Rectilinear Static and Dynamic Discrete 2-center Problems

Sergei Bespamyatnikh[1] and Michael Segal[2]

[1] University of British Columbia, Vancouver, B.C. Canada V6T 1Z4,
besp@cs.ubc.ca, http://www.cs.ubc.ca/spider/besp
[2] Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
segal@cs.bgu.ac.il, http://www.cs.bgu.ac.il/~segal

Abstract. In this paper we consider several variants of the discrete 2-center problem. The problem is: Given a set S of n demand points and a set C of m supply points, find two "minimal" axis-parallel squares (or rectangles) centered at the points of C that cover all the points of S. We present efficient solutions for both the static and dynamic versions of the problem (i.e. points of S are allowed to be inserted or deleted) and also consider the problem in fixed $d, d \geq 3$ dimensional space. For the static version in the plane we give an optimal algorithm.

1 Introduction

In this paper we consider the following problem: Given a set S of n demand points and a set C of m supply points, find two axis-parallel squares (or rectangles) that cover all the points of S and centered at the points of C such that that size of largest square (rectangle) is minimized. The measure of size is an area or perimeter of the square (rectangle). If $C = S$ then the squares (rectangles) called *discrete* or *constrained*.

The problems above continue a list of optimization problems that deal with covering a set of points in the plane by two "minimal" identical geometric objects. We mention some of them: the two center problem, solved in time $O(n \log^9 n)$ by Sharir [21] and recently in time $O(n \log^2 n)$ by Eppstein [10], employing a randomized algorithm; the constrained two center problem, solved in time $O(n^{\frac{4}{3}} \log^5 n)$ by Agarwal et al.[1]; the two line-center problem, solved in time $O(n^2 \log^2 n)$ by Jaromczyk et al.[16] (see also [18, 12]); the two square-center problem, where the squares are with mutually parallel sides solved in time $O(n^2)$ by Jaromczyk et al.[14]. Hershberger and Suri [13] and Glozman et al.[12] considered the problem of covering the set S by two axis-parallel rectangles such that the size of the larger rectangle is minimized. They present an $O(n \log n)$ algorithm for this problem. In [22, 19, 20] several algorithms are presented that deal with a number of squares (rather than 2) that are not constrained. A recent paper of Katz et al. [17] presents three algorithms for various versions of the discrete two square problem. In the first version, the squares are axis parallel ($O(n \log^2 n)$ time algorithm is presented), in the second, the squares are

allowed to rotate, but remain mutually parallel ($O(n^2 \log^4 n)$ time algorithm) and in the third the squares are allowed to rotate independently ($O(n^3 \log^2 n)$ time algorithm).

We present an optimal algorithm for discrete 2-center problem with $O((n + m) \log(n + m))$ running time. The algorithm is quite simple and uses direct method.

For our problem we generalize the definition of constrained objects. We consider a dynamic version when the points of S are allowed to be inserted or deleted. We show that if $m = o(n)$, then we obtain the algorithm with the sublinear (of the number of points in S) time query for a dynamic version of the problem. Our algorithms work for any d-dimensional space. We implicitly use Frederickson and Johnson technique of sorted matrices [11], i.e. we embed this technique into the decision algorithm in order to speed up the running time. This is crucial for the dynamic version of our algorithm, because standard use of this technique may lead to additional factors of $O(n)$, in the case of squares, and $O(n^2)$, in the case of rectangles, to the running time. We obtain an $O(\max(n \log n,\ m \log n(\log n + \log m)))$ runtime algorithm for the squares case and an $O(mn \log m \log n)$ runtime algorithm for the rectangle case. As for the dynamic versions the runtimes for update operations for both algorithms are polylogarithmic in n for any values of m.

This paper is organized as follows. In the next section we present our algorithm for the case of squares and also describe the generalization to the dynamic and high dimensional versions. Section 3 deals with the case of rectangles. We conclude in Section 4.

