed at some point $\mathcal{O}$. A point p is said to be critically covered by S w.r.t $\mathcal{O}$ if $p \in S$ and any smaller axis aligned square which shares a corner at $\mathcal{O}$ will not contain p. See figure 2.6.

Theorem 2.3.1. Given 2 diagonal corners X, Z of the axis-aligned bounding rectangle of a given set of points P and the two axis-aligned squares, with fixed corners at X and Z respectively, having the smallest side length that covers all points in P, there exists a point in P such that it is either critically covered by both squares or is critically covered by one square and uncovered by the other square.

Proof. Let S_X, S_Z be the two required squares with their respective corners fixed at X and Z. If both have the smallest possible side length, there exists at least one point that is critically covered by one of the squares S_X, S_Z. This is because if no

Algorithm 2: MinSquaresContractive(P, W, X, Z)

Input: A set of points P, its axis aligned bounding rectangle W (assumed to have longer side parallel to x-axis), two diagonally opposite corners X, Z of W

Output: Two squares S_X, S_Z that share corners with W at X, Z respectively and has properties as in Observations 2.2.1 and 2.2.2 and their side-length l_{XZ}

$P_1^*, P_2^*, Point\_Map \leftarrow \varnothing, \varnothing, \varnothing$;
$X_x, X_y \leftarrow$ x,y coordinates of X respectively;
$Z_x, Z_y \leftarrow$ x,y coordinates of Z respectively;
$R = (R_x, R_y) \leftarrow (X_x + b, Z_y)$;
$S = (S_x, S_y) \leftarrow (Z_x - b, X_y)$;
for $p \in P$ **do**
    $p_x, p_y \leftarrow$ x,y coordinates of p respectively;
    if $p_x \leq R_x$ **then**
        $q \leftarrow$ Square Projection of p on XR;
        P_1^*. append(q);
        insert into $Point\_Map$ set the mapping $q \mapsto (p, M(q))$;
    if $p_x \geq S_x$ **then**
        $q \leftarrow$ Square Projection of p on ZS;
        P_2^*. append(q);
        insert into $Point\_Map$ set the mapping $q \mapsto (p, M(q))$;
$P^* \leftarrow \text{sort}(P_1^* \cup P_2^*,\ \text{key} = \textbf{lambda}\ q : M(q), \text{reverse} = \text{True})$;
$q\_prev \leftarrow P^*.pop()$;
for $q \in P^*$ **do**
    $p, l \leftarrow Point\_Map(q\_prev)$;
    $S_X \leftarrow$ Square cornered at X with diagonal along line XR and diagonal length $M(q)$;
    $S_Z \leftarrow$ Square cornered at Z with diagonal along line ZS and diagonal length $M(q)$;
    if *p is contained by S_X or S_Z* **then**
        $q\_prev \leftarrow q$;
    else
        $S_X \leftarrow$ Square cornered at X with diagonal along line XR and diagonal length l;
        $S_Z \leftarrow$ Square cornered at Z with diagonal along line Zq and diagonal length l;
        return S_X, S_Z, l;

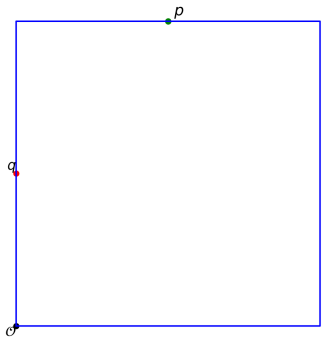


Figure 2.6: This axis-aligned square with fixed corner at $\mathcal{O}$; critically covers p but not q.

such point exists then we could have reduced the size of both squares by some small amount(keeping corners X, Z fixed) and still could cover all points of P, thereby making S_X, S_Z not the smallest squares as assumed.

Let D be the collection of all points $p \in P$ such that at least one of the squares critically covers p. Assume, contrary to the statement of the theorem, all of them are critically covered by one square and non-critically covered by the other.

Assume that $p' \in D$ is critically covered by S_X without loss of generality. By assumption p' is non-critically covered by S_Z. Let S_X^{nc} be the non-critically covered points of P in S_X. Then there exists a length $\varepsilon_{p'} > 0$ by which we could reduce the size of both squares such that p' is covered by S_Z and all points of S_X^{nc} is covered by S_X. By the assumption, such a value must exist for all the points of D. So we let

$$\varepsilon = \min\{\varepsilon_p : p \in D\}$$

Now if we reduce the size of both squares by ε, then the union of the squares will cover all points of D and all points of S_X^{nc} and S_Z^{nc}. This implies that the squares cover all the points of P even when their sizes are reduced by ε. This contradicts the assumption that all points of D were critically covered by one square and non-critically covered by the other.

