On some geometric selection and optimization problems via sorted matrices

Alex Glozman   Klara Kedem*   Gregory Shpitalnik

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

December 8, 1998

Abstract

In this paper we apply the selection and optimization technique of Frederickson and Johnson to a number of geometric selection and optimization problems, some of which have previously been solved by parametric search, and provide efficient and simple algorithms. Our technique improves the solutions obtained by parametric search by a $\log n$ factor. For example, we apply the technique to the two-line center problem, where we want to find two strips that cover a given set S of n points in the plane, so as to minimize the width of the largest of the two strips.

Key words: computational geometry, algorithm, selection, optimization, two-line center.

A version of this paper appeared in *Fourth Workshop on Algorthms and Data Structures*, S.G. Akl, F. Dehne, J. Sack and N. Santoro, editors, Lecture Notes in Computer Science 955, Springer-Verlag, pp. 26–35.

*Work by K. Kedem has been supported by a grant from the U.S.-Israeli Binational Science Foundation, and by a grant from the Israel Science Foundation founded by The Israel Academy of Sciences and Humanities.

1 Introduction

Consider a set $\mathcal{S}$ of arbitrary elements. *Selection* in the set $\mathcal{S}$ determines, for a given rank k, an element that is k^{th} in some total ordering of $\mathcal{S}$. The complexity of selection in $\mathcal{S}$ has been shown to be proportional to the cardinality of the set [2]. Frederickson and Johnson [10] considered selection in a set of sorted matrices. An $m \times n$ matrix M is a *sorted matrix* if each row and each column of M is in a nondecreasing order. Frederickson and Johnson have demonstrated that selection in a set of sorted matrices, that together represent the set $\mathcal{S}$, can be done in time sublinear in the size of $\mathcal{S}$. They have also observed that given certain constraints on the set $\mathcal{S}$, one can construct a set of sorted matrices representing $\mathcal{S}$. For instance, given two sorted arrays, $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, the Cartesian sum $X + Y$, defined as $\{x_i + y_j \mid 1 \leq i \leq m, 1 \leq j \leq n\}$, can be *represented succinctly* as a sorted matrix M by means of X and Y ($M_{ij} = x_i + y_j$) [8]. In [7, 9, 10] a number of selection and optimization problems on trees were considered. For example, in [9] Frederickson and Johnson presented an $O(n \log n)$ time algorithm for selecting the k^{th} longest path in a tree with n nodes.

More generally, the Frederickson and Johnson algorithm for selecting the k^{th} smallest item in an $m \times n$ sorted matrix M runs in time $O(m \log(2n/m))$, when $m \leq n$ [10]. When M is sorted by columns only, their algorithm runs in time $O(n \log m)$ [10].

The optimization technique via sorted matrices follows the general idea of the parametric search [17], although this technique has a number of advantages over the parametric search technique: it does not require parallelization of the decision algorithm and, usually, it produces more efficient algorithms than the ones obtained by the parametric search technique. Assume we have a decision problem $\mathcal{P}(d)$ that receives as an input a real parameter d and we need to find the minimal value d^* of the parameter d such that $\mathcal{P}(d)$ satisfies certain properties, depending monotonically on d.

Assume, furthermore, that the set $\mathcal{S}$ of all potential values that d^* can get is presented in an $m \times n$ sorted matrix M and there is a decision algorithm A for $\mathcal{P}(d)$, which in time T decides whether the given d is equal to, smaller than, or larger than the desired value d^*. Frederickson and Johnson show [7, 9] that the overall runtime consumed by the optimization algorithm is $O(T \log n + m \log(2n/m))$ when $m \leq n$. If the matrix M is sorted by columns only then the overall runtime of the optimization algorithm is $O(T \max\{\log m, \log n\} + n \log m)$.

There have been some attempts to apply these techniques to geometric selection and optimization problems (e.g., [3, 4, 16]). In this paper we show some more such applications. In general our approach is the following: We examine geometric properties of the optimal solution for the particular problem. This leads to the determination of constraints on the set $\mathcal{S}$ of potential solutions for the problem. The constraints allow us to partition the set $\mathcal{S}$ into subsets, with a total or partial ordering defined on their elements, and to represent the subsets succinctly, as a set of sorted matrices. Next we apply the technique of Frederickson and Johnson, using as subroutines the linear time selection algorithm of [2] and the decision algorithm for the particular optimization problem. Below we present a list of the problems which we solve in this paper:

The Two-Line Center Problem. Given a set S of n points in the plane, find two strips Δ_1 and Δ_2 that together cover the set S such that the width of the larger strip is as small as possible (see Figure 1.a).