2 Squares

2.1 Initial approach

We consider the following problem: Given a set S of n demand points and set C of m supply points in d-dimensional space ($d \geq 2$), find two axis-parallel squares that cover all the points of S and which are centered at the points of C, such that the size of the larger square is minimized. Let us call the solution of this problem *minimal cover*. Some variation of this problem was considered by [17], where they take $C = S$ in the plane and obtain an $O(n \log^2 n)$ time and $O(n \log n)$ space solution. From now we will call a square (rectangle) *discrete* or *constrained* if their center lies on some point of C. The main idea of the algorithm of Katz et al. [17] was to solve first the decision problem and then apply the sorted matrices technique [11] as the optimization scheme. Their decision problem was: Given a set S of n points, are there two axis-parallel discrete squares with a given area $\mathcal{A}$ whose union covers S. To solve the decision problem, they put on each point of S an axis-parallel square of size $\mathcal{A}$, thus, obtaining a set P of n squares. The decision problem was transformed to answering whether there are two points of S which intersect each square of P.

The problem we solve below is very similar to that of [17] and differs just by the fact that the centers of the squares are constrained to be in C instead

of S. Our runtime is comparable as well, but our algorithm is simpler and can be easily extended to a dynamic version of the problem, where points in S may be added or deleted, and to higher dimensional space. Below we present our algorithm for the planar case. As in [17], we solve the decision problem and then show how we apply two optimization schemes.

We start with some notations and observations. Given a set of points S, the *bounding box* of S, denoted by $B(S)$, is the smallest axis-parallel rectangle that contains S. The bounding box of S is determined by the four points, two from each axis : leftmost (smallest coordinate) and rightmost (largest coordinate) points in each of the axes, which we denote by l_x, l_y, r_x, r_y. We call these points the *determinators* of $B(S)$. Denote by X_S (Y_S) the sorted list of the points in S according to x (y) axis.

The decision algorithm Let s_1 be a square of area $\mathcal{A}$. In the decision algorithm we go over all the points of C as a center of s_1. At each step check whether we can cover the rest of the points of S (which are not covered by s_1) by a second constrained square s_2 of size $\mathcal{A}$. Denote by K the set of points which is not covered by s_1. Denote by s_{v_1} and s_{v_2} two vertical lines that go through the left and right side of s_1, respectively. Similarly, s_{h_1} and s_{h_2} are two horizontal lines that go through the bottom and the top sides of s_1, respectively. For $s_{v_1}(s_{v_2})$ we compute (by a binary search) the nearest point p (q) in X_S from the left (right) of $s_{v_1}(s_{v_2})$. For $s_{h_1}(s_{h_2})$ we compute the nearest point p' (q') in Y_S that is below (above) of $s_{h_1}(s_{h_2})$.

Let S_i^l (S_j^r) be the set that contains all the points of S with the x-coordinate that less or equal (equal or larger) to the x-coordinate of ith point (jth point) in the list X_S. Similarly, let S_k^b (S_m^t) be the set that contains all the points of S with the y-coordinate that less or equal (equal or larger) to the y-coordinate of kth point (mth point) in the list Y_S.

Observation 1 *The determinators of $B(K)$ are defined by the determinators of $B(S_i^l)$, $B(S_j^r)$, $B(S_k^b)$, $B(S_m^t)$. More precisely, the determinators of $B(K)$ are the leftmost, rightmost, lowest bottom and highest top points of the set $S_i^l \cup S_j^r \cup S_k^b \cup S_m^t$.*

This observation provides a way to solve the decision problem. For each point in C as the center for the first square s_1 we do the following:

1. Find $B(K)$. If $B(K)$ has a side of length greater than $\sqrt{\mathcal{A}}$, then the answer to the decision problem is "no".
2. Otherwise define the search region R' which is the locus of all points of L_∞ distance at most $\frac{\sqrt{\mathcal{A}}}{2}$ from all four sides of $B(K)$ and search for a point of C in R'. As was pointed in [17] R' is an axis-parallel rectangle.

As in [17] we perform orthogonal range searching [7] to determine whether there is a point of C in R'. If there is at least one point the answer is "yes"; otherwise it is "no". It remains to explain how we compute efficiently the determinators of $B(S_i^l), B(S_j^r), B(S_k^b), B(S_m^t)$. A bounding box might be empty or

degenerate, in which case we compute the rest of determinators for this bounding box. We explain the algorithm for $B(S_i^l)$.

