Constrained Square-Center Problems*

Matthew J. Katz[1], Klara Kedem[1,2] and Michael Segal[1]

[1]Department of Mathematics and Computer Science
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel

[2]Computer Science Department
Upson Hall, Cornell University, Ithaca, NY 14853

October 28, 1999

Abstract

Given a set P of n points in the plane, we seek two squares whose center points belong to P, their union contains P, and the area of the larger square is minimal. We present efficient algorithms for three variants of this problem: In the first the squares are axis parallel, in the second they are free to rotate but must remain parallel to each other, and in the third they are free to rotate independently.

1 Introduction

In this paper we consider the problems of covering a given set P of n points in the plane by two constrained (discrete) squares, under various conditions.

*Work by M. Katz and K. Kedem has been supported by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities. K. Kedem has also been supported by the U.S.-Israeli Binational Science Foundation, and by the Mary Upson Award, College of Engineering, Cornell University.

We call a square *constrained* if its center lies on one of the input points. In particular, we solve the following problems:

1. Find two constrained axis-parallel squares whose union covers P, so as to minimize the size of the larger square. We present an $O(n \log^2 n)$-time algorithm; its space requirement is $O(n \log n)$.

2. Find two constrained parallel squares whose union covers P, so as to minimize the size of the larger square. The squares are allowed to rotate but must remain parallel to each other. Our algorithm runs in $O(n^2 \log^4 n)$ time and uses $O(n^2)$ space.

3. Find two constrained squares whose union covers P, so as to minimize the size of the larger square, where each square is allowed to rotate independently. We present an $O(n^3 \log^2 n)$-time and $O(n^2)$-space algorithm for this problem.

The problems above continue a list of optimization problems that deal with covering a set of points in the plane by two geometric objects of the same type. We mention some of them: The two center problem, solved in time $O(n \log^9 n)$ by Sharir [18], and recently in time $O(n \log^2 n)$ by Eppstein [7] (by a randomized algorithm); the constrained two center problem, solved in time $O(n^{\frac{4}{3}} \log^5 n)$ by Agarwal et al. [2]; the two line-center problem, solved in time $O(n^2 \log^2 n)$ by Jaromczyk and Kowaluk [11] (see also [9, 13]); the two square-center problem, where the squares are with mutually parallel sides (the unconstrained version of Problem 2 above), solved in time $O(n^2)$ by Jaromczyk and Kowaluk [10].

We employ a variety of techniques to solve these optimization problems. The decision algorithm of Problem 1 searches for the centers of a solution pair (of squares) in an implicit special matrix, using a technique that has recently been used in [6, 18]. To find an optimal solution, a search in a collection of sorted matrices [8] is performed.

The decision algorithm of Problem 2 involves maintenance of dynamically changing convex hulls, and maintenance of an orthogonal range search tree that must adapt to a rotating axes system. For the optimization, we apply Megiddo's [14] parametric search. However, since our decision algorithm is not parallelizable, we had to find an algorithm that solves a completely different problem, but is both parallelizable and enables to generate the optimal square size when the parametric search technique is applied to it.

In Problem 3 we describe the sizes of candidate solution squares as a collection of curves. For a dynamically changing set of such curves, we transform the problem of determining whether their upper envelope has a point below some horizontal line, into the problem of stabbing a dynamically changing set of segments. The latter problem is solved using a (dynamic) segment tree.

2 Two constrained axis-parallel squares

We are given a set P of n points in the plane, and wish to find two axis-parallel squares, centered at points of P, whose union covers (contains) P, such that the area of the larger square is minimal. We first transform the corresponding decision problem into a constrained 2-piercing problem, which

we solve in $O(n \log n)$ time. We then apply the algorithm of Frederickson and Johnson [8] to find an optimal solution.

2.1 The decision algorithm

The decision problem is stated as follows: Given a set P of n points, are there two constrained axis-parallel squares, each of a given area $\mathcal{A}$, whose union covers P. We present an $O(n \log n)$ algorithm for solving the decision problem.

We adopt the notation of [20] (see also [12, 17]). Denote by $\mathcal{R}$ the set of axis-parallel squares of area $\mathcal{A}$ centered at the points of P. $\mathcal{R}$ is *p-pierceable* if there exists a set $\mathcal{X}$ of p points which intersects each of the squares in $\mathcal{R}$. The set $\mathcal{X}$ is called a *piercing set* for $\mathcal{R}$. Notice that $\mathcal{X}$ is a piercing set for $\mathcal{R}$ if and only if the union of the axis-parallel squares of area $\mathcal{A}$ centered at the points of $\mathcal{X}$ covers P. $\mathcal{R}$ is *p-constrained pierceable* if there exists a piercing set of p points which is contained in P. Thus, solving the decision problem is equivalent to determining whether $\mathcal{R}$ is 2-constrained pierceable.

