Adaptive Techniques to find Optimal Planar Boxes

J. Barbay [*] G. Navarro [†] P. Pérez-Lantero [‡]

Abstract

Given a set P of n planar points, two axes and a real-valued score function $f()$ on subsets of P, the Optimal Planar Box problem consists in finding a box (i.e. an axis-aligned rectangle) H maximizing $f(H \cap P)$. We consider the case where $f()$ is monotone decomposable, i.e. there exists a composition function $g()$ monotone in its two arguments such that $f(A) = g(f(A_1), f(A_2))$ for every subset $A \subseteq P$ and every partition $\{A_1, A_2\}$ of A. In this context we propose a solution for the Optimal Planar Box problem which performs in the worst case $O(n^2 \lg n)$ score compositions and coordinate comparisons, and much less on other classes of instances defined by various measures of difficulty. A side result of its own interest is a fully dynamic *MCS Splay tree* data structure supporting insertions and deletions with the *dynamic finger* property, improving upon previous results [Cortés et al., J.Alg. 2009].

1 Introduction

Consider a set P of n planar points, and two axes x and y forming a base of the plane, such that the points are in general position (i.e. no pair of points share the same x or y coordinate). We say that a real-valued function $f()$ on subsets of P is *decomposable* [2, 7] if there exists a *composition function* $g()$ such that $f(A) = g(f(A_1), f(A_2))$ for every subset $A \subseteq P$ and every partition $\{A_1, A_2\}$ of A. Without loss of generality, we extend $f()$ to P such that $f(p) = f(\{p\})$. A decomposable function is *monotone* if the corresponding composition function $g()$ is monotone in its two arguments. A *box* is a rectangle aligned to the axes, and given a monotone decomposable function $f()$, such a box is $f()$-*optimal* if it optimizes $f(H \cap P)$. Without loss of generality, we assume that we want to maximize $f()$ and that its composition function $g()$ is monotone increasing in its two arguments. Given a monotone decomposable function $f()$ well defined for the empty set $\varnothing$, a point p of P is *positive* if $f(p) > f(\varnothing)$. Otherwise, this point p is *negative*. Observe that if p is positive then $f(A \cup \{p\}) = g(f(A), f(p)) > g(f(A), f(\varnothing)) = f(A)$ by monotonicity of $g()$: hence a point p is positive if and only if $f(A \cup \{p\}) > f(A)$ for every subset $A \subset P$ not containing p. A *stripe* is an area delimited by two lines parallel to the same axis. A *positive stripe* (resp. *negative stripe*) is one which contains only positive (resp. negative) points. A *monochromatic stripe* is a stripe in which all points have the same sign.

Given a set of planar points, a simple example of such monotone decomposable functions is counting the number of points the box contains. Other examples include counting the number of blue points; returning the difference between the number of blue points and the number of red points contained; returning the number of blue points in the box or $-\infty$ if it contains some red points; summing the weights of the points contained; taking the maximum of the weights of contained points; etc.

Given a set P of n planar points and a real-valued function $f()$ on subsets of P, the Optimal Planar Box problem consists in finding an $f()$-optimal box. Depending on $f()$, this problem has various practical applications, from identifying rectangular areas of interest in astronomical pictures to the design of optimal rectangular forest cuts or the analysis of medical radiographies. We present various adaptive techniques for the Optimal Planar Box problem:

- In the worst case over instances composed of n points, our algorithm properly generalizes Cortés et al.'s solution [5] for the Maximum Weight Box problem, within the same complexity of $O(n^2 \lg n)$ score compositions.
- For any $\delta \in [1..n]$ and $n_1, \ldots, n_\delta \in [1..n]$ summing to n, in the worst case over instances composed of δ monochromatic stripes of alternating signs when the points are sorted by their y-coordinates, such that the i-th stripe contains n_i points, our algorithm executes $O(\delta n(1 + \mathcal{H}(n_1, \ldots, n_\delta))) \subset O(\delta n \lg(\delta + 1))$ score compositions (Theorem 4), where $\mathcal{H}(n_1, \ldots, n_\delta) = \sum_{i=1}^{\delta} (n_i/n) \lg(n/n_i)$ is the usual entropy function.
- Assuming the same y-coordinate order, for any $\lambda \in [0..n^2]$, in the worst case over instances where

[*]Department of Computer Science, University of Chile, Chile, jeremy.barbay@dcc.uchile.cl. Partially funded by grant FONDECYT 1-120054, Chile.

[†]Department of Computer Science, University of Chile, Chile, gnavarro@dcc.uchile.cl. Partially supported by Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.

[‡]Escuela de Ingeniería Civil en Informática, Universidad de Valparaíso, Chile, pablo.perez@uv.cl. Partially supported by grant FONDECYT 11110069 (Chile) and project MEC MTM2009-08652 (Spain).