The rightmost point p of S_i^l has been computed. The leftmost point of S_i^l is the leftmost point of S. Thus, it remains to find the lowest and highest points of the set S_i^l. These values can be precomputed for $i = 1, \ldots, n$. For the dynamic version of the problem maintaining these values will be too costly. Therefore we maintain a balanced binary search tree T as follows. The nodes of T contain the x-coordinates of the points of S. As we create the tree we maintain at each inner node the maximum of the y-coordinates of the points in the subtree rooted at this node. Thus, given the point p, the highest and lowest points of $B(S_i^l)$ can be found in $O(\log n)$ time. Similarly we do for $B(S_j^r)$, $B(S_k^b)$, $B(S_m^t)$.

Considering the time complexity of the whole algorithm. We spend $O(n \log n)$ to sort all the points of S and build T. For each point in C as a center for s_1 we compute the determinators of $B(S_i^l), B(S_j^r), B(S_k^b), B(S_m^t)$ in total $O(\log n)$ time. Checking the search region R' for a point of C takes $O(\log m)$ time using a standard orthogonal range tree with fractional cascading [7]. We have shown:

Theorem 2. *Given a set S of n demand points and a set C of m center points in the plane, one can find whether there exist two axis-parallel squares of area $\mathcal{A}$, centered at points of C, that cover all the points of S in time $O(\max(n \log n, m(\log n + \log m)))$ using $O(n + m \log m)$ space.*

Optimization If we generalize the observation in [17], we obtain that each rectilinear (x or y) distance between the points of C and the points of S (multiplied by 2 and squared) can be a potential area solution. Thus there are $O(mn)$ potential areas. One possibility for the optimization step is to use Frederickson and Johnson algorithm for sorted matrices [11]. For example all the potential size solutions defined by x distances can by represented as shown below. Define a matrix M as following : consider X_S the sorted x order of points of S and also X_C the sorted x order of points of C. Entry $(i, j), 1 \leq i \leq m, 1 \leq j \leq n$ in the matrix M stores the value $x_i^S - x_j^C$ where x_i^S is the x coordinate of the point with index i in X_S and x_j^C is the x coordinate of the point with index j in X_C. The matrix M is sorted, but some of the potential area values appear in matrix with negative sign. To overcome this difficulty, we split M into two matrices M^1 and M^2. The positive entries of M^1 are equal to M except that the negative entries are switched to be 0. In M_2 the negative entries of M become positive and the positive entries of M are switched to 0. Clearly, M^1 and M^2 are sorted matrices and they represent the set of all possible areas according to x-coordinates. Similar procedure works for the y-coordinates, and thus, we obtain four sorted matrices that represent all the possible solutions. This technique works fine in our case, but still has two disadvantages. First disadvantage is that it leads to some additive factor to the runtime of the optimization scheme ($O(m \log(2n/m))$) and second is that we need to maintain these matrices under deletions and insertions for the dynamic version of our problem.

Denote by T_d the runtime of the decision algorithm after the preprocessing step (which is $O(n \log n + m \log m)$). Instead of representing all the distances by

sorted matrices, we perform a search of the square size for each point $c \in C$ as a center for s_1. The search is for each axis and in each direction (left, right, up, down). Below we describe the algorithm for axis x, center c of s_1 and the right direction. The size of s_1 (and also s_2) is defined as follows:

1. Let the number of points of S that lie to the right of c be $0 \leq k \leq n$. We denote the x-sorted set of these points by $S^r_{n-k+1} = \{p_{n-k+1}, \ldots, p_n\}$.
2. Perform a binary search on the size of s_1. This size is defined by c and some of S^r_{n-k+1}. Namely we perform the following actions.
   (i) Find a median point $p_{n-\frac{k}{2}+1}$ in the set S^r_{n-k+1}.
   (ii) Compute the x-distance between c and $p_{n-\frac{k}{2}+1}$.
   (iii) This distance multiplied by 2 and squared defines the size $\mathcal{A}$.
   (iv) Run the decision algorithm for $\mathcal{A}$. If the answer to the decision problem is "yes", then set $k = \frac{k}{2}$ and return to step (ii). If the answer to the decision problem is "no", then set $k = k + \frac{k}{2}$ and return to step (ii).
3. Repeat the above procedure for the remaining directions.

