The Maximum Box Problem for Moving Points in the Plane

S. Bereg* J.M. Díaz-Báñez † P.Pérez-Lantero‡
I. Ventura †

August 7, 2009

Abstract

Given a set R of r red points and a set B of b blue points in the plane, the static version of the Maximum Box Problem is to find an isothetic box H such that $H \cap R = \emptyset$ and the cardinality of $H \cap B$ is maximized. In this paper, we consider a kinetic version of the problem where the points in $R \cup B$ move move along bounded degree algebraic trajectories. We design a compact and local quadratic-space kinetic data structure (KDS) for maintaining the optimal solution in $O(r \log r + r \log b)$ time per each event. We also give an algorithm for solving the more general static problem where the maximum box can be arbitrarily oriented. This is an open problem in [1]. We show that our approach can be used to solve this problem in $O((r+b)^2(r \log r + r \log b))$ time. Finally we propose an efficient data structure to maintain an approximated solution of the kinetic Maximum Box Problem.

1 Introduction

In Pattern Recognition and Classification problems, a natural method for selecting prototypes that represent a class is to perform cluster analysis on the training data [7]. Typically, two sets of points X^+ and X^- are given, and one would like to find patterns which intersect exactly one of these sets. The clustering can be obtained by using simple geometric shapes such as circles or boxes. Recent papers deal with the *Maximum Box Problem*, where the clustering is due by considering maximum boxes, i.e., boxes containing the maximum number of points in the given data set. See [9, 10]. The basic problem is the following:

*Department of Computer Science, University of Texas at Dallas, Box 830688, Richardson, TX 75083.

†Partially supported by Grant MEC-MTM2006-03909.
Departamento de Matemática Aplicada II, Universidad de Sevilla, Camino de los Descubrimientos s/n, 41092 Sevilla, Spain.

‡Departamento de Computación, Universidad de La Habana, San Lázaro y L, 10400 La Habana, Cuba.

given a finite set of blue points $B = X^+$ and a finite set of red points $R = X^-$ in the plane, compute a box (axis-aligned rectangle) containing the maximum number of points of B but avoiding points of R. In many applications, the data are given in a dynamic scenario. For instance, in fixed wireless telephony access and driving assistance, detection and recognition of patterns for moving objects are key functions [2]. In fact, with the continued proliferation of wireless communication and advances in positioning technologies, algorithms to efficiently solve optimization problems about large populations of moving data are gaining interest. New devices offer to the companies an opportunity of providing a diverse range of e-services, many of which will exploit knowledge of the user's changing location. This results in new challenges to database and classification technology [4].

In this paper, we introduce and investigate the dynamic version of the Maximum Box Problem where the dataset is modeled by points moving along bounded degree algebraic trajectories. We present a *Kinetic Data Structure* (KDS) to efficiently maintain the maximum box for a bi-colored set of moving points. A KDS is used for keeping track the attributes of interest in a system of moving objects [8]. The main idea in the kinetic framework is that even though the points move continuously, the relevant combinatorial structure changes only at certain predictable discrete events. Therefore, one does not have to update the data structure continuously. These events have a natural interpretation in terms of the underlying structure, for example, when the x- or y-projections of two points coincide.

The paper is organized as follows. In Section 2 we give an algorithm for the static version of the Maximum Box problem that will be useful for the dynamic version. A new data structure for maintaining the maximum box in a kinetic setting is proposed in Section 3. In Section 4 we give an algorithm for the Maximum Box Problem in a static setting for which the box can be arbitrarily oriented. Finally, in Section 5 we present an efficient data structure to maintain an approximated solution of the Maximum Box Problem when the points move.

2 The Static Version

The static version of the maximum box problem in the plane has been already studied. In [9], an $O(b^2 \log b + br + r \log r)$-time algorithm has been presented, where $r = |R|$ and $b = |B|$. We propose here a simple algorithm which will be used later to design a data structure for the kinetic version. First, we observe that a maximum box for B and R can be enlarged until each of its sides either contains a red point or reaches the infinity. Thus, the maximum box can be transformed to one of the following isothetic objects with red points on its boundary and no red points inside: a rectangle, a half-strip, a strip, a quadrant or a half-plane (see Figure 1). By symmetry we can assume that the top side of the box contains a red point.

We show how to compute the maximum box of type 1), 2) and 3) shown in Figure 1. The cases of a quadrant 4) and a half-plane 5) are even easier and

can be addressed by applying the techniques based on the *Dynamic Bentley's Maximum Subsequence Sum Problem* presented in [6]. Both cases can be solved in $O(n \log n)$ time. Actually, the simple cases 4) and 5) can be solved using our approach (below) if we allow sweeping to infinity (the sweep can stop when all points are swept).

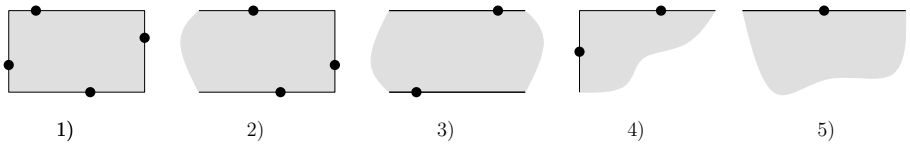

Figure 1: Configurations of the maximum box.

We proceed as follows. For every red point p_r, we compute a rectangle $H(p_r)$ such that
(i) the top side of $H(p_r)$ contains p_r,
(ii) the interior of $H(p_r)$ contains the maximum number of blue points, and
(iii) the boundary of $H(p_r)$ contains only red points.
Then, we take the best of all $H(p_r)$. To do this, for each red point p_r, we draw a horizontal line l passing through p_r and sweep the points below l by moving l downwards.