Agarwal and Sharir [1] presented for this problem an $O(n^2 \log^5 n)$ time algorithm, with an $O(n^2 \log^3 n)$ time decision algorithm and an extra factor of $O(\log^2 n)$ for the Megiddo [17] optimization scheme. Recently this result has been improved to $O(n^2 \log^4 n)$ by Katz and Sharir [14, 15] who applied an expander-based optimization approach with the decision algorithm of [1]. A recent paper by Jaromczyk and Kowaluk [13] improved this runtime to $O(n^2 \log^2 n)$ using special data structures and polar coordinates. Their data structure can be applied to a decision version algorithm that runs in time $O(n^2 \log n)$. In Section 2 we indeed plug in their decision algorithm as a subroutine in our optimization scheme and present an $O(n^2 \log^2 n)$ optimization algorithm. Our algorithm is based on new observations regarding the geometry of the problem which allow us to perform an efficient search in the set of strips that are candidates for the optimal solution.

The k^{th} Triangle Area Problem. Given a set S of n points in the plane, determine which three points define the triangle with the k^{th} largest area. This problem was stated and solved by Chazelle [3]. He presented an $O(n^2 \log^2 n)$ time algorithm based on Cole's scheme [5] for searching among similar lists, combined with the Frederickson and Johnson selection algorithm. Chazelle has conjectured that this bound can be improved to $O(n^2 \log n)$. In Section 3 we show that the geometric observations and structures which we use in our solution for the two-line center problem lead to the improved $O(n^2 \log n)$ time algorithm for the k^{th} triangle area problem.

The Min-Max Two Rectangle Problem. Given a set S of n points in the plane, find two axis-parallel rectangles r_1 and r_2 that together cover the set S and minimize the following expression: $\max(\mu(r_1), \mu(r_2))$, where μ is a two-argument nondecreasing in each argument function of the height and width of a rectangle (e.g., area, perimeter or diagonal length).

Hershberger and Suri [11] consider a *rectangular measure* $\mu(r)$ which is a two-argument, non-decreasing in each argument function of the height and width of a rectangle r. They solve the following clustering problem: Given a planar set of points S, a rectangular measure μ acting on S and a pair of values μ_1 and μ_2, does there exist a bipartition $S = S_1 \cup S_2$ satisfying $\mu(S_i) \leq \mu_i$ for $i = 1, 2$? They have presented an algorithm which solves this problem in $O(n \log n)$ time.

We use a modification of their algorithm as a decision algorithm for the min-max two rectangle problem. Then we show that the set of potential solutions for the min-max two rectangle problem can be represented as a constant number of sorted matrices. Eventually, applying the optimization technique of Frederickson and Johnson we obtain an $O(n \log n)$ algorithm for the min-max two rectangle problem (see Section 4).

The k^{th} L_∞-Interdistance Problem. Given a set S of n points in the plane, determine which pair of points of S defines the k^{th} distance in the L_∞ metric. Salowe [19] has solved this problem based on parametric search in $O(n \log^2 n)$ time. In Section 5 we present a simpler $O(n \log^2 n)$ algorithm based on our technique.

2 The Two-Line Center Problem.

2.1 Geometric Observations.

To proceed further we need some notations and definitions. Let us denote the subset of points covered by a strip Δ by S_Δ. We denote the width of a set S by $\omega(S)$ and the width of a strip Δ by $\omega(\Delta)$. We call a pair of strips Δ_1 and Δ_2 a *two-line configuration* and denote it by $\mathcal{C}(\Delta_1, \Delta_2)$. The width $w_\mathcal{C}$ of a configuration $\mathcal{C}(\Delta_1, \Delta_2)$ is defined by $max(\omega(\Delta_1), \omega(\Delta_2))$. We assume wlog that $\omega(\Delta_1) \geq \omega(\Delta_2)$. We denote an optimal two-line configuration by $\mathcal{C}^*(\Delta_1, \Delta_2)$ and the corresponding optimal width by w^* (see Figure 1.a). Let us also denote the convex hull of a set of points S, by $CH(S)$. An edge-vertex pair (e, p) of $CH(S)$ is called an *antipodal pair* if there is a supporting line to $CH(S)$ through p which is parallel to e. We observe that

Observation 1 *The width of the optimal two-line configuration $\mathcal{C}^*(\Delta_1, \Delta_2)$ is defined by an edge-vertex antipodal pair of $CH(S_{\Delta_1})$.*

Proof. Clearly Δ_1 conists of two parallel lines that support $CH(S_{\Delta_1})$. If, by contradiction, they support $CH(S_{\Delta_1})$ at two vertices, then there is one direction (clockwise or anticlockwise) in which this strip can be rotated, maintaining support of $CH(S_{\Delta_1})$ and shrinking in width, hence contradicting the assumption that $\mathcal{C}^*(\Delta_1, \Delta_2)$ is an optimal configuration. ∎

Observation 1 implies that there is a triplet of points $p_1, p_2, p_3 \in S$ that defines the width of the optimal two-line configuration. More precisely, the line (p_1, p_2) defines the slope of strip Δ_1 in the optimal configuration and the point p_3 defines its width w^* (see Figure 1.b).

We call the set of all values of the parameter w, induced by the triplets $p_1, p_2, p_3 \in S$, the *set of potential solutions* of the two-line center problem, and denote it by $\mathcal{W}$. We will perform a binary search over the set $\mathcal{W}$ using the decision algorithm of [13] as an oracle. In order to do it efficiently we will present the set $\mathcal{W}$ in a certain succinct form.