The smallest size for which the decision algorithm answers "yes" after running it for each axis and in each direction is the solution to the optimization problem. Clearly, the described algorithm takes $O(n \log n + T_d \log n)$ time. Thus, we have

Theorem 3. *Given a set S of n demand points and a set C of m center points in the plane, one can find a minimal cover in time $O(\max(n \log n, m \log n(\log n + \log m)))$ using $O(n + m \log m)$ space.*

A related lower bound We prove a lower bound to the following (closely related to our) problem: Given an integer A and a set S of n demand points and a set C of m center points on the line, find two segments of length A centered at points of C that cover the largest possible number of points of S. An $\Omega(n \log n)$ lower bound under the linear decision tree model is achieved by a reduction from the set element uniqueness problem as in [3]. We set $C = S$ and asking the question for a limit $A = 0$. The answer is 2 if and only if the elements of the set are disjoint.

The dynamic version In the dynamic version of the problem above points may be inserted to or deleted from S. Our algorithm for static version can be extended to support dynamic updates and queries.

The sorted order of the points of S according to x and y coordinates is maintained in T as following. When we delete from or insert to T some point we should update all the maximum y-values stored at the inner nodes on the updating path from the corresponding leaf to the root. In addition, for each node v in T we store the information about the number of nodes that are in the left and right subtrees of the tree rooted at v. This information is useful to compute the median for optimization step (2.i) and to find the set S^r_{n-k+1} by a binary search in T in $O(\log n)$ time. Storing this information does not affect the running time of the insertion or deletion, since we can update while walking on the same updating path. The update of the tree T takes $O(\log n)$ time [5]. When we have a

query "What is the minimal cover?", we can run our decision algorithm together with the embeded optimization scheme using T in order to get the answer. Using the result from the section 2.1.2 we can conclude by theorem.

Theorem 4. *Given a set S of n demand points and a set C of m center points in the plane, where the points of S are allowed to be inserted or deleted, we can answer the query "What is the minimal cover?" in $O(m \log n(\log n + \log m))$ time. The update time is $O(\log n)$ for the points of S. The preprocessing time is $O(n \log n + m \log m)$.*

Higher dimensions Our algorithm can be generalized to work in any (fixed) d-dimensional space, $d \geq 3$. The changes we need to perform in order to allow this are following:

1. For the points of C we use d-dimensional orthogonal range tree [6] with a query time $O(\log^{d-1} m)$ for the static version.
2. We maintain d balanced binary search trees T_i, $i = 1, \ldots, d$ for the points of S, one for each axis. But now each node contains the $d-1$ maximal and minimal values of the other coordinates. The update scheme of T_i is done in time $O(d \log n)$.

The rest follows immediately.

Theorem 5. *Given a set S of n demand points and a set C of m center points in the d-dimensional space, $d \geq 3$, one can find a minimal cover (d-dimensional) in*

$$O(\max(n \log n, m \log n(\log n + \log^{d-1} m)))$$

time.

Theorem 6. *Given a set S of n demand points and a set C of m center points in the d-dimensional space, $d \geq 3$, where the points of S are allowed to be inserted or deleted, we can answer the query "What is the minimal (d-dimensional) cover?" in $O(m \log n(\log n + \log^{d-1} m))$ time The update time is $O(\log n)$ for the points of S. The preprocessing time is $O(n \log n + m \log^{d-1} m)$.*

2.2 An optimal decision algorithm

In this part we show that the static version of the discrete 2-center problem in the plane can be solved in $O((n+m) \log(n+m))$ time. Recall our problem. Given a set S of n points and a set C of m points in the plane, find whether exist two squares of area $\mathcal{A}$ centered at some points of C that cover the set S. We show how to solve the decision problem in linear time. After this we apply the optimization technique described in Section 2.1.2 in order to get $O((n+m) \log(n+m))$ time algorithm.