We first compute the rectangle $R = \cap\mathcal{R}$. If R is not empty then $\mathcal{R}$ is 1-pierceable, and we check whether it is also 1-constrained pierceable by checking whether P has a point in R. If $\mathcal{R}$ is 1-constrained pierceable then we are done, so assume that it is not. If $\mathcal{R}$ was not found to be 1-pierceable, then we apply the linear time algorithm of [20] (see also [5]) to check whether $\mathcal{R}$ is 2-pierceable. If $\mathcal{R}$ is neither 1-pierceable nor 2-pierceable, then obviously $\mathcal{R}$ is not 2-constrained pierceable and we are done. Assume therefore that $\mathcal{R}$ is 2-pierceable (or 1-pierceable).

Assume $\mathcal{R}$ is 2-constrained pierceable, and let $p_1, p_2 \in P$ be a pair of

piercing points for $\mathcal{R}$. We assume that p_1 lies to the left of and below p_2. (The case where p_1 lies to the left of and above p_2 is treated analogously.) We next show that $\mathcal{R}$ can be divided into two subsets $\mathcal{R}_1, \mathcal{R}_2$, such that (i) $p_1 \in \cap\mathcal{R}_1$, $p_2 \in \cap\mathcal{R}_2$, and (ii) $\mathcal{R}_1$ (alternatively $\mathcal{R}_2$) can be represented in a way that will assist us in the search for p_1 and p_2.

Denote by $X_{\mathcal{R}}$ the centers of the squares in $\mathcal{R}$ (the points in P) sorted by their x-coordinate (left to right), and by $Y_{\mathcal{R}}$ the centers of the squares in $\mathcal{R}$ sorted by their y-coordinate (low to high). We now claim:

Claim 2.1 *If p_1 and p_2 are as above, then $\mathcal{R}$ can be divided into two subsets $\mathcal{R}_1$ and $\mathcal{R}_2$, $p_1 \in \cap\mathcal{R}_1$, $p_2 \in \cap\mathcal{R}_2$, such that $\mathcal{R}_1$ can be represented as the union of two subsets $\mathcal{R}_1^x$ and $\mathcal{R}_1^y$ (not necessarily disjoint, and one of them might be empty), where the centers of squares of $\mathcal{R}_1^x$ form a consecutive subsequence of the list $X_{\mathcal{R}}$, starting from its beginning, and the centers of squares of $\mathcal{R}_1^y$ form a consecutive subsequence of $Y_{\mathcal{R}}$, starting from the list's beginning.*

Proof. We prove by constructing the sets $\mathcal{R}_1^x$ and $\mathcal{R}_1^y$, and then putting $\mathcal{R}_1 = \mathcal{R}_1^x \cup \mathcal{R}_1^y$ and $\mathcal{R}_2 = \mathcal{R} - \mathcal{R}_1$. We next show that indeed $p_1 \in \cap\mathcal{R}_1$ and $p_2 \in \cap\mathcal{R}_2$.

We consider the centers in $Y_{\mathcal{R}}$, one by one, in increasing order, until a center is encountered whose corresponding square A is not pierced by p_1. $\mathcal{R}_1^y$ consists of all squares in $Y_{\mathcal{R}}$ below A (i.e., preceding A in $Y_{\mathcal{R}}$). A might be the first square in $Y_{\mathcal{R}}$, in which case $\mathcal{R}_1^y$ is empty. We now find the location of the x-coordinate of the center of A in $X_{\mathcal{R}}$, and start moving from this point leftwards, i.e., in decreasing order. Thus moving, we either encounter

a square, call it B, that is **higher** than A and is not pierced by p_2, or we do not.

If we do not encounter such a square B (which is clearly the case if the bottom edge of A lies above p_1), then put $\mathcal{R}_1^x = \emptyset$, otherwise $\mathcal{R}_1^x$ consists of all squares in $X_{\mathcal{R}}$ to the left of B including B.

It remains to show that $p_1 \in \cap\mathcal{R}_1$ and that $p_2 \in \cap\mathcal{R}_2$. We assume that the square B exists, which is the slightly more difficult case. We first show the former assertion, i.e., $p_1 \in \cap\mathcal{R}_1$. The fact that p_2 is not in B implies that p_2 lies to the right of the right edge of B, because B cannot lie below p_2 since it is higher than A which is already pierced by p_2. Therefore none of the squares in $\mathcal{R}_1^x$ is pierced by p_2 thus $p_1 \in \cap\mathcal{R}_1^x$. By our construction, $p_1 \in \cap\mathcal{R}_1^y$, so together we have $p_1 \in \cap\mathcal{R}_1$. Now consider a square $C \in \mathcal{R}_2$, $C \neq A$. C is higher than A, because it is not in $\mathcal{R}_1^y$. Therefore if C is not pierced by p_2, then C must lie to the left of A. But if so, it is in $\mathcal{R}_1^x$ and thus not in $\mathcal{R}_2$. ∎

The claim above reveals a monotonicity property that allows us to design an efficient algorithm for the decision problem. We employ a technique, due to Sharir [18], that resembles searching in monotone matrices; for a recent application and refinement of this technique, see [6]. Let M be an $n \times n$ matrix whose rows correspond to $X_{\mathcal{R}}$ and whose columns correspond to $Y_{\mathcal{R}}$. An entry M_{xy} in the matrix is defined as follows. Let D_x be the set of squares in $\mathcal{R}$ such that the x-coordinate of their centers is smaller or equal to x, and let D_y be the set of squares in $\mathcal{R}$ such that the y-coordinate of their centers

is smaller or equal to y. Let $D^l_{xy} = D_x \cup D_y$ and $D^r_{xy} = (\mathcal{R} - D^l_{xy})$.