We denote by $\mathcal{L}$ the list of lines, passing through pairs of points in S and sorted by their slopes. Clearly, there are $N = n(n-1)/2$ such lines. We associate with each line $l_i \in \mathcal{L}$ $(i = 1, \ldots, N)$ a line $\bar{l}_i$ which is perpendicular to l_i. We denote by L_i the list of points of S ordered from left to right according to their orthogonal projection on $\bar{l}_i$. The following observation is straightforward (see [6])

Observation 2 *Given two consecutive lines $l_i, l_{i+1} \in \mathcal{L}$ and the list L_i, then the list L_{i+1} can be obtained from L_i by a single swap of points $p', p'' \in S$, where p' and p'' is the pair of points which defines the line l_i.*

Figure 1.c illustrates the line $l_i = l$ determined by p' and p''. The two points are projected to one point on $\bar{l}$. Clearly when we rotate l (with its perpendicular line $\bar{l}$) very slightly clockwise, and then anticlockwise, the projections of p' and p'' swap order on $\bar{l}$, while the other points remain in the same order as before.

Observation 3 *Given the list L_i and the pair of points $p', p'' \in S$ which defines the line l_i. The point $p \in S$ having the k^{th} distance from line l_i, among the points of S lying on the right (left) side of line l_i, can be found in constant time. (Here $i = 1, \ldots, N$ and $k = 1, \ldots, n$).*

Proof. The list L_i can be maintained as a sorted array, ordered along the projections of the points on $\bar{l}_i$ from left to right. We know the (consecutive) indices of the points p' and p'' in L_i, say $p' = p_a$ and $p'' = p_{a+1}$. The point $p \in S$ with k^{th} distance from l_i, among the points of S to the right of l_i, is $p = p_{a+1+k}$. (Symmetrically, on the left side of l_i, it is $p = p_{a-k}$, see Figure 1.d.) If the resulting index is greater than n (or, symmetrically, less than 1) then there is no such point in S and we assume that the k^{th} distance is $+\infty$. ∎

2.2 The Representation of $\mathcal{W}$.

We build a list $\mathcal{R}$ of N records. The first record $r_1 \in \mathcal{R}$ remains empty for the time being. Each record $r_i \in \mathcal{R}$ $(i = 2, \ldots, N)$ consists of the two indices i', i'' in L_{i-1} of the two elements which have to be swapped in order to produce L_i. Clearly, with this structure available, and with the first list L_1, we can produce all the lists L_i $(i = 2, \ldots, N)$ successively by proceeding from one record of $\mathcal{R}$ to the next one, each time performing a single swap of two elements of L_{i-1}.

In the following preprocessing algorithm we build the structure $\mathcal{R}$. We use a dynamic auxiliary array, $Index$, which contains, at the i-th step, pointers from the points of S to their location in the list L_i. (E.g., if $p_k \in S$ is the third point in L_i then $Index[k] = 3$.)

The Preprocessing Algorithm:

1. Produce the list $\mathcal{L}$ by sorting all the lines l_i by their slopes. Store them by means of the pairs of points of S defining the lines.

2. For the first line $l_1 \in \mathcal{L}$, sort the set of points S according to their projections on $\bar{l}_1$, store the sorted set in L_1, and the corresponding pointers in $Index$.

3. For each record r_i $(i = 2, \ldots, N)$ do:

(a) Assume that the line l_i is defined by a pair of points with indices $1 \leq j_1, j_2 \leq n$. Perform $i' = Index[j_1]$ and $i'' = Index[j_2]$.

(b) Swap the elements with indices i', i'' in the list L_{i-1}.

(c) Perform $Index[j_1] = i''$ and $Index[j_2] = i'$.

Lemma 4 *The preprocessing algorithm above builds correctly the structure $\mathcal{R}$ in $O(N \log N)$ time, $\mathcal{R}$ requires $O(N)$ space.*

Proof. The correctness of the algorithm can be proved by induction on Step 3, applying Observation 2 $N-1$ times. Step 1 of the algorithm requires $O(N \log N)$ time. Step 2 consumes $O(n \log n)$ time. Steps 3(a) – 3(c) require a constant amount of time each. Clearly each record r_i $(i = 1, \ldots, N)$ is of constant size. This completes the proof. ∎

Corollary 5 *The structure $\mathcal{R}$ can be traversed record by record in $O(N)$ time, allowing access to the list L_i at each step i, $i = 1, \ldots, N$.*

Corollary 6 *The list L_1 and the structure $\mathcal{R}$ provide a succinct representation of the set $\mathcal{W}$.*

2.3 The Algorithm.

In this section we present our algorithm which solves the two-line center problem. It uses the following result of Frederickson and Johnson [7, 8]: Given a matrix M of size $m \times n$ with each column sorted in ascending order, and provided a constant time access to each element of M, one can solve the optimization problem over the values of the matrix M in time $O(T \max\{\log m, \log n\} + n \log m)$, where T is the runtime of the decision algorithm.