λ is the sum of the distances between the insertion positions of the consecutive points according to their x-coordinate, our algorithm makes $O(n^2(1+\lg(1+\lambda/n))$ score compositions (Lemma 5). Measure λ relates to the local insertion sort complexity [11] of the sequence of x-coordinates. It holds $\lambda \in O(n + \texttt{Inv})$, where $\texttt{Inv}$ is the number of inversions in the sequence. When the points are grouped into δ monochromatic stripes, the complexity drops to $O(n\delta(1+\lg(1+\texttt{Inv}/n))$ (Theorem 7).

- Assuming the same y-coordinate order, for a minimal cover of the same sequence of x-coordinates into $\rho \leq n$ *runs* (i.e. contiguous increasing subsequences) of lengths $r_1, \ldots, r_\rho$, our algorithm executes $O(n^2(1+\mathcal{H}(r_1,\ldots,r_\rho))) \subset O(n^2\lg(\rho+1))$ score compositions (Lemma 6). When the points can be grouped into δ monochromatic stripes, this complexity decreases to $O(n\delta(1+\mathcal{H}(r_1,\ldots,r_\rho))) \subset O(n\delta\lg(\rho+1))$ (Theorem 7 again).
- Using an approach orthogonal to the one of Cortés et al. [5], we partition (via a clever strategy considering axis-parallel lines) the point set P into subsets called diagonal blocks, so that a new adaptive algorithm is obtained (Theorem 14). The algorithm solves the Optimal Planar Box problem in each block and combine the solutions. Extending this algorithm, we obtain another adaptive algorithm running in $O(n\lg n + \sigma n)$ comparisons and $O(\sigma n \lg n)$ score compositions, where $\sigma \in [1..n]$ is a measure of difficulty of the instance that depends on the partition in diagonal blocks (Theorem 18).

Due to the lack of space, several proofs are omitted. A longer version of this paper is available at `http://arxiv.org/abs/1204.2034`.

2 Optimal Boxes and Related Problems

Given a set P of n weighted planar points, in which the weight of a point can be either positive or negative, the Maximum Weight Box problem [5] consists in finding a box R maximizing the sum of the weights of the points in $R \cap P$. Cortés et al. [5] gave an algorithm solving this problem in time $O(n^2 \lg n)$ using $O(n)$ space, based on *MCS trees*, a data structure supporting in $O(\lg n)$ time the dynamic computation of the Maximum-Sum Consecutive Subsequence problem [3] (hence the name "MCS").

The Maximum Weight Box problem [5] and, by successive reductions, the Maximum Subarray problem [14], the Maximum Box problem [5, 8, 10], and the Maximum Discrepancy Box problem [5, 6] can all be reduced to a finite number of instances of the Optimal Planar Box problem by choosing adequate definitions for the score functions $f()$ to optimize.

Cortés et al.'s algorithm [5] first sorts the points by their y-coordinate in $O(n\lg n)$ time and then traverses the resulting sequence of points $p_1, p_2, \ldots p_n$ as follows. For each p_i, it sets an MCS tree (described in more details in Section 3) with points $p_i, \ldots p_n$, where the key is their x-coordinate x_i, and all have value $f(\varnothing)$. It then successively activates points p_j for $j \in [i..n]$, setting its weight to value $f(p_j)$, updating in time $O(\lg n)$ the MCS tree so that to compute the optimal box contained between the y-coordinate of p_i to that of p_j. The whole algorithm executes in time $O(n^2 \lg n)$.

3 Fully Dynamic MCS Trees

The MCS tree [5] is an index for a fixed sequence $S = (x_i)_{i\in[1..n]}$ of n elements, where each element x_k of S has a weight $w(x_k) \in \mathbb{R}$, so that whenever a weight $w(x_k)$ is updated, a consecutive subsequence $(x_i)_{i\in[l..r]}$ of S maximizing $\sum_{i\in[l..r]} w(x_i)$ is obtained (or recomputed) in $O(\lg n)$ time. This behavior is dynamic in the sense that it allows modification of element weights, yet it is only partially dynamic in the sense that it admits neither addition nor deletion of elements.

Existing dynamic data structures can be easily adapted into a truly dynamic data structure with the same functionalities as MCS trees. We start by generalizing MCS trees [5] from mere additive weights to monotone decomposable score functions in Lemma 1. We further generalize this solution to use an AVL tree [1] in Lemma 2 and a Splay tree [13] in Lemma 3, whose "finger search" property will play an essential role in the results of Sections 4 and 5.

Lemma 1 *Let S be a static sequence of n elements, and $f()$ be a monotone decomposable score function receiving as argument any subsequence of S, defined through the activation and deactivation of each element of S. There exists a semi-dynamic data structure for maintaining S using linear space that supports the search for an element in $O(\lg n)$ comparisons; the activation or deactivation of an element in $O(\lg n)$ score compositions; and $f()$-optimal sub range queries in $O(\lg n)$ comparisons and score compositions.*

The MCS tree data structure can be converted into a truly dynamic data structure supporting both insertions and deletions of elements. This data structure can be used to index a dynamic sequence $S = (x_i)_{i\in[1..n]}$ of n elements so that whenever an element is inserted or removed, a consecutive subsequence $S' = (x_i)_{i\in[l..r]}$ of S optimizing $f(S')$ can be (re)computed in $O(\lg n)$ score compositions and comparisons. The following lemma establishes the property of this data structure, which we call *MCS AVL tree*.