Hence at least one point of D was critically covered by both squares or critically covered by one square and uncovered by the other. This proves the theorem. □

Following the result of this theorem, we can deduce that if the squares, which have

fixed corners at X, Z, had their size reduced to smaller than their optimal length, then both squares will not cover a critically covered point when the squares were at their optimal side length. So we could keep reducing the lengths of the squares from a previously acceptable length and then check at each juncture whether the previous critically covered point is currently uncovered or not. This is what our algorithm does. This establishes the correctness of our algorithm.

2.4 Expanding Squares Algorithm

Here the idea is to start from two small squares and if the initial set of squares does not contain all points, we expand them until they do. A detailed methodology is given in the following subsection.

2.4.1 Expanding Squares Methodology

Draw the axis-aligned bounding rectangle for the given set of points. Just like in the Contracting squares case, we do not know at which two diagonal corners do the solution squares share their corners. So, as before, we shall run our algorithm on both pairs of diagonal corners and return the smaller pair of squares.
For describing the algorithm, we shall choose the off-diagonal corners of the rectangle. The algorithm proceeds as follows.
Draw two axis-aligned squares of side length $l/2$ such that one corner of each square coincides with the diagonal corners selected before. We shall refer to these two squares as *'Check Squares'*. See figure 2.7 for reference.

Definition 2.4.1. We shall define the region of the bounding rectangle that lies outside the area of the two check squares as the *white region* corresponding to the two selected corners.

Check if all input points are within these two squares. If so, find out the minimum bounding squares for both sets of points that lie within each of the squares. The side length of the larger square between the two generated squares is the required answer and we are done. If not, then that means that there are points in the 'white region'

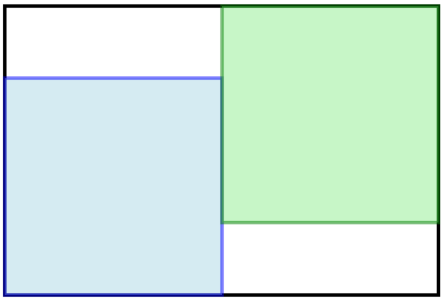


Figure 2.7: Check Squares for the given bounding rectangle w.r.t. its off-diagonal corners

of the rectangle and we shall continue to the next step of the algorithm. Draw line segments Q_1J and Q_2K as shown in figure 2.8 Now for all points in the region DNQ_1J

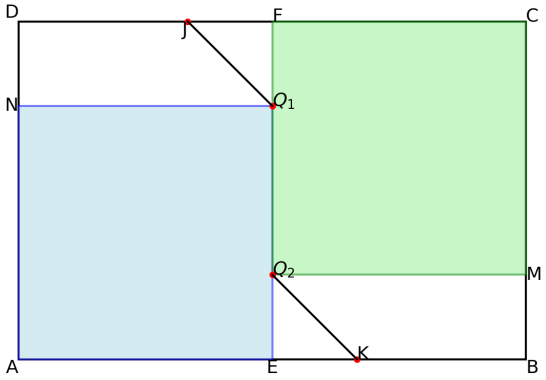
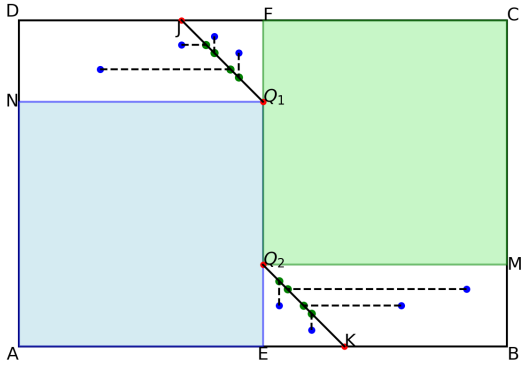


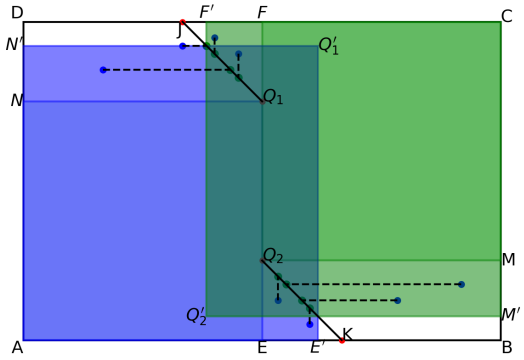
Figure 2.8: Step 2 of Expanding Squares algorithm.

draw a horizontal projection onto the the line Q_1J. Also draw vertical projections of points in region FQ_1J on to the line Q_1J. Similarly draw horizontal projections of points in region BMQ_2K on to the line Q_2K and vertical projections of points in region EQ_2K onto Q_2K. See Figure 2.9a for an illustration. The green points indicates the projected points. For each point p that lies in the 'white region', we denote its corresponding projection point as p^*. This leads us to define a function $H(p)$ as

$$H(p) = \begin{cases} d(p^*, Q_1N) & \text{,if } p \text{ in region } DNQ_1F \\ d(p^*, Q_2M) & \text{,if } p \text{ in region } BMQ_2E \end{cases}$$



(a) Step 3 of expanding squares algorithm.