The structure $\mathcal{R}$ and L_1 succinctly represent a matrix M of size $n \times N$, sorted by its columns, which contains $\mathcal{W}$. In this matrix the i^{th} column, c_i, corresponds to one line $l_i \in \mathcal{L}$, and contains the sorted array of distances from the points in S to the line l_i. Notice that instead of maintaining all the lists L_i, $i = 1, \ldots, N$, we just keep L_1 (of length n), and the structure $\mathcal{R}$ with N records, each containing zero to four integer numbers. It follows from Corollary 5 and Observation 3 that given a column $c_i \in M$ we can access an element of it in a constant time. This representation is therefor sufficient for the application of the optimization algorithm of Frederickson and Johnson [9]. Below we apply their algorithm to our problem, adding some steps for efficiently accessing the elements in M. The algorithm recurses on *subarrays* which are sub-columns in M. Associated with each subarray s is its smallest element (smallest distance) $\min(s)$ which is the element with smallest row index in the subarray, and its largest element $\max(s)$ which is the element with largest row index in the subarray.

Initially we subdivide each column of M into four equal size subarrays. We use the decision algorithm [13] to decide which of the subarrays should be discarded and which will take part in the continuation of the search. We discard at least half of the subarrays [9], divide each subarray into four and recurse. When there is only one element in each subarray we combine them to one array, sort it and search for the minimum width with the standard binary search and the decision algorithm. The number of discarded subarrays in each column is unknown, but we know that the subarrays that are left in the column for the continuation of the search have consecutive row indices in the column (because M is sorted by columns). We maintain in $\mathcal{R}$ also the lowest and the highest indices of the subarrays left in each column c_i, $i = 1, \ldots, N$. Assume that at the end of the preprocessing algorithm we added to each record of $\mathcal{R}$ the indices 1 and n as the beginning and end of the initial subarrays.

The Optimization Algorithm:

1. Perform the preprocessing algorithm.

2. Perform the following steps until there is one element in each subarray:

(a) Produce the columns c_i of the matrix M one column after the other, starting from the first column $c_1 \in M$, and then traversing $\mathcal{R}$ to achieve each c_i in its turn.

   Partition the subarrays in each column c_i into four subarrays of equal size and select the minimum and maximum elements from each subarray, forming two lists: L_{min} of the minimum values and L_{max} of the maximum values, respectively, of the subarrays.

(b) Select the medians V_{min} of L_{min} and V_{max} of L_{max}.

(c) Run the decision algorithm of [13] on V_{min} and V_{max}.

(d) Produce the columns of the matrix M successively again. This time, by going over the subarrays of each column c_i, discard the subarrays according to the answers of the decision algorithm in the previous step, store in $\mathcal{R}$ the lowest and highest indices of the part of c_i that was not discarded.

3. Find the optimal value w^* by performing a binary search

4. Sort the remaining $O(N)$ values (one from each subarray), and find the optimal value w^* by performing a binary search using the decision algorithm of [13].

Discarding a subarray s is done according to the result of the decision algorithm on V_{min} and V_{max}. Assume, wlog, that the decision algorithm found that V_{min} is smaller than the optimal width and V_{max} is larger than the optimal width. In this case a subarray s is discarded if $\min(s) \geq V_{max}$ or if $\max(s) \leq V_{min}$. The other cases are treated similarly.

2.4 Analysis of the Algorithm.

Step 1 requires $O(N \log N)$ time as was shown in Lemma 4. From the analysis in [9, 10] it follows that Step 2 is repeated $O(\log n)$ times and the number of the subarrays in each stage of Step 2 is $O(N)$. In step 2(a) we partition every subarray in each column of M into four subarrays of equal size, and form two lists L_{min} and L_{max}. Corollary 5 implies that Step 2(a) can be performed in $O(N)$ time. Indeed, as was shown in Observation 2, we can obtain the column c_{i+1} from column c_i by making a single swap in the list L_i and producing the list L_{i+1}. Moreover, the selection of a minimal or maximal element in each subarray can be done in constant time as stated in Observation 3. In Step 2(b) we apply the linear time median algorithm of [2] in time $O(N)$. In Step 2(c) we run twice the decision algorithm, hence it runs in $2 \cdot T$ time. Step 2(d) requires the same runtime as Step 2(a). Thus we see that one iteration of Step 2 requires $O(T+N)$ time, and multiplying by the number of iterations we get to a total time of $O((T+N)\log n)$ for Step 2. Step 3 performs a sort of $O(N)$ elements in $O(N \log N)$ time, and then $O(\log N)$ decision algorithms, for a total of $O((T+N)\log N)$ time. Thus our algorithm runs in time $O((T+N)\log N)$. Plugging in this formula the runtime of the decision algorithm $T = O(n^2 \log n)$ [13], and $O(n^2)$ for N, we obtain an $O(n^2 \log^2 n)$ algorithm for the two-line center problem. It can be easily seen that the space requirement of the algorithm is $O(n^2)$. We conclude with the following theorem:

Theorem 7 *The algorithm above solves the two-line center problem in $O(n^2 \log^2 n)$ time and $O(n^2)$ space.*

3 The k^{th} Area Triangle Problem.

The k^{th} area triangle problem is stated as follows: Given a set S of n points in the plane, determine which three points define the triangle with the k^{th} largest area.