Let h_r be the horizontal line that passes through p_r. During the sweeping we maintain two balanced binary search trees T_R and T_B such that T_R (resp. T_B) contains all the red (resp. blue) points that lie between h_r and l sorted by x-coordinate. In the tree T_B, for each node v we also maintain a counter of the number of blue points in the subtree rooted at v. When l passes through a point p we do the following. Let $left(p)$ (resp. $right(p)$) be the rightmost (resp. leftmost) red point in T_R located to the left (resp. right) of the vertical line that passes through p, and let x_{left} (resp. x_{right}) be its x coordinate. If $left(p)$ (resp. $right(p)$) does not exist then x_{left} (resp. x_{right}) is $-\infty$ (resp. $+\infty$). We only process p if the x coordinate of p_r lies in the interval (x_{left}, x_{right}). In that case, p is inserted in T_R or T_B according to its color and if p is a red point, we consider a candidate $H(p_r)$ as the isothetic rectangle (half-strip or strip) H_p whose sides pass through $left(p)$, $right(p)$, p_r and p. The count of blue points inside H_p is equal to the count of blue points in T_B whose x coordinate lies in (x_{left}, x_{right}).

In the above procedure $left(p)$, $right(p)$ and the count of blue points inside H_p are obtained in logarithmic time each, thus the top-down sweeping from p_r is done in $O(r \log r + r \log b + b \log b)$ time, giving then an overall $O(r^2 \log r + r^2 \log b + rb \log b)$-time and $O(r + b)$-space process. In many applications, one of the classes, say the negative data, (the red points) has few elements with respect the positive data (the blue points). It means that $r \ll b$ and thus our complexity is essentially $O(b \log b)$.

We consider the problem of computing the *smallest-area* axis-parallel rectangle (box) that maximizes the number of covered points from B and does not

contain any point from R. We refer to this problem as SAMBP. Segal [10] designed an algorithm for SAMBP with $O(n^3 \log^4 n)$ running time.

We show that our approach can be applied to solve SAMBP. Note that none of the boxes $H(p_r)$ is the solution of SAMBP since every $H(p_r)$ contains red points on its boundary (or the boundary extentds to infinity). We augment tree T_B as follows. For each node $v \in T_B$, we store the smallest and largest y-coordinates of points in the subtree rooted at v. Since the smallest and largest x-coordinates of these points can be computed using the sorted x-order, the smallest bounding box of blue points in $H(p_r)$ can be computed in $O(\log b)$ time. The additional cost is $O(\log b)$ per one sweeping step. The overall time does not change.

It remains to show that the algorithm is correct. This can be shown by taking an optimal axis-parallel rectangle and enlaging it (horizontally and vertically). The algorithm checks all such boxes and, therefore, is correct.

Theorem 2.1 *SAMBP can be solved in* $O(r^2 \log r + r^2 \log b + rb \log b)$ *time using* $O(r + b)$ *space.*

Note that our algorithm improves the runing time of the previous algorithm [10] by a factor of $O(n \log^3 n)$ where $n = r + b$.

3 The Maximum Box Problem for moving points

In this section we introduce a new kinetic data structure (KDS), for maintaining the maximum box. A KDS is a structure that maintains an attribute of a set of continuously moving objects [8]. It consists of two parts: a combinatorial description of the attribute and a set of certificates. The certificates are elementary tests on the input objects with the property that as long as their outcomes do not change, the attribute does not.

Lemma 3.1 *Let X (resp. Y) be the elements of $R \cup B$ sorted by abscissa (resp. ordinate). The maximum box for R and B is univocally determined by X and Y, and it does not change combinatorially over time as long as X and Y do not.*

We design a KDS called *Maximum Box Kinetic Data Structure* (MBKDS). In our problem, the set of moving objects is $R \cup B$ and the attribute is its maximum box. According to Lemma 3.1, we consider as a certificate the condition that two consecutive points in X (resp. Y) satisfy the x-order (resp. y-order). An event is the failure of a certificate at some instant of time t and it implies an update of the MBKDS. We denote each event as a pair $\langle c, t \rangle$ where c is the certificate and t is the instant of time where c is violated. We name an event as *flip*.

A KDS is *compact* if it has *few* certificates and *local* if no object participates in *too many* certificates. With this it can be updated easily when the flight plan

of an object changes (see [8]). It is easy to prove that the MBKDS is compact and local.

The MBKDS is composed by the event queue $\mathcal{E}$ and the structure $\mathcal{D} = \{\mathcal{D}(p_r) \mid p_r \in R\}$ where $\mathcal{D}(p_r)$ is a data structure for each red point p_r. When an event corresponding to the certificate of two consecutive elements of X, say $[X_i, X_{i+1}]$, occurs it is removed from $\mathcal{E}$. Then, we remove from $\mathcal{E}$ the events corresponding to $[X_{i-1}, X_i]$ and $[X_{i+1}, X_{i+2}]$. After that, we swap X_i and X_{i+1} and insert in $\mathcal{E}$ new events for $[X_{i-1}, X_i]$, $[X_i, X_{i+1}]$ and $[X_{i+1}, X_{i+2}]$. The event time is computed based on our knowledge of the motions of X_{i-1}, X_i, X_{i+1} and X_{i+2}. $\mathcal{E}$ is designed as a priority queue with its basic operations in logarithmic time. The certificates $[Y_i, Y_{i+1}]$ are treated similarly.

The definition and details of $\mathcal{D}(p_r)$ are as follows. First consider the horizontal line h_r passing through a red point p_r and let $\mathrm{Red}(h_r)$ be the set of red points lying below h_r (see Figure 2). Denote as $x(p)$ (resp. $y(p)$) the abscissa (resp. ordinate) of p. For each $p \in \mathrm{Red}(h_r)$ we define

- $\mathcal{V}(p)$ as the vertical half-line for which p is its top-most point.
- $\mathcal{I}(p)$ as the maximum-length horizontal segment that contains p and does not intersect any other $\mathcal{V}(q)$ for q in $\mathrm{Red}(h_r) \setminus \{p\}$.
- $\mathrm{Blue}(s)$ as the set of blue points that lie in the rectangle whose bottom side is the horizontal segment s and its top side lies on h_r.
- $left(p)$ (resp. $right(p)$) as the rightmost (resp. leftmost) point in $\mathrm{Red}(h_r)$ located to the left (resp. right) of the vertical line that passes through p and whose y coordinate is greater than $y(p)$ (see Figure 2). Notice that the end points of $\mathcal{I}(p)$ lie on $\mathcal{V}(left(p))$ and $\mathcal{V}(right(p))$.
- the set $candidates(p_r)$ of the points $p \in \mathrm{Red}(h_r)$ such that $\mathcal{I}(p)$ intersects the vertical line that passes through p_r. Observe that, according to the previous section, $H(p_r)$ is the box whose top side lies on h_r and its bottom side is the interval $\mathcal{I}(p)$ such that $p \in candidates(p_r)$ and $|\mathrm{Blue}(\mathcal{I}(p))|$ is maximized.