(b) Step 4 of expanding squares algorithm.

Figure 2.9

where $d(p^*, L)$ denotes perpendicular distance of p^* from line L. Among all the points p outside the two squares let $H_{max} = \max\{H(p) : p \text{ in white region}\}$. Expand both squares' length by H_{max} keeping corners A and C fixed as in figure 2.9b.

The result will be the squares $AN'Q_1'E'$ and $CM'Q_2'F'$ and their side length is $l/2 + H_{max}$. A complete pseudocode of this approach is given below:

Algorithm 3: Expanding Squares Algorithm

Input: A set P of n points in the $x - y$ plane
Output: Two congruent axis aligned squares of the smallest side length

```
1 BR ← BoundingRect(P) ;
  /* BR is expected to have side aligned along x-axis to be
     longer.  If this not the case rotate the frame by 90°, get
     results and rotate the results back by 90°                  */
2 A, B, C, D = Corners of BRin cyclic order;
3 S_A, S_C, l_AC ← MinSquaresExpansive(P, BR, A, C);
4 S_B, S_D, l_BD ← MinSquaresExpansive(P, BR, B, D);
5 if l_AC < l_BD then
6 |   return S_A, S_C, l_AC;
7 else
8 |   return S_B, S_D, l_BD;
```

Algorithm 4: MinSquaresExpansive(P, BR, X, Z)

Input: A set of points P, its axis aligned bounding rectangle BR (assumed to have longer side parallel to x-axis), two diagonally opposite corners X, Z of BR

Output: Two squares S_X, S_Z that share corners with BR at X, Z respectively and has properties as in Observations 2.2.1 and 2.2.2 and their side-length l_{XZ}

$P_1, P_2 \leftarrow \varnothing, \varnothing$;
$X_x, X_y \leftarrow$ x,y coordinates of X respectively;
$Z_x, Z_y \leftarrow$ x,y coordinates of Z respectively;
$l, b, x_{mid} \leftarrow Z_x - X_x, |Z_y - X_y|, (X_x + Z_x)/2$;
for $p \in P$ **do**
    $p_x, p_y \leftarrow$ x,y coordinates of p respectively;
    if $p_x < x_{mid}$ **then**
        if $|p_y - X_y| < l/2$ **then**
            P_1.append(p);
            remove p from P;
    else if $|p_y - Z_y| < l/2$ **then**
        P_2.append(p);
        remove p from P;
if $P == \varnothing$ **then**
    $S_X \leftarrow$ BoundingSq(P_1);
    $S_Z \leftarrow$ BoundingSq(P_2);
    $l_{XZ} =$ maximum of sidelength of S_X and S_Z ;
// Find coordinates of Q_1, Q_2 as in figure 2.8
$Q_1 \leftarrow (x_{mid}, X_y + sign(Z_y - X_y) \cdot l/2)$;
$Q_2 \leftarrow (x_{mid}, Z_y - sign(Z_y - X_y) \cdot l/2)$;
$J, K \leftarrow (x_{mid} - |(Z_y - Q_1^y)|, Z_y), (x_{mid} + |(X_y - Q_2^y)|, X_y)$;
// Computing p^* for all remaining p and getting maximum $H(p)$
$H_{max} \leftarrow 0$;
for $p \in P$ **do**
    $p_x, p_y \leftarrow$ x,y coordinates of p respectively;
    if $p_x < x_{mid}$ **then**
        if $|Q_1^x - p_x| > |Q_1^y - p_y|$ **then**
            project p onto line Q_1J horizontally to get its corresponding p^*;
        else
            project p onto line Q_1J vertically to get its corresponding p^*;
    else
        if $|Q_2^x - p_x| > |Q_2^y - p_y|$ **then**
            project p onto line Q_2K horizontally to get its corresponding p^*;
        else
            project p onto line Q_2K horizontally to get its corresponding p^*;
    $H_{max} = \max\{H_{max}, H(p)\}$;
Draw two axis aligned squares S_X, S_Z of sidelength $l/2 + H_{max} = l_{XZ}$ inside BR which shares corners X and Z with BR;
return S_X, S_Z, l_{XZ};

Theorem 2.4.1. The Expanding Squares Algorithm will correctly generate an optimal solution to L^1P2C problem in $O(n)$ run time and $O(n)$ space complexity.