Consider all triplets of points of the set S. Clearly, each triplet $p_1, p_2, p_3 \in S$ defines a triangle T. We can compute the area of T as the length of the segment $[p_1, p_2]$ times half the distance from point p_3 to the line through p_1, p_2. Let us denote by $\mathcal{T}$ the set of all possible areas of triangles, determined by S. The data structures for this problem are identical to those in the former section. As in the former section we can represent the set $\mathcal{T}$ succinctly in a matrix M of size $n \times N$, sorted by its columns, which contains all the areas. Each column, c_i, corresponds to a line $l_i \in \mathcal{L}$, and thus is defined by some pair of points of S. The base of a triangle for this column is the distance between the points, and the rows within the column c_i store the sorted distances from the points of S to the line l_i. We use the selection algorithm of Frederickson and Johnson [8], which states the following: Given a matrix M of size $m \times n$, with each column sorted in ascending order, and provided a constant time access to each element of M, one can solve the selection problem over the values of M in time $O(n \log m)$.

Our algorithm for the k^{th} area triangle problem is almost identical to the algorithm for the two-line center problem. The difference is that instead of calling a decision algorithm as a subroutine it runs a linear time selection algorithm [2]. Thus we have proved the following theorem:

Theorem 8 *The k^{th} area triangle problem can be solved in $O(n^2 \log n)$ time consuming $O(n^2)$ space.*

4 The Min-Max Two Rectangle Problem.

The min-max two rectangle problem is formulated as follows: Given a set S of n points in the plane, find two axis-parallel rectangles r_1 and r_2 that together cover the set S and minimize the expression: $\max(\mu(r_1), \mu(r_2))$, where μ is a two-argument nondecreasing in each argument, function of the height and the width of a rectangle (for example area, perimeter or diagonal length).

4.1 The Decision Algorithm for the Min-Max Two Rectangle Problem.

In this subsection we present an algorithm which solves the following decision problem: Given a planar set of points S, a rectangular measure μ acting on S and a value μ', does there exist a bipartition $S = S_1 \cup S_2$ satisfying $\max(\mu(S_1), \mu(S_2)) \leq \mu'$. Hershberger and Suri [11] consider such a rectangular measure $\mu(r)$ of the height and the width of a rectangle r and solve the following clustering problem: Given a planar set of points S, a *rectangular measure* μ acting on S and a pair of values μ_1 and μ_2, does there exist a bipartition $S = S_1 \cup S_2$ satisfying $\mu(S_i) \leq \mu_i$ for $i = (1, 2)$? They present an algorithm which solves this problem in $O(n \log n)$ time. First we review some notations and observations from [11]. The *bounding box* of the given point set S, denoted by ∂S, is the smallest axis-parallel rectangle that contains S. The bounding box of S is determined by the topmost,bottommost, leftmost, and rightmost points of S, which we denote by t,b,l and r (see Figure 2). We call these points *determinators* of ∂S. Clearly, the rectangles r_1 and r_2 have to cover together these four points. There are essentially three different ways to partition the determinators between sets S_1 and S_2:

1. One of the subsets gets three determinators and the other gets one determinator.

2. Each subset gets two determinators lying on opposite sides of ∂S.

3. Each subset gets two determinators lying on adjacent sides of ∂S.

According to the first case, assigning three determinators to, say S_1, fixes three of the four sides of r_1. Thus we need to find a placement of the fourth side of r_1 such that $\mu(r_1) \leq \mu'$ and $\mu(r_2) \leq \mu'$. We place the fourth side of r_1 such that $\mu(r_1) = \mu'$. In linear time we can test whether the set of points outside of r_1 can be covered by a rectangle with measure $\mu \leq \mu'$ (see Figure 2.a).

For the second case our decision algorithm proceeds in a way similar to [11]. Assume, wlog, that r_1 is defined by the top and bottom determinators and r_2 is defined by the left and right determinators. The fixed height of r_1 determines a width w_1 satisfying $\mu(r_1) = \mu'$. The algorithm slides r_1 of width w_1 from left to right over the set S, assigning all points within r_1 to S_1 and all the points outside it to S_2. At each sliding step the algorithm checks whether $\mu(S_2) \leq \mu'$. There sre O(n) steps in this algorithm (see Figure 2.b). Clearly, the set S_2 can be partitioned into two subsets S_2^l and S_2^r lying to the left and to the right of the sliding rectangle r_1, respectively, such that the set S_2^l underegos insertions only and the set S_2^r underegos deletions only. We can obtain an extremal point $p \in S_2^l$ in a given direction in a constant time per insertion by maintaining a pointer to the current extremal point p and comparing it with the inserted point; for dealing with points in S_2^r there exist standard data structures that allow to perform deletions and extremum querys off-line in a constant time per query after $O(n \log n)$ preprocessing (see [18]). Also, one can easily show that if the order of points of the set S along the x-axis (y-axis) is known then the ordered list of updates to the sets S_2^l and S_2^r can be constructed in linear time. It implies that after $O(n \log n)$ preprocessing the decision algorithm for the second case runs in $O(n)$ time.