The key idea for $D(p_r)$ is to dynamically compute $H(p_r)$ when two points that lie below h_r make a flip. $\mathcal{D}(p_r)$ is designed as follows.

1. The elements of $candidates(p_r)$ are stored in the leaves of a dynamic balanced binary search tree $\mathcal{Q}$ in decreasing order of ordinate. Every node v is labeled with a such that $|\mathrm{Blue}(\mathcal{I}(p))|$ is equal to the sum of the a labels of the nodes in the unique simple path from the leaf that represents p to the root. In addition, every internal node v is labeled with b whose value is the candidate p in the subtree rooted at v that maximizes $|\mathrm{Blue}(\mathcal{I}(p))|$. With this, the label b of the root is the candidate red point that determines $H(p_r)$.
2. Let $\mathcal{T}_R$ be a balanced binary search tree whose elements are the elements of $\mathrm{Red}(h_r)$ in increasing order of abscissa. Every node v is labeled with y_{max}

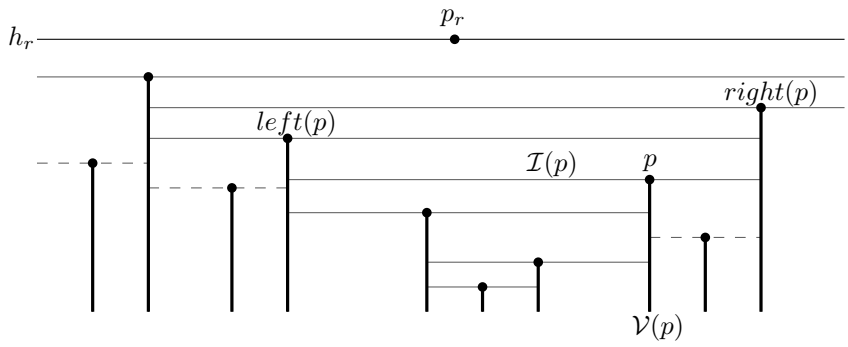

Figure 2: $\mathcal{D}(p_r)$. Candidate points are those whose $\mathcal{I}(p)$ is drawn with a continuous line.

that is the element that has the maximum ordinate in the subtree rooted at v. It permits us to obtain in logarithmic time $left(p)$ and $right(p)$ for any $p \in \text{Red}(h_r)$.

3. For each $p \in \text{Red}(h_r)$ we divide $\mathcal{I}(p)$ at p obtaining the horizontal segments $\mathcal{I}_{left}(p)$ and $\mathcal{I}_{right}(p)$ (see Figure 3). Let $\mathcal{T}_{left}(p) = \{\mathcal{I}_{right}(q) \mid q \in \text{Red}(h_r) \setminus \{p\} \wedge right(q) = p\}$ and $\mathcal{T}_{right}(p) = \{\mathcal{I}_{left}(q) \mid q \in \text{Red}(h_r) \setminus \{p\} \wedge left(q) = p\}$. The set $\mathcal{T}_{left}(p)$ is represented in a dynamic balanced binary search tree where its elements $\mathcal{I}_{right}(q)$ are stored at the leaves in decreasing order of $y(q)$. Every node v is labeled with a in a similar way as we did for $\mathcal{Q}$. In this case $|\text{Blue}(\mathcal{I}_{right}(q))|$ is the sum of the a labels of the nodes in the path from the leaf that represents $\mathcal{I}_{right}(q)$ to the root. Also, every node v is labeled with the boolean c indicating if all the elements $\mathcal{I}_{right}(q)$ in the subtree rooted at v satisfy that q is a candidate red point. $\mathcal{T}_{left}(p)$ is designed to support the operators *join* and *split* both in logarithmic time [5]. The set $\mathcal{T}_{right}(p)$ is represented in the same manner.

4. The set $\{\mathcal{I}(p) \mid p \in \text{Red}(h_r)\} \cup \{\mathcal{V}(p) \mid p \in \text{Red}(h_r)\}$ determines a partition $\mathcal{P}$ of the lower half-plane of h_r. For each cell $\mathcal{C}$ of $\mathcal{P}$ we store in a balanced binary search tree $\mathcal{T}_B(\mathcal{C})$ (supporting operators *join* and *split*) the elements of $B \cap \mathcal{C}$ in decreasing order of ordinate. Each node v is labeled with the count of blue points in the subtree rooted at v in such a way the label of the root is $|B \cap \mathcal{C}|$. Each blue point p below h_r has a reference $\mathcal{T}_{\mathcal{C}}(p)$ to the tree $\mathcal{T}_B(\mathcal{C})$ where $\mathcal{C}$ is the cell that contains p. Also, we associate to each $p \in \text{Red}(h_r)$ the tree $\mathcal{T}_B(p)$, which is the tree of blue points that corresponds to the cell of $\mathcal{P}$ that is bounded below by $\mathcal{I}(p)$, and the tree $\mathcal{T}_{B_{right}}(p)$ that corresponds to the cell of $\mathcal{P}$ bounded by $\mathcal{V}(p)$ and $\mathcal{V}(q)$ and has no bottom boundary, where q is the element that follows p on $\mathcal{T}_R$. (see Figure 3).