$$M_{xy} = \begin{cases} `YY' & \text{if both } D^r_{xy} \text{ and } D^l_{xy} \text{ are 1-constrained pierceable} \\ `YN' & \text{if } D^r_{xy} \text{ is 1-constrained pierceable but } D^l_{xy} \text{ is not} \\ `NY' & \text{if } D^r_{xy} \text{ is not 1-constrained pierceable but } D^l_{xy} \text{ is} \\ `NN' & \text{if neither } D^r_{xy} \text{ nor } D^l_{xy} \text{ is 1-constrained pierceable} \end{cases}$$

Sharir's technique enables us to determine whether M contains an entry of the form 'YY' without having to construct the entire matrix. In order to apply his technique the lines and columns of M^1 must be non-decreasing (assuming 'Y' > 'N'), and the lines and columns of M^2 must be non-increasing, where M^i is the matrix obtained from M by picking from each entry only the i'th letter, $i = 1, 2$. In our case this property clearly holds, since, for example, if for some x_0 and y_0, $M^1_{x_0,y_0}$ ='Y', then for any $x' \geq x_0$ and $y' \geq y_0$, $M^1_{x',y'}$='Y'. Thus we can determine whether M contains an entry 'YY' by inspecting only $O(n)$ entries in M, advancing along a *connected* path within M [6]. For each entry along this path, we need to determine whether D^z_{xy} is 1-constrained pierceable, $z \in \{l, r\}$. This can be done easily in $O(\log n)$ time by maintaining dynamically the intersection $\cap D^z_{xy}$, and utilizing a standard orthogonal range searching data structure of size $O(n \log n)$ [4]. Thus in $O(n \log n)$ time we can determine whether M contains a 'YY' entry.

Theorem 2.2 *Given a set P of n input points and area $\mathcal{A}$, one can find two constrained axis-parallel squares of area $\mathcal{A}$ each that cover P in time $O(n \log n)$ using $O(n \log n)$ space.*

We have just found whether a set of equal-sized squares is 2-pierceable by two of their centers. For the optimization, we shrink these squares as much as possible, so that they remain 2-constrained pierceable.

2.2 Optimization

For solving the optimization problem we observe that each L_∞ distance (multiplied by 2 and squared) can be a potential area solution. We can represent all L_∞ distances as in [9] by sorted matrices. We sort all the points of P in x and y directions. Entry (i, j) in the matrix M_1 stores the value $4(x_j - x_i)^2$, where x_i, x_j are the x-coordinates of the points with indices i, j in the sorted x-order, and, similarly, entry (i, j) in the matrix M_2 stores the value $4(y_j - y_i)^2$, where y_i, y_j are the y-coordinates of the points with indices i, j in the sorted y-order. We then apply the Frederickson and Johnson algorithm [8] to M_1 and M_2 and obtain the smallest value in the matrices for which the decision algorithm answers "Yes" and thus obtain the optimal solution. We have shown:

Theorem 2.3 *Given a set P of n input points, one can find two constrained axis-parallel squares that cover all the input points such that the size of the larger square is minimized in $O(n \log^2 n)$ time using $O(n \log n)$ space.*

3 Two constrained parallel squares

In this section we deal with the following problem. Given a set P of n points in the plane, find a pair of parallel constrained squares whose union contains P, so as to minimize the area (equivalently, the side length) of the larger square. The problem where the squares are not constrained was recently solved by Jaromczyk and Kowaluk [10] in $O(n^2)$ time using $O(n^2)$ space.

We first solve the decision problem for squares with a given area $\mathcal{A}$ in time $O(n^2 \log^2 n)$ and $O(n^2)$ space. For the optimization, we present a parallel

version of another algorithm (solving a different problem), to which we apply Megiddo's parametric search [14] to obtain an $O(n^2 \log^4 n)$ time and $O(n^2)$ space optimization algorithm.

3.1 The decision algorithm

For each of the input points, $p_i \in P$, draw an axis-aligned square Q_i of area $\mathcal{A}$, centered at p_i. For each p_i denote by U_i the set of points in P that are not covered by Q_i. If, for some i, there is a constrained axis-aligned square of area $\mathcal{A}$ which covers U_i, then we are done. Otherwise, we rotate the squares $\{Q_i \mid i = 1, \ldots, n\}$ simultaneously about their centers, stopping at certain *rotation events* to check if any of the corresponding U_i's can be covered by a parallel square of area $\mathcal{A}$, and halting when the answer is "yes".

A *rotation event* occurs whenever a point of P enters or leaves a square Q_i, $i = 1 \ldots n$. When a square Q_i rotates by $\frac{\pi}{2}$ from its initial axis-aligned position, every point of P enters and leaves Q_i at most once. Thus, the number of rotation events for Q_i is $O(n)$. For all the points in P we can precompute all the $O(n^2)$ rotation events in $O(n^2)$ time with $O(n^2)$ space. We sort the rotation events according to their corresponding angles.