In the third case we assign two adjacent determinators, say l and b, to r_1, and t and r to r_2. In this case the lower left corner of r_1 must coincide with the lower left corner of ∂S (let us define it by z as in [11]). Similarly, the upper right corner of r_2 must coincide with the upper right corner of ∂S. The decision algorithm for this case is based on the same idea as in the previous case. Namely, it trys all placements of r_1 of measure μ' and checks whether the set of points outside of r_1 can be covered by rectangle r_2 , such that $\mu(r_2) \leq \mu'$. As was observed in [11], all the rectangles r_1, satisfying $\mu(r_1) = \mu'$, whose lower left corner is z, have their upper right corner on a particular curve Γ. If μ is the area then Γ is a hyperbola; if μ is the perimeter then Γ is a segment with slope -1; and if μ is the diagonal length then Γ is a quarter-circle of radius μ' centered at z. Figure 2.c illustrates the case where μ is the perimeter. As seen in the figure, the set S_2 can be partitioned into three subsets S_2^{out}, S_2^a and S_2^r. The subset S_2^{out} consists of the points of S lying in ∂S to the right of Γ, outside of r_1. The set S_2^a consists of the points above the sliding rectangle r_1 and to the left of Γ, and the set S_2^r consists of the points to the right of r_1 and to the left of Γ. Clearly, S_2^{out} does not change during the running of the algorithm, since r_1 cannot cover any point from S_2^{out}. When sliding the upper right corner of r_1 along Γ from its topmost point to its bottommost point, the set S_2^a underegos insertions only and the set S_2^r underegos deletions only. As above, if the order of the points of S along the x-axis (and y-axis) is known, then the ordered list of updates for sets S_2^a and S_2^r can be constructed in linear time. It implies that after $O(n \log n)$ preprocessing the algorithm for the third case runs in $O(n)$ time. These observations lead to the following lemma:

Lemma 9 *After $O(n \log n)$ preprocessing time, the decision algorithm for the min-max two rectangle problem runs in $O(n)$ time.*

4.2 The Optimization Step.

We make the following two observations:

Observation 10 *The set of all distances measured along the x-axis (y-axis) between the points of S can be represented as a sorted matrix M of size $n \times n$ if the order of the points of S along the x-axis (y-axis) is known.*

Observation 11 *Given two sorted arrays X and Y, the values of the monotone, nondecreasing in each argument, function $F(X,Y)$, can be represented succinctly as a sorted matrix M of size $n \times n$ by means of X and Y.*

Proof. It is a straightforward extension of the case where the function F(X,Y) is a Cartesian product of two arrays X and Y [8]. ∎

Let us consider the geometry of the optimal solution for the min-max two rectangle problem and determine the potential values of μ for the optimal solution. Clearly there can be three kinds of solutions according to the three cases of positioning the determinators on the rectangles r_1 and r_2. In the first and the second cases the bigger rectangle has its height (width) fixed and thus its measure μ is defined by the second parameter - width (height). By Observation 10 we can represent all these values as a sorted matrix M of size $n \times n$. In the third case the optimal rectangle with the larger measure μ, say r_1, has its width and height defined by two pairs of points lying on its opposite sides, where one point in each pair is a determinator of r_1. The distances from each such determinator to the other points of S, measured along the x-axis (y-axis respectively) define an array X (resp. Y) of linear size. Having the arrays X and Y available we can represent all potential values of μ in the third case as a sorted matrix M of size $n \times n$, as stated in Observation 11. These observations lead to the following lemma:

Lemma 12 *The set of potential values of μ can be represented as a constant number of sorted matrices of size $n \times n$ each.*

Taking into account Lemma 9 and Lemma 12 and applying the optimization technique of Frederickson and Johnson [7, 9] we obtain the following result

Theorem 13 *The Min-max two rectangle problem can be solved in $O(n \log n)$ time.*

5 The k^{th} L_∞-Interdistance Problem.

Salowe [19] has solved the following problem: Given a set S of n points in the plane, determine which pair of points of S defines the k^{th} distance in metric L_∞. Following the parametric search paradigm Salowe has presented a decision algorithm, which in $O(n \log n)$ time decides whether the given value d is less, equal or greater than the k^{th} interdistance in metric L_∞. Applying the parametric search technique he was able to obtain an $O(n \log^2 n)$ solution to the k^{th} L_∞-interdistance problem.