Proof. Assume that the side length of the squares produced by the aforementioned Expansive Squares method is t^* and that there is a solution to an instance of the L^1P2C issue whose square side length is r^*. We know by observation 2.2.2 and 2.2.1 that an optimal solution to any instance of L^1P2C will have corners situated at diagonally opposite corners of its bounding rectangle and will lie in between the two breadth sides. So, without loss in generality assume that the solutions are cornered at the off diagonal corners A and C of the bounding rectangle. If $r^* \leq l/2$, then there is no intersection between the two squares and there will be no points in the corresponding white region. This case is covered in lines 14-17 of the MinSquaresExpansive() subroutine. The optimal squares will be the bounding squares of the two well-separated point regions. Thus, $t^* = r^*$ in this case.
On the other hand, if $r^* > l/2$ then there is an intersection between the two squares and there are points in the white region. Let $r^* = l/2 + h$. We must have that $h \leq H_{max}$ (H_{max} as previously defined) as the squares of side length r^* are assumed to be optimal. In the point set P, let $\tilde{p}$ be a point such that $H(\tilde{p}) = H_{max}$. If $h < H_{max}$, then the assumed optimal squares will not contain $\tilde{p}^*$. Because the points in the white region are projected onto the projection line segments based on the most efficient coverage of points by the expanding check squares, if a projection point on the projection lines is not covered by any squares, it will also not cover its corresponding original point. Thus, the squares, as it does not contain $\tilde{p}^*$, will also not contain $\tilde{p}$ which gives a contradiction since the optimal squares are supposed to cover all points. This implies $h = H_{max}$ and thus $t^* = r^*$ which proves optimality of the covering squares of the Expanding Squares Algorithm.
While $T(n)$ is the time complexity of the MinSquaresExpansive() subroutine, which is called twice in this method, the algorithm's run time complexity is $2T(n)$. Because it consists of two loops with conditional statements for every point in the set that is simply verified in $O(1)$ time, the MinSquaresExpansive() function executes in $O(n)$ time. The algorithm's time complexity is therefore $2T(n) = 2O(n) = O(n)$.

Total memory used in MinSquaresExpansive() is $|P_1| + |P_2| + |P| + c = O(n)$. Since extra space is only used by MinSquaresExpansive() we get that the space complexity is $O(n)$. □

2.5 Concluding Remark

Even though we have given algorithms for covering points with axis-aligned squares, we can see that this algorithm could also be used for any specific orientation. The easiest thing to do is rotate the coordinate axes to align with the specific direction of orientation to run the algorithm as usual and rotate the results back.

Chapter 3

Undirected $k-$Square Coverage

In the previous chapter, we have seen two algorithms to determine the optimal solution to the L^1P2C problem by constructing the required two, smallest possible, axis-aligned squares. This leads us to ask the following question. If we change the question to covering the set of points using simply two congruent squares, will the solution change? If so, how do we find the optimal solution?
It is intuitive to see that the answer to the first question is yes. Of course, L^1P2C had the additional restriction of a specific directional orientation for the squares. Considering that there is no orientation specified, the minimum possible side length for the covering squares must be smaller.
We will generalize and formalize our question into something called the Undirected $k-$Square coverage problem (abbreviated as the "UkSC Problem.") which are of two different types as defined below.

i. **Aligned UkSC**: All the given points must be covered by k squares with the same side length so that every square points in the same direction. i.e., all sides of one square must be either parallel or perpendicular to each side of the other squares.

ii. **Unaligned UkSC**: k squares of the same side length must cover all given points. No directional requirement for any square.

We investigate UkSC for $k = 1$ where both aligned and unaligned cases are one and the same problem. So we shall call this problem the Minimum Enclosing Square

problem.

3.1 Minimum Enclosing Square Problem

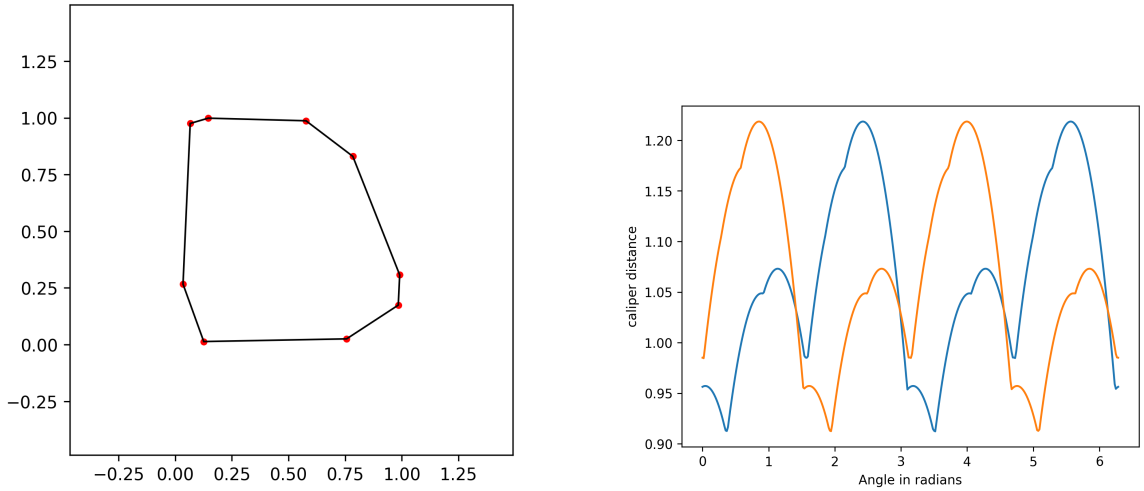
A set P of n points in $\mathbb{R}^2$ is presented to us. The smallest square that can encompass every point in P is what we are looking for. The smallest square that would encompass m points of P may be found using a $O(n^2 \log n + mn(n-m)^2 \log n)$ time method, according to a study by S. Das et al.[9]. An $O(n^2 \log n)$ time method that determines the least enclosing square in all orientations is obtained when we set $m = n$. We would like to do better that this. Inspiration for the rest of the work has been drawn from [12].