Let s_1 and s_2 be two required squares. We assume that the s_1 is left of s_2 (x-coordinate of the center of s_1 is at most the one of s_2). We also assume that the s_1 is below of s_2 (another case is similar). Let $\rho = \sqrt{\mathcal{A}}/2$. Consider the

bounding box $B(S) = [l_x, r_x] \times [l_y, r_y]$. It is clear that the center of s_1 belongs the region $R_1 =]-\infty, l_x + \rho] \times [-\infty, l_y + \rho]$. In fact we can assume that it belongs to the set of maxima of $C \cap R_1$, i.e. the set

$$\{(x, y) \in C \cap R_1 \mid \forall (x', y') \in C \cap R_1, x' < x \text{ or } y' < y\}.$$

Let L_1 denote the list of these points sorted by x-coordinate. It can be extracted from the sorted points of C in $O(m)$ time. Similarly the center of the square s_2 belongs to the list L_2 of points of minima of $\{(x, y) \in C \mid x \geq r_x - \rho, y \geq r_y - \rho\}$ which can be obtained in linear time. If one of the lists L_1 or L_2 is empty, there are no required squares such that the center of one square (s_2) dominates to other one (s_1).

We assume that the lists L_1 and L_2 are non-empty. For each point p of the list L_1, the algorithm detects whether a point q in the list L_2 exists such that the square s_1 with center $p = (p_x, p_y)$ and the square s_2 with center $q = (q_x, q_y)$ cover the set S. Let $U(p)$ denote the set of points of S whose y-coordinate are greater than $p_y + \rho$. Let $R(p)$ denote the set of points of S whose x-coordinate are greater than $p_x + \rho$. The set of points outside the square s_1 is the union of the sets $U(p)$ and $R(p)$. The set $U(p)$ is changing by insertions of points of S when the point p steps down the list L_1. Hence we can compute the leftmost points of the sets $U(p)$ for all points p of the list L_1. It takes $O(n + m)$ time. Similarly the bottommost points of the sets $U(p)$ can be computed by walking through L_1 from left to right. The set $R(p)$ is updating by insertions only if we walk through L_1 from right to left. So the leftmost and bottommost points of the set $R(p)$ can be computed in linear time.

The square s_2 covers the points outside the square s_1 if and only if $q_x - \rho$ is at most x-coordinate of the leftmost point of $U(p) \cup R(p)$ and $q_y - \rho$ is at most y-coordinate of the bottommost point of $U(p) \cup R(p)$. The points q of the list $L_2 = \{a_1, \ldots, a_k\}$ such that $q_x - \rho$ is at most x-coordinate of leftmost point of $U(p)$ form a sublist $\{a_1, \ldots, a_\alpha(p)\}$ ($\alpha(p) = 0$ if the sublist is empty). Note that $\alpha(p)$ is non-increasing sequence. We compute $\alpha(p)$ simultaneously with computing the leftmost points of $U(p)$. It can be done in linear time. The points q of the list L_2 such that $q_y - \rho$ is at most y-coordinate of the bottommost point of $U(p)$ form a sublist $\{a_\beta(p), \ldots, a_k\}$ ($\beta(p) = k + 1$ if the sublist is empty). The sequence $\beta(p)$ is non-decreasing and it can be computed simultaneously with computing the bottommost points of $U(p)$ in linear time.

Walking through L_1 from right to left, in linear time we can compute indexes $\alpha'(p)$ and $\beta'(p)$ that relate to the set $R(p)$. A pair of points p and $q = a_i$ form the solution if and only if $\max(\alpha(p), \alpha'(p)) \leq i \leq \min(\beta(p), \beta'(p))$. Such a pair can be found in linear time if the indexes $\alpha(p), \alpha'(p)), \beta(p)$ and $\beta'(p))$ are known. We have shown:

Theorem 7. *Given a set S of n demand points and a set C of m center points in the plane, one can find whether there exist two axis-parallel squares of area $\mathcal{A}$, centered at points of C, that cover all the points of S in $O(m + n)$ time.*

Remark: Unfortunately, we did not find a way for dynamizing the decision algorithm above. Any success in this direction will lead immediately to the better results for the dynamic version of the problem.

3 Rectangles

We consider first the planar version: Given a set S of n demand points and set C of m center points in d-dimensional space ($d \geq 2$), find two axis-parallel rectangles that cover all the points of S and are centered at the points of C and size of the larger rectangle is minimized. Let us call the solution of this problem *minimal rectangular cover*. Here we consider the size as a perimeter. Hershberger and Suri [13], Glozman et al. [12] and Bespamyatnikh and Segal [4] consider a similar problem, but without constraining the centers of the rectangles to be in C. They present an algorithm which runs in time $O(n \log n)$. Our algorithm runs in time $O(mn \log m \log n)$.