We start with the following simple observation:

Observation 14 *The k^{th} L_∞-interdistance is defined by the distance between a pair of points of the set S, measured along the x-axis or the y-axis.*

By combining this observation and Observation 10 with the optimization technique of Frederickson and Johnson [7, 9] and using the same decision algorithm as in [19] we obtain the following result:

Theorem 15 *The k^{th} L_∞-interdistance problem can be solved in $O(n \log^2 n)$ time.*

6 Summary

In this paper we solve efficiently a number of geometric selection and optimization problems by exploring the geometry of the optimal configurations. The discovering of some partial or total ordering on the set of potential solutions allows one to apply the technique of Frederickson and Johnson for the selection or the optimization part of the problem.

7 Acknowledgement

The authors thank the referees for their relevant and helpful remarks.

References

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

[2] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest and R.E. Tarjan, "Time bounds for selection", *J. Comput. Syst. Sci.*, 7(1972), pp. 448–461.

[3] B. Chazelle, "New techniques for computing order statistics in Euclidean space", *Proc. 1st ACM Symp. on Computational Geometry*, 1985, pp. 125–134.

[4] L. P. Chew and K. Kedem, "Improvements on geometric pattern matching problems", *SWAT, LNCS 621, Springer-Verlag*, 1992, pp. 318–325. Revised version to appear as "Getting around a lower bound for the minimum Hausdorff distance", *Computational Geometry: Theory and Applications.*

[5] R. Cole, "Searching and storing similar lists", *J. Comput. Syst. Sci.*, 23(1981), pp. 166–204.

[6] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, Germany, 1987.

[7] G.N. Frederickson, "Optimal algorithms for tree partitioning", *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, 1991, pp. 168–177.

[8] G. N. Frederickson and D. B. Johnson, "The complexity of selection and ranking in X+Y and matrices with sorted columns", *J. Comput. Syst. Sci.*, 24(1982), pp. 197–208.

[9] G. N. Frederickson and D. B. Johnson, "Finding k^{th} paths and p-centers by generating and searching good data structures", *J. Algorithms*, 4(1983), pp. 61–80.

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

[11] J. Hershberger and S. Suri, "Finding tailored partitions", *J. Algorithms*, 12(1991), pp. 431–463.

[12] M. Houle and G. Toussaint, "Computing the width of a set", *Proc. 1st ACM Symp. on Computational Geometry*, 1985, pp. 1–7.

[13] J.W. Jaromczyk, 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.

[14] M.J. Katz, "Improved algorithms in geometric optimization via expanders", *Proc. 3rd Israel Symposium on Theory of Computing and Systems*, 1995, pp. 78–87.

[15] M.J. Katz and M. Sharir, "An expander-based approach to geometric optimization", *Proc. 9th ACM Symp. on Computational Geometry*, 1993, pp. 198–207.

[16] D. Kravets and J. Park, "Selection and sorting in totally monotone arrays", *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, 1990, pp. 494–502.

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

[18] M. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane", *J. Computer and System Sciences*, 23(1981), pp. 166–204.

[19] J. S. Salowe, "L-infinity interdistance selection by parametric search", *Inf. Proc. Lett.*, 30(1989), pp. 9–14.