We compute the initial convex hulls for each U_i, $i = 1, \ldots, n$ (i.e., at orientation $\theta = 0$), and start rotating the squares till we get to the next rotation event. Assume that at the current rotation event a point p_j enters Q_i. (The case where a point p_j leaves Q_i is treated similarly.) The set U_i and its convex hull are updated as p_j leaves U_i, and we check whether there exists a constrained cover of P involving Q_i and another constrained square (that covers U_i).

We explain how this is done for one square Q_i at orientation $\theta = 0$. First we find the tangents of the convex hull of U_i that are parallel to the sides of Q_i. They define a rectangle R which is the bounding box of U_i. If R has a side of length greater than $\sqrt{\mathcal{A}}$, then none of the other $n-1$ constrained squares covers U_i. Otherwise we define a *search region* C which is the locus of all points of L_∞ distance at most $\frac{\sqrt{\mathcal{A}}}{2}$ from all four sides of R, and search for a point of P in C. (Clearly C is a rectangle whose sides are parallel to the sides of Q_i.) We perform orthogonal range searching to determine whether there is a point of P in C. If there exists such a point then the answer to the decision problem is "yes".

Assume we have computed all the rotation events and have $O(n^2)$ rectangular search regions associated with them. (Assume the coordinate system rotates together with the rotating squares $\{Q_i\}$, thus, at any rotation event, the corresponding rectangular search region is parallel to the current axes.) In order to perform orthogonal range search on the rectangular regions we use a dynamic orthogonal range search tree which is updated at certain rotation events as follows.

Denote by L the list of all $O(n^2)$ lines passing through pairs of points in P. Let S consist of all the slopes of lines in L that lie in the range $[0, \pi/2)$, and of all the slopes in the range $[0, \pi/2)$ of lines that are perpendicular to the lines in L. We sort S, obtaining the sorted sequence $\{\alpha_1, \alpha_2, \ldots\}$. We rotate the axes so that the x-axis has slope α_1, and compute an orthogonal range search tree for P with respect to the rotated axes, storing just the labels of the points of P in the tree. For each search region whose side slope is between α_1 and α_2 we perform a usual range search with this tree. Before

considering the next search regions, we rotate the axes some more until the x-axis has slope α_2. Notice that just one pair of points in P has swapped in x or y order in this angle range. We update the range search tree accordingly: Assuming the leaves of the main structure in the range tree are sorted by x-coordinate, and the leaves in the secondary trees are sorted by y-coordinate. If, when moving from α_1 to α_2, the swap occurred in the x-order of the pair of points, then we swap the (labeling of the) points in the main structure and in the secondary structures affected by that swap; if the swap occurred in the y-order, then we swap the labeling in the affected secondary structures. Now we can proceed with the search ranges whose sides have slopes between α_2 and α_3. And so on.

We analyze the time and space required for the decision algorithm. The total number of rotation events is $O(n^2)$. They can be precomputed and sorted in $O(n^2 \log n)$ time with $O(n^2)$ space. Similarly S can be obtained and sorted within the same bounds. Merging the two sets of slopes (rotation events and S) is done in time $O(n^2)$. Initially computing the convex hulls for all sets U_i takes $O(n^2 \log n)$ time with $O(n^2)$ space. Applying the data structure and algorithm of Overmars and van Leeuwen [16], each update of a convex hull takes $O(\log^2 n)$ time, totaling in $O(n^2 \log^2 n)$ time and $O(n^2)$ space for all rotation events. Our range searching algorithm takes $O(\log^2 n)$ time per query and per update, after spending $O(n \log n)$ preprocessing time and using $O(n \log n)$ space (notice that this is the total space requirement for the range searching), and we perform $O(n^2)$ queries and updates. Thus we have shown:

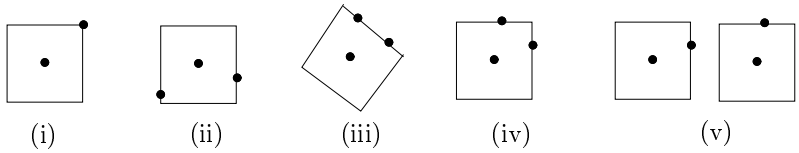

Figure 1: Critical events that determine candidate square sizes. Cases (i) – (iv) involve a single square, and case (v) two squares.

Theorem 3.1 *Given a set P of n points and an area $\mathcal{A}$, one can decide whether P can be covered by two constrained parallel squares, each of area $\mathcal{A}$, in $O(n^2 \log^2 n)$ time and $O(n^2)$ space.*

3.2 Optimization

Having provided a solution to the decision problem, we now return to the minimization problem. The number of candidate square sizes is $O(n^4)$ (see below and Figure 1). The candidate sizes are determined by either

- A point of P as a center of a square (see Figure 1(i)–(iv)) and either (i) another point of P on a corner of this square, or (ii) two points of P on parallel sides of the square, or (iii) two points of P on one side of the square, or (iv) two points of P on adjacent sides of the square, or
- Two points of P as centers of two squares and another point of P on the boundary of each of the squares (Figure 1(v)).