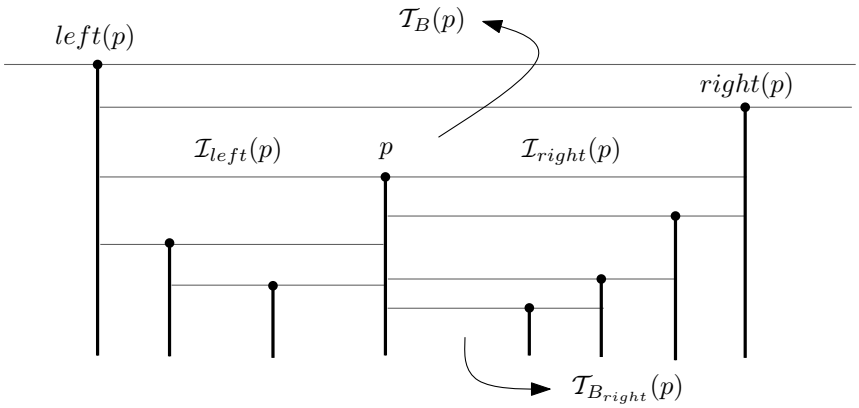

Figure 3: Details of $\mathcal{D}(p_r)$.

Because of the similarity between the structure of $\mathcal{D}(p_r)$ and the operations of the algorithm of Section 2, a similar algorithm to build $\mathcal{D}(p_r)$ can be designed. We establish the following result,

Theorem 3.1 *Given R, B and p_r such that $|R| = r$, $|B| = b$ and $p_r \in R$, $\mathcal{D}(p_r)$ requires $O(r + b)$ space and it can be built in $O(r \log r + r \log b + b \log b)$ time.*

Proof. Both $\mathcal{Q}$ and $\mathcal{T}_R$ have $O(r)$ space complexity. The total complexity of $\mathcal{I}_{left}(p)$ and $\mathcal{I}_{right}(p)$ for all $p \in \mathrm{Red}(h_r)$ is $O(2|\,\mathrm{Red}(h_r)|) = O(r)$. The total space required by $\mathcal{T}_B(\mathcal{C})$ for all cell $\mathcal{C}$ of $\mathcal{P}$ is $O(\sum_{\mathcal{C}\in\mathcal{P}} |B \cap C|) = O(|B \cap (\bigcup_{\mathcal{C}\in\mathcal{P}} \mathcal{C})|) = O(b)$. Then the space complexity follows.

The construction of $\mathcal{D}(p_r)$ can be done in $O(r \log r + r \log b + b \log b)$ time by making the following changes in the procedure presented in Section 2 for the static case. Process all points p below (h_r) by inserting P in T_R or T_B according to its color and, if p is a red point, then perform the following steps. Obtain $left(p)$ and $right(p)$, compute $|\,\mathrm{Blue}(\mathcal{I}(p))|$, $|\,\mathrm{Blue}(\mathcal{I}_{left}(p))|$ and $|\,\mathrm{Blue}(\mathcal{I}_{right}(p))|$ by querying T_B and insert $\mathcal{I}_{left}(p)$ and $\mathcal{I}_{right}(p)$ in $\mathcal{T}_{right}(left(p))$ and $\mathcal{T}_{left}(right(p))$ respectively. Construct $\mathcal{T}_B(p)$ with the blue points that lie in the isothetic box whose bottom side is $\mathcal{I}(p)$ and one of its top vertices is the point with minimum ordinate between $left(p)$ and $right(p)$. At the end construct $\mathcal{T}_{B_{right}}(p)$ for all red point p lying below h_r. All the operations described above are done in logarithmic time each except the construction of $\mathcal{T}_B(p)$ and $\mathcal{T}_{B_{right}}(p)$ that are done by using efficient *Range Search Techniques* over the elements of B in $O(\log b + k)$ where k is the number of reported blue points [3]. Each blue point below h_r is reported once in the overall process and the time complexity follows. □

Corollary 1 *Given R and B such that $|R| = r$ and $|B| = b$, the data structure $\mathcal{D}$ has space complexity $O(r^2 + rb)$ and it can be built in $O(r^2 \log r + r^2 \log b + rb \log b)$ time.*

In the following we show all the possible types of flips that can occur when the elements of $R \cup B$ move. The details on how to process any of them depend on its type and are given in the full version. We say that a flip is *horizontal* (resp. *vertical*) when the two involved points change their y-order (resp. x-order).

Flip type (A). Two blue points make a vertical flip. Nothing to do.

Flip type (B). Two blue points b_1 and b_2 make a horizontal flip. Exchange b_1 and b_2 in every tree of blue points $\mathcal{T}_B(.)$ or $\mathcal{T}_{B_{right}}(.)$ that contains both.

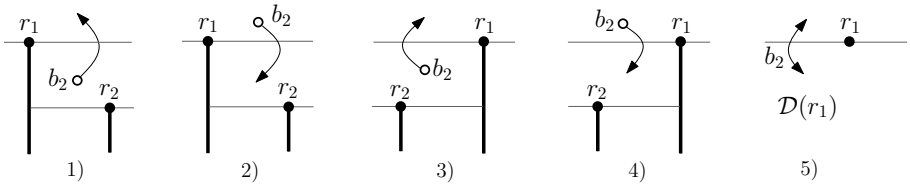

Figure 4: Horizontal flip cases between a red point r_1 and a blue point b_2.

Flip type (C). A red point r_1 and a blue point b_2 make a horizontal flip. Consider five cases as depicted in Figure 4. The procedure below *Horizontal-Flip-Red-Blue-1* solves case 1). Note that case 5) occurs only in one structure $\mathcal{D}(\cdot)$ and it can be processed by rebuilding $\mathcal{D}(\cdot)$. Procedures for the other cases are similar.