Recall that if we are given a specific direction $\hat{\mathbf{v}}$ of orientation, it is easy to construct the enclosed square following the remark from section 2.5. Therefore, in order to construct the smallest square that contains P, we may simply find the smallest longer side length among all longer side lengths of all the possible bounding rectangles of P. i.e., we need to find

$$l_{\min} = \min \{l : l \text{ is the longer side of the bounding rectangle } W(P, \hat{\mathbf{v}}), \hat{\mathbf{v}} \in \mathbb{R}^2\}$$

The smallest bounding square will be the bounding square oriented along some vector $\mathbf{v}$ where longer side length of $W(P, \mathbf{v})$ is $l_{\min}$. Now, it remains to see how we will find the minimum of that set above.

All bounding rectangles $W(P, \hat{\mathbf{v}})$, $\hat{\mathbf{v}} \in \mathbb{R}^2$ (and bounding squares) is a convex shape, and any convex shape that contains any set of points must also contain its convex hull. A bounding rectangle of a convex hull of points could be found by simply having two sets of parallel lines that are perpendicular to each other rotating around the convex hull in a rotating caliper fashion[21]. One set of caliper lines along the direction of $\hat{\mathbf{v}}$ and the other set of caliper lines along the direction $\hat{\mathbf{v}}^{\perp}$ both containing the hull in between them. Consider the illustrations below. Each of the caliper distances will repeat in periods of π as the caliper lines at angle θ is the same as the caliper lines at $\pi + \theta$. So, it suffices to check for the smallest greater distance for 2 calipers that run from angles 0 to π and $\pi/2$ to $3\pi/2$. The caliper that rotates from 0 to π shall be



(a) Convex Hull of a sample P

(b) Two sets of Caliper distances

Figure 3.1

named caliper-1 and its perpendicular counterpart will be called caliper-2. Here in figure 3.1b the blue graph is the distance between the caliper-1 and the orange line is the distance between caliper-2. The points with non- differentiability on each graph of 3.1b are explained by the changing antipodal points[21] as the calipers touch the edges of the hull.

Definition 3.1.1. Given any caliper on a convex hull, the distance between the supporting antipodal points is called *antipodal length* and the line segment connecting the two antipodal points is called the antipodal line corresponding to the caliper or simply *antipodal line* when the caliper lines are understood.

A calliper and its corresponding antipodal points p and q in the hull can be used to calculate the distance between the calliper line using the formula $l_{pq} \sin \varphi$, where φ is the angle between the calliper and the antipodal line segment $\overline{pq}$ and l_{pq} is the antipodal distance corresponding to the calliper. See Figure 3.2 for an illustration. We need two perpendicular caliper lines that track their respective antipodal points as they rotate.

Now, to find out the distance between caliper lines at any instance, it suffices to know

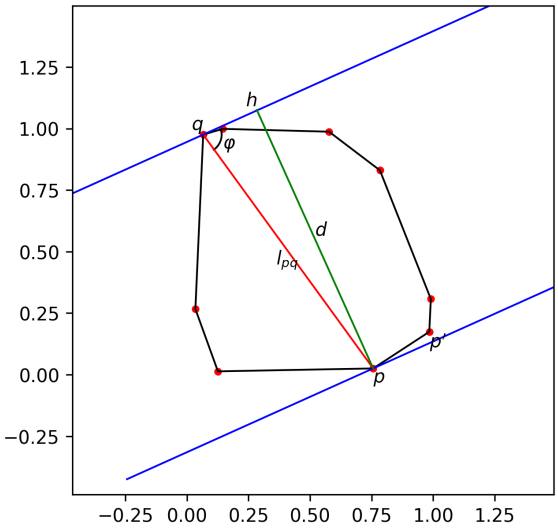


Figure 3.2: Distance between the caliper lines in blue is $d = l_{pq} \sin \varphi$

the antipodal distance and the angle between the antipodal line and the caliper at each point of non-differentiability, i.e. at angles where one of the caliper lines goes through any of the edges of the hull. Similarly, to find out the greater distance among the two sets of caliper lines, it suffices to know the antipodal distance and the angle between antipodal line and caliper lines at some event points.
There are two sets of event points:

1. Hull-edge events: points at which there is a non-differentiability in the graph of caliper distance of any of the two calipers.