In order to apply the Megiddo optimization scheme we have to parallelize our decision algorithm. However, the range searching part of the decision algorithm is not parallelizable, so, as in [1], we come up with an auxiliary problem whose parallel version will generate the optimal solution to our problem.

The auxiliary problem is described as follows. Assume we have a set P of $n > 2$ points and a fixed size d. Assume we have produced the set of strips such that each strip is of width d and contains at least one point of P on each of its boundaries. In this situation a point on one boundary might stand for the square center and the point on the other boundary is the one on the side of the square. Maintain the set of strips by storing their slopes and the corresponding pairs of points that define them in S. Let $\bar{S}$ be the set of *slopes* obtained by the slopes of S by adding $\pi/2 \pmod{\pi}$. With each slope in $\bar{S}$ we store the pair of points associated with the corresponding slope in S.

A slope $\bar{s} \in \bar{S}$ stands for a pair of square sides perpendicular to the ones defined by its corresponding slope $s \in S$. So that if two perpendicular slopes, s_1 and s_2 (in S) define a square (as in Figure 1 (i),(iv) and (v)), then s_1 and $\bar{s_2}$ are equal. The set of squares thus defined is a superset of the candidate solution squares as defined above. Let $\mathcal{S} = S \cup \bar{S}$ be a set of slopes with their associated point pairs. The **auxiliary problem** is to sort the slopes in $\mathcal{S}$.

Clearly not all pairs of points in P define strips, and thus slopes, in $\mathcal{S}$. A pair of points in P whose distance is smaller than d will not generate the required width strip. For every pair of points in P whose distance from each other is larger than d, there are exactly two slopes for which the width of the strip, with a point of this pair on each of its boundaries, is d. We add these slopes (and their $\bar{S}$ corresponding slopes) to $\mathcal{S}$. Reporting the sorted order of $\mathcal{S}$ can be done in $O(n^2 \log n)$ time, and a parallel algorithm with $O(n^2)$ processors will sort the list in $O(\log n)$ time [3].

We now want to (generically) apply this parallel sort algorithm for finding

the optimal square size d^*. For this we first augment our algorithm, as in [1], and get an initial interval where d^* resides. We perform a preliminary stage that disposes of the cases in which the width of the strip is exactly the distance between two points of P, and those in which the width is the distance between two points multiplied by $\sqrt{2}/2$. We call these distances *special distances*. We can afford to list all these $O(n^2)$ strip widths, sort them, and perform a binary search for d^* over them, applying our decision algorithm of the previous subsection at each of the comparisons. This results in an initial closed interval of real numbers, I_0, that contains the optimal square size d^*, and none of the just computed special sizes is contained in its interior.

Consider now a single step in the parallel sort (the auxiliary problem). In this step we perform $O(n^2)$ slope comparisons, each comparison involving two pairs of points. There are two cases: (a) the two compared slopes are from S (or both are in $\bar{S}$), and (b) one slope is in S and the other in $\bar{S}$. Let one such comparison involve the pairs (p_1, p_2) and (p_3, p_4). In order to resolve this comparison, we must compute for the point pair (p_1, p_2) the slopes of the two strips of width d^* that have p_1 on one boundary of the strip and p_2 on the other. Similarly, we compute the slopes of the two strips of width d^* through (p_3, p_4). Then we sort the four strips by their slopes. Of course, we do not know d^*, so we compute the (at most two) *critical values* d where the sorted order of the four strips changes, namely, for case (a) above, where the two strips are parallel, and for case (b), when the two strips are perpendicular to each other. We do this for all $O(n^2)$ critical value comparisons. Now we apply the decision algorithm of the subsection above to perform a binary

search over the $O(n^2)$ critical values that were computed. Thus we find an interval $I \subseteq I_0$ where d^* resides, resolve all the comparisons of this parallel stage, and proceed to the next parallel stage.

What does resolving mean here? See Figure 2 which depicts case (a). If the comparison was made for two pairs of points (p_1, p_2) and (p_3, p_4) then, if the distance between a pair of points, $d_1 = (p_1, p_2)$ or $d_2 = (p_3, p_4)$, is smaller than the smaller endpoint of the current interval I then this pair will not have a strip of width d^* and it is omitted from the rest of the sort. If the distance is larger than the smaller endpoint of I then the slope ordering of the four strips at d^* is uniquely determined as follows. In Figure 2(a) the strips s_1 and s_2 are parallel at some width d', and in Figure 2(b) we plot the strips of width d^* for the two pairs of points. In Figure 2(c) we graph d as a function of $\theta \in [0, \pi)$ for the two pairs of points. The graph of $d = d_1 \cos(\theta - \theta_1)$ achieves its maximum at (θ_1, d_1), and similarly the graph of $d = d_2 \cos(\theta - \theta_2)$ achieves its maximum at (θ_2, d_2), where θ_1 (θ_2) is the angle that the line perpendicular to the line through (p_1, p_2) $((p_3, p_4))$ makes with the positive x-axis. It is easy to see that for every d each pair of points has two strips and that the two functions intersect at two points. We split the domain of definition of each function to two parts, one in which the function strictly increases and one in which it strictly decreases. In Figure 2(a) and 2(b) the strip s_1 corresponds to the decreasing half of the function in Figure 2(c) and s_3 to the increasing half. Similarly with the strips of (p_3, p_4), s_2 corresponds to the increasing half and s_4 to the decreasing half. Thus the order of the strips at d^* is the order in which the line $d = d^*$ intersects their functions, and the width values at the intersection points of the two functions