3.1 The decision algorithm

Assume we are given a rectangle perimeter $\mathcal{A}$. The general idea is very similar to the one used for the squares: we go over all the points in C as a center for the first constrained rectangle r_1, and at each step we check whether the rest of the points can be covered by a second discrete rectangle r_2. The difference is that we do not know the form of r_1 and r_2. In order to solve this problem our decision algorithm tries all possible placements of r_1 on points of C and checks whether the set of points not covered by r_1 can be covered by a constrained rectangle r_2. We demonstrate our algorithm for a point $c \in C$. Four lines l_1, l_2, l_3, l_4 with slopes $-1, 1, -1, 1$ in quadrants in clockwise direction, starting with a positive x and y quadrant, respectively, define the locus of all rectangles with a given perimeter $\mathcal{A}$, centered at O. The lines have to construct a 45° tilted square Q. Assume for a moment that $c = O$. Consider the $S' \subseteq S$ that contains all the points of S which are inside of intersection Q of the halfplanes defined by lines l_1, l_2, l_3, l_4 and containing c. Each point $s \in S'$ defines two rectangles with center c and the given perimeter: where s either determines the *width* of the rectangle, or its *height*. For the time being we look at the rectangle whose width is determined by s. Let s be the point that determines the widest rectangle r_1 and assume w.l.o.g. that s is to the left of c.

We shrink the width of the rectangle, keeping its corners on the corresponding lines until an *event* happens. An event is when a point of S is added to or deleted from the rectangle during the width shrinking. We check if the rest of points of S is covered by r_2. If it does then we are done; otherwise we continue to shrink the rectangle until the next event. We perform the same actions for the height as well.

In order to speed up this algorithm we define four dynamic subsets U, D, R, L of S' corresponding to the halfplanes that bound r_1. R is the set of all the points of S' that contained in the halfplane to the right of the left side of

r_1. Similarly, L (U, D) is the set of points of S' that contained in the halfplane to the left (up, down) of the right (upper, lower) side of the rectangle r_1. We define $p_r(p_l)$ to be the point x-closest to r_1 in R (L) and $p_u(p_d)$ to be the point y-closest to r_1 in U (D). Assume that we are shrinking r_1 in x direction until the next event. Assume that the x-closest neighbor of $p_r(p_l)$ in $R(L)$ is $p_r^h(p_l^h)$ and the y-closest neighbor of $p_u(p_d)$ in $U(D)$ is $p_u^v(p_d^v)$. Thus, our event is when one of p_r^h, p_l^h or p_u^v, p_d^v enters or leaves the rectangle r_1. If the next event is a point from R or L, then the number of points uncovered by r_1 increases by 1, otherwise decreases by 1. We update p_r, p_l, p_u, p_d (and also the subsets U, D, R, L). We check whether r_2 can cover the rest of points $K \subset S$ that are uncovered by r_1 by following algorithm.

We first find the determinators of the bounding box $B(K)$. For the static version of this problem, we can precompute for each set S_i^l, S_j^r, S_k^b, S_m^t the minimal and maximal values. If the length of some side of $B(K)$ is larger than $\mathcal{A}$ then the answer to the decision problem is "no". Otherwise we find a search region R' for the center of r_2. It can be done as following. We make a rectangle r_2 with a perimeter $\mathcal{A}$ and a minimal height such that r_2 covers $B(K)$ and its left lower corner of r_2 coincides with the left lower corner of $B(K)$. We slide r_2 up keeping in touch the left sides of r_2 and $B(K)$ till the left upper corners of r_1 and $B(K)$ coincide. Then we continue sliding r_2 to the right keeping in touch the upper sides of r_2 and $B(K)$, then up while touching right sides and finally to the left while touching down sides till we reach the initial position of r_2. We look onto segments on which the center of the r_2 lies during the sliding motion of the square. This defines a rectangular search region R' where can be found the center of the r_2 that covers $B(K)$, but only for this form of r_2. Generally, r_2 can have an infinite number of forms. But, as was observed in [13], all the rectangles r_2, with the same perimeter and the same lower left, have their upper right corner on particular curve Γ. In this case of perimeter Γ is a segment with slope -1. Thus we should compute R' as before for all the forms of r_2 and then take their union, thus obtaining the final search region R''. The region R'' has a form of axis-parallel rectangle rotated to 90°. In order to find whether R'' contains any point of C we perform a standard orthogonal range searching algorithm but only for coordinate axes rotated to 90°.