Algorithm 1 Horizontal-Flip-Red-Blue-1(r_1, b_2)

Rebuild $\mathcal{D}(r_1)$
for all $p_r \in R$ such that $r_1, b_2 \in \mathcal{D}(p_r)$ **do**
  Find the leaf representing $\mathcal{I}_{right}(r_1)$ in $\mathcal{T}_{left}(right(r_1))$ and add $+1$ to its a label.
  if r_1 is a candidate point **then**
    Find the leaf w that represents r_1 in $\mathcal{Q}$ and add $+1$ to its a label. Make the update of the b labels from w to the root. $H(p_r)$ is now determined by the b label of the root.
  end if
  Let $T = \mathcal{T}_B(r_2)$ if r_2 exists, otherwise $T = \mathcal{T}_{B_{right}}(r_1)$
  Pass b_2 from T to $\mathcal{T}_B(r_1)$
end for

Flip type (D). A red point r_1 and a blue point b_2 make a vertical flip. Consider two cases as depicted in Figure 5, and their symmetric cases.

Flip type (E). Two red points r_1 and r_2 make any flip. We consider two cases as depicted in Figure 6, and their symmetric cases. We use the operators *join* and *split* [5] for vertical flips. The procedure for solving case 2 is as follows:

Each $\mathcal{D}(p_r)$ is updated in $O(\log r + \log b)$ time, then it implies an $O(r \log r + r \log b)$-time update for $\mathcal{D}$. The basic operations of the procedures for solving the flip cases are done in logarithmic time each. Each of the trees that are used

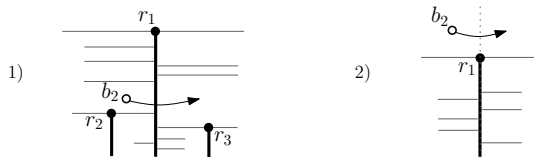

Figure 5: Two cases of vertical flip between a red point r_1 and a blue point b_2.

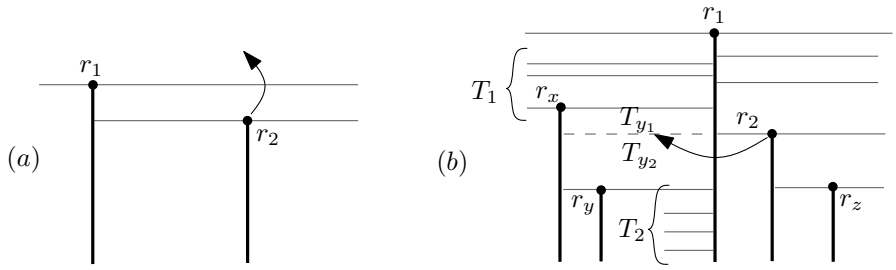

Figure 6: Flipping two red points r_1 and r_2. (a) Horizontal flip. (b) Vertical flip.

in $\mathcal{D}(p_r)$ can be implemented by using *Red-Black Trees* where all the elements are stored in the leaves [5]. We obtain the following result.

Theorem 3.2 *Given a set of moving red points R and a set of moving blue points B such that $|R| = r$ and $|B| = b$, whenever two points in $R \cup B$ make a flip, the data structure MBKDS can be updated in $O(r \log r + r \log b)$ time.*

4 The Arbitrarily Oriented Maximum Box Problem

In this section we consider, as an application of our method for the dynamic case, the problem of finding the Maximum Box for a static set $R \cup B$ in the plane, when the box can be oriented according with any angle (direction) in $[0, \pi]$.

It is easy to see that there are $O((r+b)^2)$ critical directions and the method of Section 2 can be applied for each of them. With this we obtain an $O((r+b)^2(r^2 \log r + r^2 \log b + rb \log b))$-time algorithm. However, we can do it better by iterating all the critical directions and computing dynamically the Maximum Box per each. Start with the direction given by the angle $\theta = 0$. Compute the isothetic Maximum Box and the data structure $\mathcal{D}$. Then make a rotational sweeping of the coordinate axis maintaining the x-order and the y-order of the elements of $R \cup B$. The x-order (resp. y-order) changes whenever two points are at the same distance to the y-axis (resp. x-axis), implying the swap (*flip*) of two consecutive elements and the appearance of a new critical orientation θ.

Algorithm 2 Vertical-Flip-Red-Red-2(r_1, r_2)

Rebuild $\mathcal{D}(r_1)$
for all $p_r \in R$ such that $r_1, r_2 \in \mathcal{D}(p_r)$ **do**
Define r_x, r_y and r_z as depicted in Figure 6 (b). Note that $left(r_y) = r_x$, $right(r_x) = right(r_y) = r_1$ and $left(r_z) = r_2$.

Let $T_z = \mathcal{T}_B(r_z)$ if r_z exists, otherwise $T_z = \mathcal{T}_{B_{right}}(r_2)$
Set $T_z = \text{JOIN}(T_z, \mathcal{T}_B(r_2))$

If r_x exists let $v_1 = |\,\text{Blue}(\mathcal{I}_{right}(r_x))|$, otherwise let $v_1 = |\,\text{Blue}(\mathcal{I}_{left}(r_1))|$ and $r_x = left(r_1)$
Let $T_y = \mathcal{T}_B(r_y)$ if r_y exists, otherwise $T_y = \mathcal{T}_{B_{right}}(r_x)$
Apply SPLIT on T_y by using $y(r_2)$ to obtain the trees T_{y_1} and T_{y_2} as shown in Figure 6 and set $T_y = T_{y_2}$ and $\mathcal{T}_B(r_2) = T_{y_1}$.