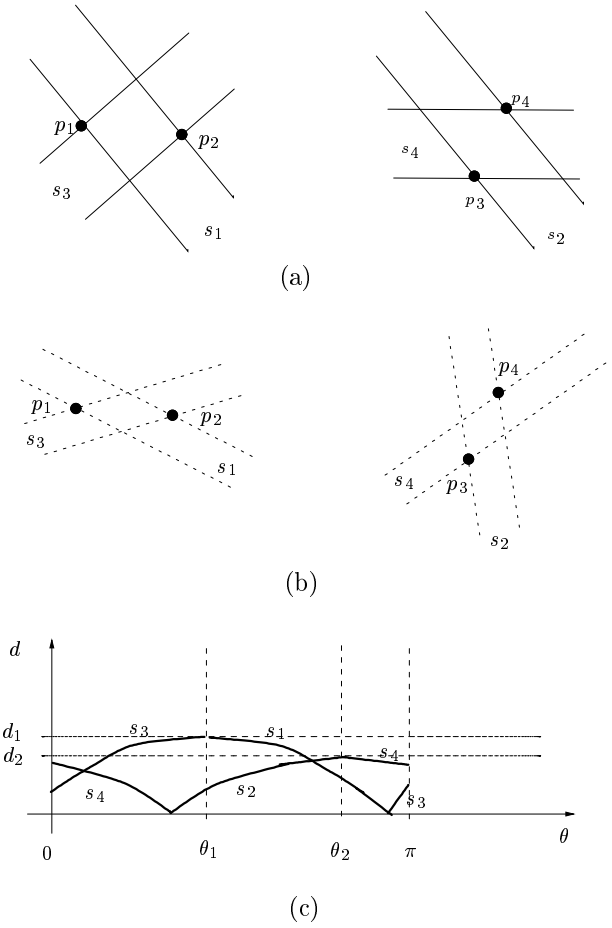

Figure 2: Slope ordering for the comparison of (p_1, p_2) and (p_3, p_4): (a) strips s_1 and s_2 are parallel for some d, (b) the ordering of the slopes at d^*, (c) d as a function of θ

consist of the critical values for these two pairs of points.

For case (b) assume the pair (p_1, p_2) belongs to a strip of $\bar{S}$. We simply cyclically shift the function of (p_1, p_2) (of S) by $\pi/2$. The intersection points of the functions are now at two values of d' where the two strips are perpendicular to each other, and all the rest of the argument is analogous.
Note: We have to be a little more careful here about the notion of the domain of definition of the functions, and we might want to break the domain of definition of the functions also at $\theta = 0$. This is a slight formality that we neglect since it does not change anything in the analysis.

The closed interval I is always guaranteed to contain d^* but we need to show that a comparison is made where $d = d^*$.

Claim 3.2 *If d^* is not one of the special distances then the slope order of the strips changes as d changes from values slightly smaller than d^* to values slightly larger than d^*.*

Proof. Observe again Figure 1. Clearly if d^* is not one of the special distances then it involves two pairs of points. In Figure 1(ii), (iii), (iv), the pairs are the center point of the square paired with each of the two points on the boundary of this square, and in Figure 1(v) the pairs are the center point of each square paired with the point on the side of its square. None of these cases represents a special distance, and hence the slopes of the strips are monotone functions of their widths. These two monotone functions intersect at d^* thus in a small neighborhood of d^* one function is above the other for $d < d^*$ and below for $d > d^*$. ∎

Note that at some stage the optimal solution will appear on the boundary of the interval I computed at that stage (it could even appear on the boundary of I_0). However, once it appears, it will remain one of the endpoints of all subsequently computed intervals. At the end, we run the decision algorithm for the left endpoint of the final interval. If the answer is positive, then this endpoint is d^*, otherwise d^* is the right endpoint of the final interval.

Theorem 3.3 *Let P be a set of n points, we can find a pair of parallel constrained squares whose union covers P and such that the area of the larger square is minimized in $O(n^2 \log^4 n)$ time and $O(n^2)$ space.*

4 Two constrained general squares

In this section the squares may rotate independently. We first state a subproblem whose solution is used as a subroutine in the full solution. Then we present an algorithm for solving the decision problem. This algorithm is used to perform a binary search over the sorted set of potential solutions, producing the solution to the optimization problem.

The subproblem: Given a set P of n points in the plane and a point q, find the minimum area square that is centered at q and that covers P. The square may rotate.