After preprocessing $O(n \log n)$ time, the algorithm above runs in $O(n \log m)$ time for one point $c_i \in C$ if the values of $B(S_i^l), B(S_j^r), B(S_k^b), B(S_m^t)$ are precomputed before. This is because we can carry each step of the algorithm in constant time (except of orthogonal range searching) after computing the first time boundaries of the rectangle r_1.

Thus, we have

Theorem 8. *Given a set S of n demand points and a set C of m center points in the plane, one can find whether exist two axis-parallel rectangles of perimeter $\mathcal{A}$ centered at the points of C that cover all the points of S in $O(\max(n \log n, mn \log m))$ time.*

3.2 Optimization

As in the case of squares we embed the optimization step into the decision algorithm. Similar to the squares algorithm, the explicit use of sorted matrix may lead to the additional additive factor $O(n^2)$ to the runtime for the optimization algorithm. We would like to avoid the explicit use of sorted matrices for the dynamic version of this problem by embedding the search into the decision algorithm. In our case we obtain that each pair containing one rectilinear x-distance and one y-distance between the points of S and the same point in C (multiplied by 4 and summarized) can be a potential perimeter solution. The optimization scheme is very similar to previous one, but instead of performing a binary search for each one of the directions, we define a sorted matrix M whose rows contain the sorted x-distances from $c_i \in C$ to the points of S and whose columns contain the sorted y-distances from $c_i \in C$ to the points of S. Note that the number of elements in M is n^2. Denote by T_d^i the running time of the rectangles decision algorithm for point c_i as a center of r_1. (Thus the total number of potential perimeter solution is mn^2.) Then we can perform a binary search on the elements of the matrix M, making only a constant number of calls to the decision algorithm for point c_i per iteration. As was shown in [11] the overall runtime consumed by the algorithm is $O(\Sigma_{i=1}^{m} T_d^i \log n + n)$. We obtain

Theorem 9. *Given a set S of n demand points and a set C of m center points in the plane, one can find a minimal rectangular cover in $O(mn \log m \log n)$ time.*

3.3 The dynamic version

For dynamization of the decision algorithm for rectangles we use the same updating scheme as for the decision algorithm for squares. The update and query operations the points of S remain the same. We use the same data structures as in the dynamic version of the algorithm for squares. For the optimization step we also have to take care of maintaining the sorted matrix for every point of C. It can be easily done while maintaining dynamically the sorted order of the points of S according to their x and y-coordinates. The difference form the static version is using a balanced binary search trees in the decision algorithm. Thus, we have

Theorem 10. *Given a set S of n demand points and a set C of m center points in the plane, where the points of S are allowed to be inserted or deleted, we can answer the query "What is the minimal rectangular cover?" in $O(mn \log n(\log n + \log m))$ time. The update time is $O(\log n)$ for the points of S. The preprocessing time is $O(n \log n + m \log m)$.*

3.4 Higher Dimensions

Similarly to the case of squares, our algorithm can be generalized to work in any (fixed) d-dimensional space, $d \geq 3$. The changes are exactly as in the d-dimensional algorithm for the squares, which include maintaining d-dimensional

orthogonal range tree for the points of C, d balanced binary search trees, d sorted orders of points. In addition, we perform the d-dimensional decision algorithm by fixing one dimension and applying recursively $d-1$-dimensional decision algorithm. For the optimization step the number of potential perimeters is mn^d. We can represent them as m sorted matrices, each one of the dimension d. Each sorted matrix is obtained by cartesian product of d 1-dimensional arrays, identically to the plane case. If we denote by T^d running time of the optimization algorithm (static or dynamic) in d-dimensional space, $d \geq 3$, then we can be easily verify that $T^d = O(nT^{d-1})$.