2. Intersection events: Angles at which the caliper distance graphs of both sets of caliper lines coincide.

Each event point is represented as an object which has the following attributes: The pair of caliper lines it corresponds to, the point in the graph (angle(θ),caliper_distance(d)), the corresponding antipodal length, and the angle between the caliper line and the antipodal line. It is given below in the respective order.

```
Object Event_Node{
    int caliper_number;
    float point [2];
```

```
    float antipodal_length;
    float cal_podal_angle;
}
```

We shall use a plane sweep algorithm in the angle-caliper distance graph to find the intersection events. A vertical sweep line will be swept along the angle axis from left to right in the graph. We shall maintain an array of just two elements called sweep line status that stores the pair of caliper lines which has the highest and lowest distance at any given angle θ in the angle axis. We shall also store the event node corresponding to the higher and lower caliper distances along with the sweep line status array. We shall also maintain an event queue data structure in the form of a sorted array of available events sorted along the angle axis.
Observe that when a plane sweep is performed, the status of the sweep line changes, in terms of which caliper pair has the higher distance between them, just before and after an intersection event. The change in status caused by an intersection event continues until the next event point. The intersection event node must contain the caliper number and antipodal length of the lower event curve. Its "cal_podal_angle" will be the cal_podal_angle of the preceding event curve + angular distance between previous event point and intersection point. This is because the higher curve changes after an intersection. The intersection point can be found out as follows.
Say an event curve of the caliper-1 intersects with an event curve of the caliper-2. An intersection occurs between the event curves only when their respective caliper distances are equal. Let the caliper distances of the two sets of caliper lines be given by

$$d_1 = l_1 \sin(\varphi_1 + \alpha) \text{ for the caliper line pair 1}$$
$$d_2 = l_2 \sin(\varphi_2 + \alpha) \text{ for the caliper line pair 2}$$

At intersection event angle we must have $d_1 = d_2$. This implies

$$l_1 \sin(\varphi_1 + \alpha) = l_2 \sin(\varphi_2 + \alpha)$$
$$\implies \frac{l_1}{l_2} \sin(\varphi_1 + \alpha) = \sin(\varphi_2 + \alpha)$$

On expanding we get:

$$\begin{aligned}
&\frac{l_1}{l_2}(\sin\varphi_1\cos\alpha+\cos\varphi_1\sin\alpha)=(\sin\varphi_2\cos\alpha+\cos\varphi_2\sin\alpha)\\
\implies &\left(\frac{l_1}{l_2}\sin\varphi_1-\sin\varphi_2\right)\cos\alpha=\left(\cos\varphi_2-\frac{l_1}{l_2}\cos\varphi_1\right)\sin\alpha\\
\implies &\frac{\frac{l_1}{l_2}\sin\varphi_1-\sin\varphi_2}{\cos\varphi_2-\frac{l_1}{l_2}\cos\varphi_1}=\tan\alpha\\
\implies &\alpha=\arctan\left(\frac{\frac{l_1}{l_2}\sin\varphi_1-\sin\varphi_2}{\cos\varphi_2-\frac{l_1}{l_2}\cos\varphi_1}\right)
\end{aligned}$$

Lemma 3.1.1. There can be at most one intersection event in between any two event curves.

Proof. If there is an intersection between two event curves, then we must have

$$\alpha=\arctan\left(\frac{\frac{l_1}{l_2}\sin\varphi_1-\sin\varphi_2}{\cos\varphi_2-\frac{l_1}{l_2}\cos\varphi_1}\right)$$

where l_1 and l_2 are the antipodal distances of the respective event curves, φ_1 and φ_2 are the angle between the respective antipodal line and caliper line, and α is the angular turn by which both calipers must be turned from the preceding hull-edge event to reach the intersection event. Both $\varphi_1+\alpha$ and $\varphi_2+\alpha$ must be between 0 and π, as the angle between any caliper line and its respective antipodal line cannot exceed π radians. So, we have $0\leq\alpha<\min\{\pi-\varphi_1,\pi-\varphi_2\}<\pi$. In between 0 and π there is only one value of α that will satisfy the above equation within this interval since arctan is a bijective function from $\mathbb{R}$ to $[0,\pi)$. This proves the lemma. □

The following corollary follows immediately from this.