The algorithm for solving the subproblem is as follows. Assume q is the origin. Let θ be an angle in $[0, \frac{\pi}{2})$. Consider the projections, $x_i(\theta)$ and $y_i(\theta)$, of a point $p_i \in P$ on the x-axis and y-axis, after rotating the axes by θ. If the distance between p_i and q is d_i, and the angle between the vector p_i and

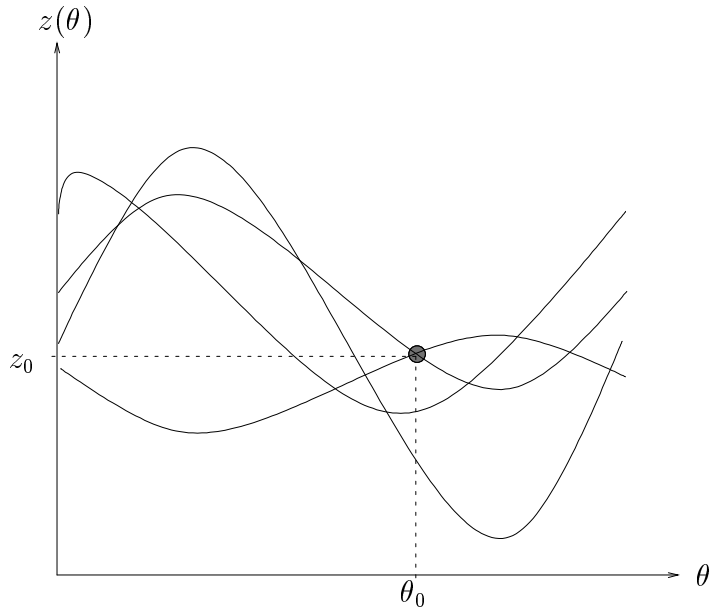

Figure 3: The functions z_i and the lowest point (θ_0, z_0) on their upper envelope

the x-axis at its initial position is θ_i, then we have

$$x_i(\theta) = d_i \cos(\theta_i - \theta) \text{ and } y_i(\theta) = d_i \sin(\theta_i - \theta) \ .$$

A square centered at q rotated by angle θ that has p_i on its boundary is of side length $2 \times \max\{|x_i(\theta)|, |y_i(\theta)|\}$. Note that it is enough to rotate the axes by angle $\theta, 0 \leq \theta < \frac{\pi}{2}$, in order to get all possible sizes of squares centered at q having p_i on their boundary.

Observe the plane (θ, z), on which we graph both $z_{2i-1}(\theta) = 2|x_i(\theta)|$ and $z_{2i}(\theta) = 2|y_i(\theta)|$, $i = 1, \ldots n$. We call the set of these $2n$ functions E_q, and depict them in Figure 3. It is easy to see that every pair of functions z_j and z_k intersects at most twice. The upper envelope of the functions in E_q

denotes, for each θ, the size $z(\theta)$ of the smallest square (centered at q and rotated by θ) that covers P, and the point (or two points) of P corresponding to the function (or two functions) that attains (attain) the maximum at this θ is the point (are the two points) of P on the boundary of the square. The lowest point on this envelope gives the angle, the size, and the point(s) that determine the minimal square. The upper envelope, and the lowest point on it, can be computed in $O(n \log n)$ time [19], and this is the runtime of the solution of the subproblem above.

For the two squares decision problem we repeat some notations and ideas from the previous section. Let Q_i be a square of the given area $\mathcal{A}$ centered at $p_i \in P$. We define rotation events for Q_i as the angles at which points of P enter or leave Q_i. Denote by U_i the set of points not covered by Q_i at the current rotation angle. Using the subproblem described above, we find the smallest constrained square that covers U_i, by computing n sets E_j, where E_j is the set of $2|U_i|$ functions associated with the center point p_j.

We describe our algorithm for determining whether one of the constrained centers is some fixed point $p_i \in P$. Then we apply this algorithm for each of the points in P. Initially, at $\theta = 0$, we construct all the sets E_j, so that each set contains only the functions that correspond to the points in the initial U_i. The rotation events for this phase are those caused by a point of P entering or leaving Q_i. At each rotation event we update U_i and all the sets E_j. We then check whether there is a point on the upper envelope of one of the E_j's which is below the line $z = \sqrt{\mathcal{A}}$. If there exists a point (θ_0, z_0), $z_0 \leq \sqrt{\mathcal{A}}$ on the upper envelope of some E_j, then the square Q_i at its current position, and the square Q_j at angle θ_0 are the solution to the decision problem.

Updating the upper envelopes corresponding to the sets E_j turns out to be time consuming, therefore we transform the problem of determining whether one of the upper envelopes has a low enough point to a segment stabbing problem as follows. Observe one set E_j. If we draw a horizontal line at $z = \sqrt{\mathcal{A}}$, then each function curve in E_j is cut into at most three continuous subcurves, of which at most two lie below the line. We project all the subcurves of E_j that are below the line on the θ-axis, obtaining a set of segments. Assume the number of points in U_i is k, then if (and only if) there is a point θ_0 on the θ-axis that is *covered* by $2k$ segments then there is a square of the required size, of orientation θ_0, centered at p_j which covers the points of U_i.