Apply SPLIT on $\mathcal{T}_{left}(r_1)$ by using $y(r_2)$ to obtain the trees T_1 and T_2 as shown in Figure 6.
Set $\mathcal{T}_{left}(r_1) = T_1$. Remove $\mathcal{I}_{right}(r_2)$ from $\mathcal{T}_{left}(right(r_2))$ and insert it in $\mathcal{T}_{left}(r_1)$ such that $|\,\text{Blue}(\mathcal{I}_{right}(r_2))| = 0$.
Set $\mathcal{T}_{right}(r_1) = \text{JOIN}(\mathcal{T}_{right}(r_1), \mathcal{T}_{right}(r_2))$, $\mathcal{T}_{left}(r_2) = T_2$ and $\mathcal{T}_{right}(r_2) = null$.
Remove $\mathcal{I}_{left}(r_2)$ from $\mathcal{T}_{right}(r_1)$ and insert it in $\mathcal{T}_{right}(r_x)$ ensuring that $|\,\text{Blue}(\mathcal{I}_{left}(r_2))| = v_1 +$ the number of elements in T_{y_1}

if r_2 is candidate and it is not after the flip **then**
Remove r_2 from $\mathcal{Q}$
else
Insert r_2 in $\mathcal{Q}$ such that $|\,\text{Blue}(\mathcal{I}(r_2))| = |\,\text{Blue}(\mathcal{I}_{left}(r_2))|$
end if
Swap r_1 and r_2 in $\mathcal{T}_R$
end for

Thus, the Maximum Box that is oriented according to θ is computed by making the $O(r \log r + r \log b)$-time update in $\mathcal{D}$. The overall rotation angle is at most π radians and we establish the following result.

Theorem 4.1 *Given a set of red points R and a set of blue points B in the plane such that $|R| = r$ and $|B| = b$, the Arbitrarily Oriented Maximum Box Problem can be solved in $O((r+b)^2(r \log r + r \log b))$ time and $O(r^2 + rb)$ space.*

5 An Approximation Algorithm

In this section we study how to maintain an approximation of the maximum box (the number of of blue points is approximated) using sub-quadratic space. For a constant $c \in (0, 1)$, a *c-approximation of the maximum box* is a box containing no red points and at least cb^* blue points where b^* is the number of blue points in the maximum box. An $O(n \log^2 n)$-time algorithm to compute a $\frac{1}{2}$-approximation of the maximum box in the static setting is presented in [9]. The key idea is as follows: take the median point p of the x-order and compute the exact maximum box H_p that has p on either its left side or its right side. Then, return the best box between H_p and the ones that are returned by the recursive call to the sets $\{q \in R \cup B | x(q) < x(p)\}$ and $\{q \in R \cup B | x(q) > x(p)\}$.

We use a similar procedure to extent MBKDS to a new KDS named as Apx-MBKDS. First of all, we study the dynamic operations on $\mathcal{D}(p_r)$, that is, how to perform efficiently insertions an deletions of red an blue points on it. After that, we consider a new data structure $\mathcal{D}^*(p_r)$ as an extension of $\mathcal{D}(p_r)$ and it is used with a binary tree that has the points of R sorted by $y(\cdot)$.

5.1 Dynamic operations in $\mathcal{D}(p_r)$

In this subsection we prove that inserting or deleting either a red point or a blue point in $\mathcal{D}(p_r)$ can be done in $O(r + b)$ time. Both actions are based on the following primitives.

1. *GetPriorities*: Given any $\mathcal{D}(p_r)$ calculates $|\operatorname{Blue}(\mathcal{I}(p))|$ for all the red points $p \in \mathcal{D}(p_r)$. It takes all such p and applies a top-bottom traversal in $\mathcal{T}_{left}(p)$ and in $\mathcal{T}_{right}(p)$ to obtain, by using the a-marks, $|\operatorname{Blue}(\mathcal{I}_{right}(q))|$ if $right(q) = p$ or $|\operatorname{Blue}(\mathcal{I}_{left}(q))|$ if $left(q) = p$.

2. *IntersectLists*(v): Given any $\mathcal{D}(p_r)$ and a point v returns two lists L_{left} and L_{rigth} such that: *i)* L_{left} (*resp.* L_{rigth}) contains the red points $q \in \mathcal{D}(p_r)$ such that $\mathcal{I}(q)$ intersects the vertical half-line emanated from v downwards and $x(q) < x(v)$ (*resp.* $x(v) < x(q)$) *ii)* the elements in L_{left} and L_{rigth} are sorted in decreasing order of $y(\cdot)$. It can be done in linear time as follows: Denote by l_v the vertical half-line emanated from v downwards and set L_{left} and L_{rigth} to two empty lists. Iterate the red points $q \in \mathcal{D}(p_r)$ by traversing the leaves of $\mathcal{T}_R$ from left to right. If $\mathcal{I}(q)$ intersects l_v then insert q at the end of L_{left} if $x(q) < x(v)$, otherwise insert it at the beginning of L_{rigth}.

3. *RebuildCandidates*: This is the operation that given $\mathcal{D}(p_r)$ rebuilds in linear time the set $\mathcal{Q}$ associated to $\mathcal{D}(p_r)$. First, apply *GetPriorities* and obtain the candidates red points by merging on a list L the two lists returned by invoking $IntersectLists(p_r)$. After that, apply a bottom-up building of $\mathcal{Q}$ from L.

By using above primitives insertions and deletions can be done as follows. Inserting a blue point v in $\mathcal{D}(p_r)$ can be done in linear time: First, find the cell $\mathcal{C}$ of the partition $\mathcal{P}$ determined by $\mathcal{D}(p_r)$ and insert v in $\mathcal{T}_B(\mathcal{C})$. After, in a naive way: Apply *GetPriorities* and, for all red point $q \in \mathcal{D}(p_r)$ increment $|\,\mathrm{Blue}(\mathcal{I}_{left}(q))|$ (*resp.* $|\,\mathrm{Blue}(\mathcal{I}_{right}(q))|$) in one if v is above $\mathcal{I}_{left}(q)$ (*resp.* $\mathcal{I}_{right}(q)$). At the end invoke *RebuildCandidates*.

Inserting a red point v requires a different approach.