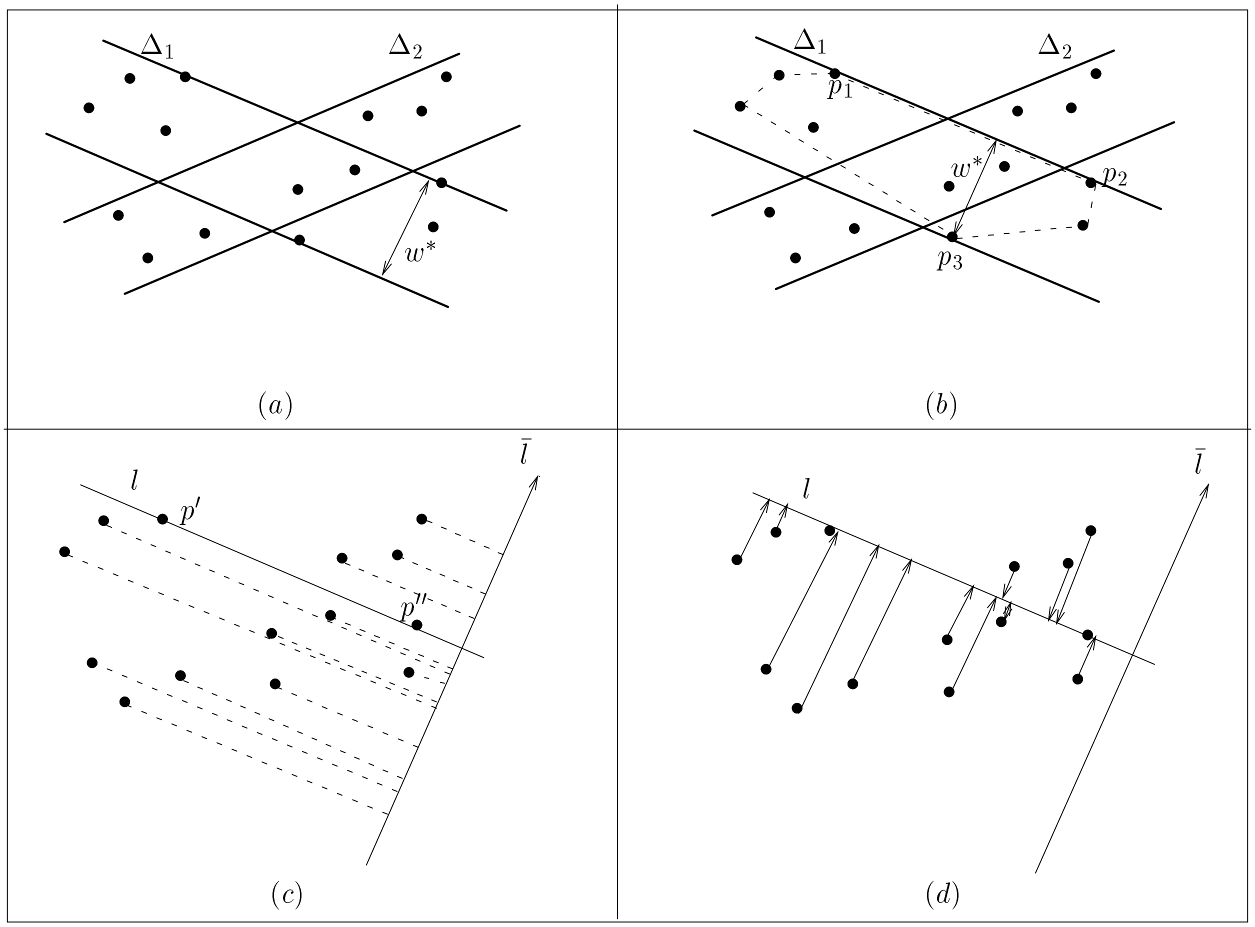

Figure 1: Two-line center problem.

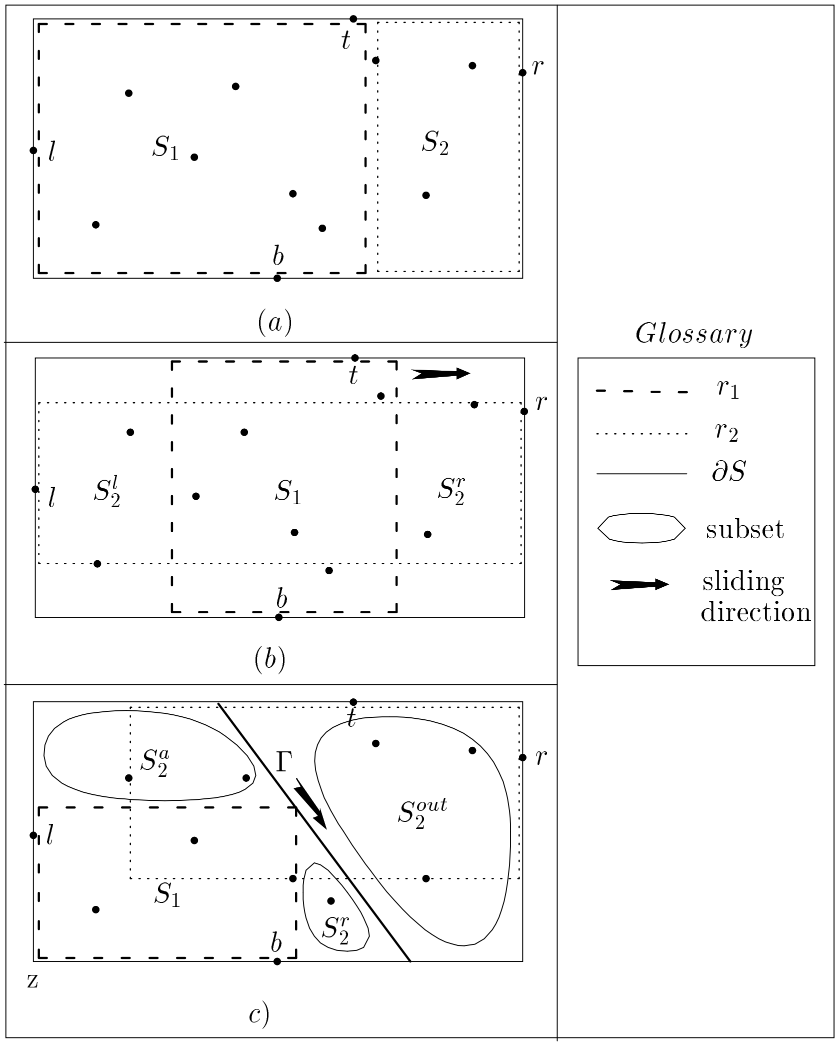

Figure 2: The decision algorithm for min-max two rectangle problem.