We construct a segment tree T_j [15] with $O(n)$ leaves (for the segments obtained from all potential curves in E_j). Each node in the tree contains, besides the standard segment information, the maximum cover of the node (namely, the largest number of segments that can be stabbed in the range of the node, for details see [15]). The root of the tree contains the *maximum cover* of the whole range $0 \leq \theta < \frac{\pi}{2}$. The size of one tree is $O(n)$ and each update is performed in time $O(\log n)$. Initially, at $\theta = 0$, we insert into T_j the segments corresponding to the curves of the points in U_i, and check whether the maximum cover equals twice the cardinality of U_i. One update to U_i involves at most four segment updates in T_j.

We consider the time and space complexity of the algorithm. For one point p_i as a candidate center, the initial trees T_j are constructed in time $O(n^2 \log n)$, occupying $O(n^2)$ space. There are $O(n)$ rotation events for Q_i, and an update to one T_j is performed in $O(\log n)$ time, totaling $O(n^2 \log n)$

time for all rotation events and all T_j's. The space requirement is $O(n^2)$. Applying the algorithm sequentially for all i in $\{1, \ldots, n\}$ gives $O(n^3 \log n)$ runtime, while the space remains $O(n^2)$.

To find an optimal solution, we perform for each i as above the following. Assume $p_i \in P$ is one of the two centers in the solution. The corresponding square is defined either by another point of P in its corner, or by two points of P on its boundary. So we compute the $O(n^2)$ potential area sizes with p_i as the center. We sort the area sizes and apply binary search to find the smallest area squares that cover P with p_i as one of the centers in the solution. At each of the $O(\log n)$ search steps, we apply the decision algorithm above (just with p_i as one of the centers). We perform this search for all $i \in \{1, \ldots, n\}$. We have shown:

Theorem 4.1 *Given a set P of n input points we can find a pair of general constrained squares whose union covers P and such that the area of the larger square is minimized in $O(n^3 \log^2 n)$ time and $O(n^2)$ space.*

Acknowledgments

We express our thanks to Paul Chew for helpful discussions.

References

[1] P. Agarwal and M. Sharir, "Planar geometric location problems", *Algorithmica* 11 (1994), 185–195.

[2] P. Agarwal, M. Sharir, E. Welzl, "The discrete 2-center problem", *Proc. 13th ACM Symp. on Computational Geometry*, 147–155, 1997.

[3] R. Cole, "Parallel merge sort", *SIAM J. Computing* 17(4) (1988), 770–785.

[4] M. de Berg, M. van Kreveld, M. Overmars and O. Schwartzkopf, *Computational Geometry, Algorithms and Applications*, Springer-Verlag, 1997.

[5] L. Danzer and B. Grünbaum, "Intersection properties of boxes in R^d", *Combinatorica* 2(3) (1982), 237–246.

[6] O. Devillers and M.J. Katz, "Optimal line bipartitions of point sets", *Int. J. Comput. Geom. and Appls*, to appear.

[7] D. Eppstein, "Faster construction of planar two-centers", *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms*, 131–138, 1997.

[8] G.N. Frederickson and D.B. Johnson, "Generalized selection and ranking: sorted matrices", *SIAM J. Computing* 13 (1984), 14–30.

[9] A. Glozman, K. Kedem and G. Shpitalnik, "Efficient solution of the two-line center problem and other geometric problems via sorted matrices", *Proc. 4th Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science (955), 26–37, 1995.

[10] J.W. Jaromczyk and M. Kowaluk, "Orientation independent covering of point sets in R^2 with pairs of rectangles or optimal squares", *Proc. European Workshop on Computational Geometry*, Lecture Notes in Computer Science (871), 71–78, 1996.

[11] J.W. Jaromczyk and M. Kowaluk, "The two-line center problem from a polar view: A new algorithm and data structure", *Proc. 4th Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science (955), 13–25, 1995.

[12] M.J. Katz and F. Nielsen, "On piercing sets of objects", *Proc. 12th ACM Symp. on Computational Geometry*, 113–121, 1996.

[13] M.J. Katz and M. Sharir, "An expander-based approach to geometric optimization", *SIAM J. Computing* 26 (1997), 1384–1408.

[14] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms", *J. ACM* 30 (1983), 852–865.

[15] K. Mehlhorn, *Multi-dimensional Searching and Computational Geometry*, in "Data Structures and Algorithms", vol. 3, Springer-Verlag, 1984.

[16] M.H. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane", *J. Comput. Syst. Sci.* 23 (1981), 166–204.

[17] M. Segal, "On the piercing of axis-parallel rectangles and rings", *Proc. 5th European Symp. on Algorithms*, Lecture Notes in Computer Science (1284), 430–442, 1997.

[18] M. Sharir, "A near-linear algorithm for the planar 2-center problem", *Discrete Comput. Geom.* 18 (1997), 125–134.

[19] M. Sharir and P. Agarwal, *Davenport-Shintzel sequences and their applications*, Cambridge University Press, New-York, 1995.

[20] M. Sharir and E. Welzl, "Rectilinear and polygonal p-piercing and p-center problems", *Proc. 12th ACM Symp. on Computational Geometry*, 122–132, 1996.