(1) Apply *GetPriorities* to obtain for all $p \in R \cup B$ $|\,\mathrm{Blue}(\mathcal{I}_{left}(p))|$ and $|\,\mathrm{Blue}(\mathcal{I}_{right}(p))|$ and $IntersectLists(v)$ to get the two lists of red points $L_{left} = \{l_1, l_2, \ldots, l_{k_1}\}$ and $L_{right} = \{r_1, r_2, \ldots, r_{k_2}\}$ where l_i $(1 \leq i \leq k_1)$ and r_j $(1 \leq j \leq k_2)$ are the left and right red points as depicted in Figure 7.

(2) Include v in $\mathcal{D}(p_r)$ by inserting v in $\mathcal{T}_R$. This implies the calculation of $|\,\mathrm{Blue}(\mathcal{I}_{left}(v))|$ and $|\,\mathrm{Blue}(\mathcal{I}_{right}(v))|$ to insert $\mathcal{I}_{left}(v)$ and $\mathcal{I}_{right}(v)$ in the trees $\mathcal{T}_{right}(left(v))$ and $\mathcal{T}_{left}(right(v))$ respectively.

(3) Suppose now w.l.o.g. that $Y(l_1) > Y(r_1)$. Determine for all red point p in L_{left} or L_{rigth} $(p \neq l_1)$ the lists of blue points $L_1(p)$ and $L_2(p)$ whose members are sorted by Y in a decreasing way and $L_1(p)$ (resp. $L_2(p)$) contains the blue points of $\mathcal{T}_B(p)$ such that $x(p) < x(v)$ (resp. $x(v) < x(p)$).

(4) For the new cells $\mathcal{C}_v$, $\mathcal{C}_{l_1}$ and $\mathcal{C}_{r_1}$, whose bottom sides are $\mathcal{I}(v)$, $\mathcal{I}(l_1)$ and $\mathcal{I}(r_1)$ respectively, compute the corresponding trees $\mathcal{T}_B(v) = \mathcal{T}_B(\mathcal{C}_v)$, $\mathcal{T}_B(l_1) = \mathcal{T}_B(\mathcal{C}_{l_1})$ and $\mathcal{T}_B(r_1) = \mathcal{T}_B(\mathcal{C}_{r_1})$.

(5) The elements of L_{left} are the left neighbors of $\mathcal{V}(v)$ and to build $\mathcal{T}_{left}(v)$ we have to compute $|\,\mathrm{Blue}(\mathcal{I}_{right}(l_i))|$ for all l_i in L_{left}. It can be done as follows: calculate first $|\,\mathrm{Blue}(\mathcal{I}_{right}(l_1))|$, and after for each $2 \leq i \leq k_1$ build $\mathcal{T}_B(l_i)$ from the concatenation of the lists $L_1(r_j)$ where $Y(l_{i-1}) < Y(r_j) < Y(l_i)$, then $|\,\mathrm{Blue}(\mathcal{I}_{right}(l_i))|$ is calculated by using the following equation:

$$|\,\mathrm{Blue}(\mathcal{I}_{right}(l_i))| = |\,\mathrm{Blue}(\mathcal{I}_{right}(l_{i-1}))| - |\,\mathrm{Blue}(\mathcal{I}_{left}(l_i))| + |\mathcal{T}_B(l_i)|$$

where $|\mathcal{T}_B(l_i)|$ is the number of elements in $\mathcal{T}_B(l_i)$. Proceed similarly to build $\mathcal{T}_{right}(v)$ from L_{right}.

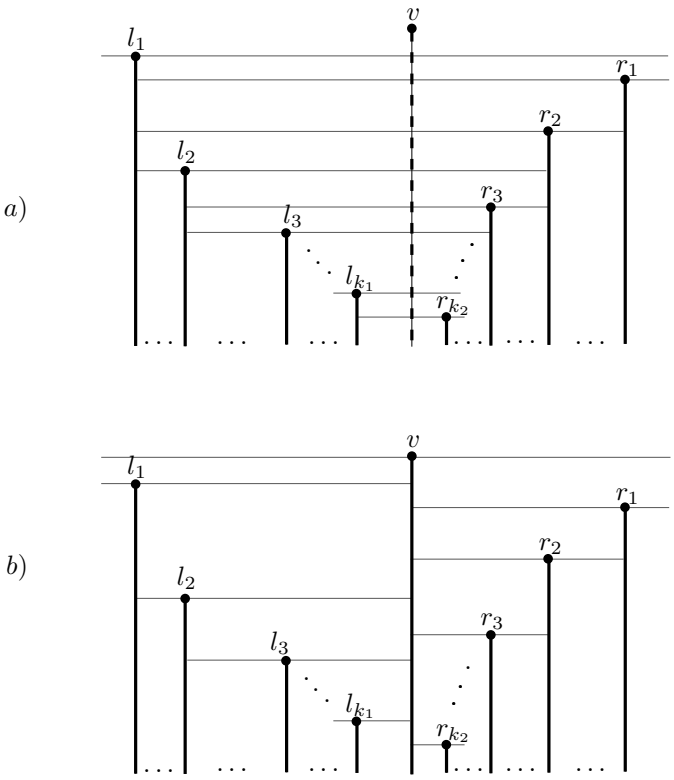

Figure 7: Inserting a red point v in $\mathcal{D}(p_r)$. a) Before insertion, b) After insertion.

(6) Compute $T_{B_{right}}(l_{k_1})$ and $T_{B_{right}}(v)$ and apply *RebuildCandidates*.