Corollary 3.1.1. There can be at most $2h-1$ intersection events, where $h=$ number of hull edges = number of hull-edge events.

Proof. Only when the sweep line's state changes does an intersection between two event curves take place. Every caliper distance curve has h hull-edge events. This implies that the sweep line status could change for a maximum of $2h-1$ times. This is similar to how in a stack of $2n$ coins with 2 different denominations and n coins of

each denomination there could only be a maximum of $2n - 1$ instances where coins of different denominations touch each other.
Given that there could only be at most one intersection between two event curves, this proves that there could only be at most $2h - 1$ intersection events. □

A pseudocode to find the intersection event is given in the GetIntersectionEvents() method below.

Algorithm 4: GetIntersectionEvents(HullEvents)

Input: The Hull-Events in sorted order
Output: The set of all Intersection events and Higher events

```
1  Higher_events = [];
2  Intersections = [];
3  Status_line[2] ← initial status at θ = 0;
4  for h ∈ HullEvents do
5      φ ← h.point[0];
6      α ← arctan( (l1/l2 sin φ1 − sin φ2) / (cos φ2 − l1/l2 cos φ1) );
7      if α < φ − θ then
8          new event_node 'e' created;
9          e.caliper_number = Status_line[1].caliper_number;
10         e.antipodal_length = l = Status_line[1].antipodal_length;
11         φ = Status_line[1].cal_podal_angle;
12         e.point = (θ + α, l sin(φ + α));
13         e.cal_podal_angle = φ + α;
14         Change status_line status;
15         Higher_events.append(e);
16         Intersections.append(e);
17     Higher_events.append(Status_line[0]);
18 return Intersections, Higher_events;
```

We can find the higher curve in the caliper distance graph by keeping track of the event curves at the top at every event point. The caliper distance at any point on an event curve is given by $d = l \sin(\varphi + \theta)$ where $0 \leq \theta \leq \theta_0 \leq \pi - \varphi$, θ_0 is the angular distance to the next event point and φ is the angle between caliper line and antipodal line at event point. The smallest value of d at any event curve as θ varies from 0 to

Algorithm 5: MinSquare(E)

Input: The set E of all the event points of the higher caliper distance graph
Output: A square of the smallest possible side length that covers the hull

```
1  d_min ← ∞;
2  for e ∈ E do
3  |   d ← e.point[1];
4  |   if d < d_min then
5  |   |   d_min ← d;
6  |   |   e_min ← e;
7  θ ← e.point[0];
8  v ← (cos θ, sin θ);
9  S_min ← BoundingSquare(P, v);
10 return S_min;
```

θ_0 is given by

$$d_{\min} = \begin{cases} l\sin(\varphi + \theta_0), & \text{if} \quad \theta_0 > \pi - 2\varphi \\ l\sin(\varphi), & \text{if} \quad 0 \leq \theta_0 \leq \pi - 2\varphi \end{cases}$$

This shows that the smallest value of d at any event curve in the higher curve graph is at one of it's endpoints. This shows that we could simply find the higher caliper distance at each of the event points and get the event point with the smallest higher caliper distance. Return the bounding square of the hull corresponding to that event point.

A pseudocode for this given in the MinSquare() subroutine.

Lemma 3.1.2. Minimum Square Coverage could be done in $O(n \log n)$ time.

Proof. Finding the hull events, intersection events, and applying the MinSquare subroutine to the Higher_events event points is all that is required. The convex hull of the provided n points could be found in $O(n \log n)$ time in order to determine the hull events. We can locate the hull events of both callipers in their angular sequence and combine them in $O(n)$ time by using a rotating caliper approach. MinSquare will execute in $O(2n) = O(n)$ time, whereas obtaining the intersection events requires $O(2n-1) = O(n)$ time. Therefore, in $O(n \log n + n) = O(n \log n)$ time, the minimal bounding square is obtained overall. □

Chapter 4

Conclusion and Future work

Two distinct algorithms for the L^1P2C problem and an algorithm to determine the minimal enclosing square of a given set of points have been presented in this dissertation. It is surprising to see that a seemingly simple problem like the Minimum Enclosing Square would need such intricate details to figure out an efficient algorithm. To an interested reader, I would humbly request to find a simpler method for the efficient computation of Minimum Enclosing Square.

The problem of undirected square coverage is very similar to the PkC problem, especially the Unaligned USC. A further work on this manuscript would be in finding an efficient algorithm for Aligned UkSC problem for $k = 2$. Solving it could provide substantial insight into the solution of the Unaligned UkSC problem for $k = 2$.

The pivotal idea that helped solve the P2C problem in $O(n \log n)$ time was an efficient data structure to maintain what is called called circular hulls [22]. Such kind of structures and the data structures that could maintain them, might be helpful in solving the Unaligned UkSC problem. Such directions could be investigated.