Theorem 11. *Given a set S of n demand points and a set C of m center points in the d-dimensional space, $d \geq 3$, one can find a minimal rectangular cover in $O(mn^{d-1} \log^{d-1} m \log n)$ time.*

Theorem 12. *Given a set S of n demand points and a set C of m center points in the d-dimensional space, $d \geq 3$, where the points of S are allowed to be inserted or deleted, we can answer the query "What is the minimal rectangular cover?" in $O(mn^{d-1} \log n(\log n + \log^{d-1} m))$ time. The update time is $O(\log n)$ for the points of S and $O(\log^d m)$ time for the points of C. The preprocessing time is $O(n \log n + m \log^{d-1} m)$.*

4 Conclusions

In this paper we have presented efficient algorithms for solving static and dynamic discrete 2-center problems. We generalize them to the case of rectangles and higher dimensional space. It would be interesting to consider polygons or disks as covering objects instead of squares and rectangles. The problem of finding an efficient algorithm for dynamic discrete p-center problem, general, but fixed p, also remains open.

One of the directions of our future work is to improve the running time of the discrete 2-center algorithms in the higher dimensions using direct method of the optimal algorithm in the plane.

Acknowledgments

We thank Klara Kedem for the helpful remarks and discussions.

References

1. Agarwal P., Sharir M., Welzl E.: The discrete 2-center problem, Proc. 13th ACM Symp. on Computational Geometry (1997) 147–155
2. Agarwal P., Erickson J.: Geometric range searching and its relatives. TR CS-1997-11, Duke University (1997)
3. Bajaj C.: Geometric optimization and computational complexity. PhD thesis, TR 84-629, Cornell University (1984)

4. Bespamyatnikh S., Segal M.: Covering the set of points by boxes. Proc. 9th Canadian Conference on Computational Geometry (1997) 33–38
5. Cormen T,, Leiserson C., Rivest R.: Introduction to algorithms. The MIT Press (1990)
6. Chazelle B.: A functional approach to data structures and its use in multidimensional searching. SIAM J. Computing **17** (1988) 427–462
7. de Berg M., van Kreveld M., Overmars M., O. Schwartzkopf O.: Computational Geometry, Algorithms and Applications. Springer Verlag (1997)
8. Drezner Z.: The p-center problem: heuristic and optimal algorithms. Journal Operational Research Society **35** (1984) 741–748
9. Drezner Z.: On the rectangular p-center problem. Naval Research Logist. Quart. bf 34 (1987) 229–234
10. Eppstein D.: Faster construction of planar two-centers. Proc. 8th ACM-SIAM Sympos. Discrete Algorithms (1997) 131–138
11. Frederickson G. N., Johnson D. B.: Generalized selection and ranking: sorted matrices. SIAM J. Computing **13** (1994) 14–30
12. Glozman A., Kedem K., Shpitalnik G.: 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** (1995) 26–37
13. Hershberger J., Suri S.: Finding Tailored Partitions. J. Algorithms **12** (1991) 431–463
14. Jaromczyk J. W., Kowaluk M.: 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** (1996) 71–78
15. Jaromczyk J. W., Kowaluk M.: An efficient algorithm for the euclidean two center problem. Proc. 10th ACM Symposium on Computational Geometry (1994) 303–311
16. Jaromczyk J. W., Kowaluk M.: 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** (1995) 13–25
17. Katz M. J., Kedem K., Segal M.: Constrained square-center problems. In 6th Scandinavian Workshop on Algorithm Theory (1998) 95–106
18. Katz M. J., Sharir M.: An expander-based approach to geometric optimization. SIAM J. Computing **26** (1997) 1384–1408
19. Nussbaum D.: Rectilinear p-Piercing Problems. Proc. of the Intern. Symposium on Symbolic and Algebraic Computation (1997) 316-323
20. Segal M.: On the piercing of axis-parallel rectangles and rings. Int. Journal of Comp. Geom. and Appls., to appear
21. Sharir M.: A near-linear algorithm for the planar 2-center problem. Proc. 12th ACM Symp. on Computational Geometry (1996) 106–112
22. Sharir M., Welzl E.: Rectilinear and polygonal p-piercing and p-center problems. Proc. 12th ACM Symp. on Comput. Geometry (1996) 122–132