The deletion can be done similar to the insertion. It is easy to show that both operations spend $O(r+b)$ time thus we obtain the following result:

Theorem 5.1 *Inserting or deleting either a red point or a blue point in the data structure $\mathcal{D}(p_r)$ can be done in $O(r+b)$ time.*

5.2 Apx-MBKDS

In this section we extend or MBKDS to support the maintenance of the $\frac{1}{2}$-approximation of the maximum box. For this, consider the extended version $\mathcal{D}^*(p_r)$ of $\mathcal{D}(p_r)$ that takes care the points that are not only below the line h_r that passes through p_r but also above it. Those points above h_r are structured symmetrically to the points that lie below (see Figure 8 (a)). Now suppose using in the $\frac{1}{2}$-approximation of the maximum box algorithm presented in [9] the median point of the y-order instead of the x-order one's. Then our Apx-MBKDS is a Two-Level data structure that is composed in the first level by

a balanced binary search tree $T_{1/2}$ having the elements of R sorted by Y and in the second level by instances of $\mathcal{D}^*(.)$. Refer to Figure 8 (b). The recursive definition is as follows:

Given R and B, let p_r be the median point of R in the y-order. The root node v stores p_r and $\mathcal{D}^*(v) = \mathcal{D}^*(p_r)$ and it is built from R and B. The left child of v is recursively constructed from the sets $R_l = \{q \in R \mid y(q) < y(p_r)\}$ and $B_l = \{q \in B \mid y(q) < y(p_r)\}$. Similarly, the right child from $R_r = \{q \in R \mid y(p_r) < y(q)\}$ and $B_r = \{q \in B \mid y(p_r) < y(q)\}$.

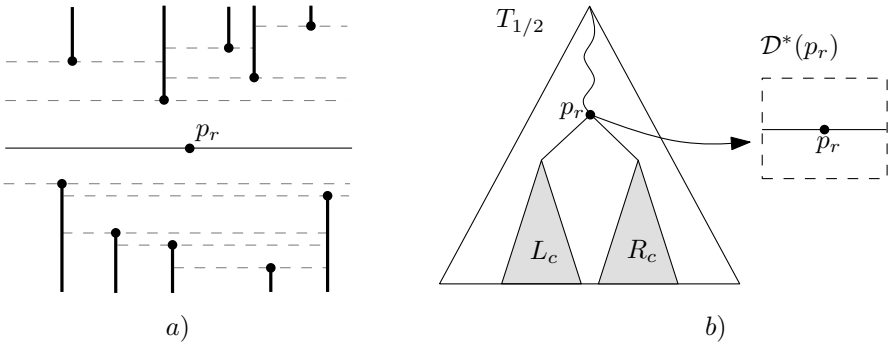

Figure 8: a) $\mathcal{D}^*(p_r)$, b) The Two-Level Data Structure for the approximation.

Additionally we associate to every node v of $T_{1/2}$ the best maximum box $H(v)$ of the elements of the subtree rooted at v. It is the best box between the solution provided by $\mathcal{D}^*(v)$ and $H(lc(v))$ and $H(rc(v))$ where $lc(v)$ and $rc(v)$ are the left and right child of v respectively. The box $H(root)$ of the root node $root$ is the $\frac{1}{2}$-approximation of the maximum box.

It is easy to show that Apx-MBKDS uses $O((r+b)\log r)$ space and can be built in $O((r+b)\log^2(r+b))$ time. Now we describe the processing of flips.

Processing the horizontal flip between two blue points p_1 and p_2 or the vertical flip between a red point p_1 and a blue point p_2 can be done as follows: Let v be the node in $T_{1/2}$ where the search paths of p_1 and p_2 splits. Apply the $O(\log r + \log b)$-time update in $\mathcal{D}^*(w)$ for every node w in the path from the parent of v to the root. Thus this operation takes $O(\log^2 r + \log r \log b)$ time.

The horizontal flip between a red point p_1 and a blue point p_2 can be processed as above but in addition: Let v be the node that stores p_1, update $\mathcal{D}^*(v)$ by applying the insertion/deletion method of a blue point described in the previous subsection and update $\mathcal{D}^*(w)$ in every node w in the path from v to the predecessor (resp. successor) of v in $T_{1/2}$. This process spends $O(r+b)$ time. The horizontal flip between two red points p_1 and p_2 can be processed similarly but adding the following: Let v_1 and v_2 be the nodes that store p_1 and p_2 respectively and suppose w.l.o.g. that v_2 belongs to the subtree rooted at v_1. Update $\mathcal{D}^*(v_1)$ by applying the insertion/deletion method of a red point in $\mathcal{D}(.)$. We obtain the main result of this section.

Theorem 5.2 *Apx-MBKDS is a compact and local kinetic data structure for*

maintaining the $\frac{1}{2}$-approximation of the maximum box over a set of moving points. It uses $O((r+b)\log r)$ memory and can be built in $O((r+b)\log^2(r+b))$ time. The events can be processed in $O(r+b)$ time in the worst case.

References

[1] B. Aronov and S. Har-Peled. On approximating the depth and related problems. SIAM J. Comput. Vol. 38, 899–921, 2008.

[2] P. Bonnet, J. Gehrke, P. Seshadri. Towards Sensor Database Systems. In Proc. of the Second International Conf. on Mobile Data Management, Hong Kong. Lecture Notes Comp. Sci., vol. 1987, 3 – 14, 2001.

[3] B. Chazelle and L. J. Guibas. Fractional cascading: A data structuring technique, Algorithmica, 1 , 133–162, 1986.

[4] A. Civilis, C. S. Jensen, S. Pakalnis. Techniques for Efficient Road-Network-Based Tracking of Moving Objects. IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 5, 698–712, 2005.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*, Second Edition. MIT Press, McGraw-Hill, 2001.

[6] C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, I. Ventura. Bichromatic separability with two boxes: a general approach. *Journal of Algorithms. Cognition, Informatics and Logic.* Vol. 64, No. 2–3, pp. 79–88, 2009.

[7] R. Duda, P. Hart, D. Stork. *Pattern Classification. John Wiley and Sons, Inc.*, New York, 2001.

[8] L.J. Guibas. Kinetic Data Structures: A State of the Art Report. In *Robotics: The Algorithmic Perspective*. A. K. Peters, Ltd 191–209, 1998.

[9] Y. Liu, M. Nediak. Planar Case of the Maximum Box and Related Problems. In Proc. Canad. Conf. Comp. Geom., Halifax, Nova Scotia, August 11–13, 2003.

[10] M. Segal. Planar Maximum Box Problem. Journal of Mathematical Modeling and Algortihms. 3, 31–38, 2004.