Bibliography

[1] M. Abellanas et al. "The farthest color Voronoi diagram and related problems". In: *17th European Workshop on Computational Geometry*. 2001.

[2] Pankaj K. Agarwal, Rinat Ben Avraham, and Micha Sharir. "The 2-center problem in three dimensions". In: *Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry*. SoCG '10. Association for Computing Machinery, 2010, pp. 87–96. ISBN: 9781450300162.

[3] Pankaj K. Agarwal and Micha Sharir. "Planar geometric location problems". In: *Algorithmica* 11.2 (1994), pp. 185–195.

[4] Pankaj K. Agarwal, Micha Sharir, and Emo Welzl. "The discrete 2-center problem". In: *Discrete and Computational Geometry* 20.3 (1998), pp. 287–305.

[5] Peter Brass et al. *Computing k-centers on a line*. Tech. rep. 0902.3282. arXiv, 2009. arXiv: 0902.3282 [cs.CG].

[6] Timothy M. Chan. "More planar two-center algorithms". In: *Computational Geometry: Theory and Applications* 13.3 (1999), pp. 189–198.

[7] Kyungjin Cho et al. "Optimal Algorithm for the Planar Two-Center Problem". In: *40th International Symposium on Computational Geometry (SoCG 2024)*. Vol. 293. Leibniz International Proceedings in Informatics (LIPIcs). 2024, 40:1–40:15. ISBN: 978-3-95977-316-4.

[8] Kyungjin Cho et al. "Optimal Algorithm for the Planar Two-Center Problem". In: *TheoretiCS* Volume 3 (2024). ISSN: 2751-4838. DOI: 10.46298/theoretics.24.23. URL: http://dx.doi.org/10.46298/theoretics.24.23.

[9] Sandip Das, Partha P. Goswami, and Subhas C. Nandy. "Smallest k-point enclosing rectangle and square of arbitrary orientation". In: *Information Processing Letters* 94.6 (2005), pp. 259–266. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2005.02.013. URL: https://www.sciencedirect.com/science/article/pii/S0020019005000621.

[10] M. E. Dyer. "On a multidimensional search technique and its application to the Euclidean one centre problem". In: *SIAM Journal on Computing* 15.3 (1986), pp. 725–738.

[11] David Eppstein. "Faster construction of planar two-centers". In: *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (1997), pp. 131–138.

[12] H. Freeman and R. Shapira. "Determining the minimum-area encasing rectangle for an arbitrary closed curve". In: *Communications of the ACM* 18.7 (July 1975), pp. 409–413.

[13] John Hershberger. "A faster algorithm for the two-center decision problem". In: *Information Processing Letters* 1 (1993), pp. 23–29.

[14] John Hershberger and Subhash Suri. "Finding tailored partitions". In: *Journal of Algorithms* 12.3 (1991), pp. 431–463.

[15] D. P. Huttenlocher, K. Kedem, and M. Sharir. "The upper envelope of Voronoi surfaces and its applications". In: *Discrete & Computational Geometry* 9.3 (1993), pp. 267–291.

[16] R. Z. Hwang, R. C. T. Lee, and R. C. Chang. "The slab dividing approach to solve the Euclidean p-center problem". In: *Algorithmica* 9.1 (1993), pp. 1–22.

[17] A. Karmakar et al. "Some variations on constrained minimum enclosing circle problem". In: *Journal of Combinatorial Optimization* 25.2 (2013), pp. 176–190.

[18] P.R.S. Mahapatra, P.P. Goswami, and S. Das. "Maximal covering by two isothetic unit squares". In: *Proceedings of the 20th Annual Canadian Conference on Computational Geometry, CCCG 2008* (2008), pp. 103–106.

[19] Spaccamela Marchetti. "The p-center problem in the plane is NP-complete". In: *Proc. 19th Allerton Conf. on Communication, Control and Computing* (1981), pp. 31–40.

[20] Nimrod Megiddo. "Linear programming in linear time when the dimension is fixed". In: *Journal of the ACM* 31.1 (1984), pp. 114–127.

[21] Godfried Toussaint. "Solving geometric problems with the rotating calipers". In: *Proceedings of the IEEE*. Athens, Greece, 1983.

[22] Haitao Wang. "On the planar two-center problem and circular hulls". In: *Discrete and Computational Geometry* 68.4 (2022), pp. 1175–1226.

[23] Haitao Wang. "Unit-disk range searching and applications". In: *Journal of Computational Geometry* 14 (2023), pp. 343–394.

[24] Haitao Wang and Jie Xue. "Improved algorithms for the bichromatic two-center problem for pairs of points". In: *Computational Geometry: Theory and Applications* 100 (2022), pp. 1–12.