Lemma 2 *Let S be a dynamic sequence of n elements, and $f()$ be a monotone decomposable score function receiving as argument any consecutive subsequence of S. There exists a fully dynamic data structure for maintaining S using linear space that supports the search for an element in $O(\lg n)$ comparisons; the update of the score of an element in $O(\lg n)$ score compositions, the insertion or deletion of an element in $O(\lg n)$ comparisons and score compositions; and $f()$-optimal subrange queries in $O(\lg n)$ comparisons and score compositions.*

The Splay tree is a self-adjusting binary search tree created by Sleator and Tarjan [13]. It supports the basic operations search, insert and delete, all of them called *accesses*, in $O(\lg n)$ amortized time. For many sequences of accesses, splay trees perform better than other search trees, even when the specific pattern of the sequences are unknown. Among other properties of Splay trees, we are particularly interested in the *Dynamic Finger Property*, conjectured by Sleator and Tarjan [13] and proved by Cole et al. [4]: every sequence of m accesses on an arbitrary n-node Splay tree costs $O(m+n+\sum_{j=1}^{m}\lg(d_j+1))$ rotations where, for $j = 1..m$, the j-th and $(j-1)$-th accesses are performed on elements whose ranks among the elements stored in the Splay tree differ by d_j. For $j = 0$, the j-th element is the element stored at the root. It is easy to see that in the MCS AVL tree we can replace the underlying AVL tree by a Splay tree, and obtain then the next lemma, which describes the *MCS Splay tree* data structure.

Lemma 3 *Let S be a dynamic sequence of n elements and $f()$ be a monotone decomposable function receiving as argument any consecutive subsequence of S. There exists a data structure for maintaining S that uses linear space and supports the search in $O(\lg n)$ amortized comparisons, the update of the score of an element in $O(\lg n)$ amortized score compositions, and the insertion and deletion of elements in $O(\lg n)$ amortized comparisons and score compositions. Joint with the insertion or deletion of any element, the consecutive subsequence S' of S maximizing $f(S')$ is recomputed. The Dynamic Finger Property is also satisfied for each operation (search, insertion and deletion), both for the number of comparisons and for the number of score compositions performed.*

4 Taking Advantage of Monochromatic Stripes

Consider an instance where positive and negative points can be clustered into δ positive and negative stripes along one given axis, of cardinalities $n_1, \ldots, n_\delta$. Such stripes can be easily identified in $O(n \lg n)$ comparisons and $O(n)$ score accesses. On such instances one does not need to consider boxes whose borders are in the middle of some stripes: all optimal boxes will start at the edge of a stripe; specifically, the top (resp. bottom) of an optimal box will align with a positive point at the top (resp. bottom) of a positive stripe.

This very simple observation not only limits the number of boxes for which we need to compute a score, but also it makes it easier to compute the score of each box: adding the n_i points of the i-th stripe in increasing order of their coordinates in a MCS Splay tree of final size n amortizes to $O(n+\sum_{i=1}^{\delta} n_i \lg(n/n_i))$ coordinate comparisons and score compositions. The reason is that the n_i distances $d_j + 1$ of Lemma 3 telescope to at most $n + n_i$ within stripe i, and thus by convexity the cost $O(n + \sum_{j=1}^{n}\lg(d_j + 1))$ is upper bounded by $O(n + \sum_{i=1}^{\delta} n_i \lg(1 + n/n_i))$ which is $O(n + \sum_{i=1}^{\delta} n_i \lg(n/n_i)) = O(n(1 + \mathcal{H}(n_1, \ldots, n_\delta))) \subset O(n \lg(\delta + 1))$. Combining this with the fact that the top of an optimal box is aligned with a positive point at the top of a positive stripe yields the following result.

Theorem 4 *For any $\delta \in [1..n]$ and $n_1, \ldots, n_\delta \in [1..n]$ summing to n, in the worst case over instances composed of δ stripes of alternating signs over an axis such that the i-th stripe contains n_i points, there exists an algorithm that finds an $f()$-optimal box in $O(\delta n(1 + \mathcal{H}(n_1, \ldots, n_\delta))) \subset O(\delta n \lg(\delta + 1))$ score compositions and $O(\delta n(1 + \mathcal{H}(n_1, \ldots, n_\delta)) + n \lg n) \subset O(\delta n \lg(\delta + 1) + n \lg n)$ coordinate comparisons.*

5 Taking Advantage of Point Alignments

Running the algorithm outlined in the first paragraph of Section 4 over the MCS Splay tree has further consequences. In this section we show how it makes the algorithm adaptive to local point alignments.

The cost of our algorithm using the MCS Splay tree can be upper bounded as follows. Let λ denote the sum of the distances between the insertion positions of the consecutive points according to their x-coordinate. When we insert the points in the MCS Splay tree starting from p_1, the total cost is $O(n + \sum_{j=1}^{n}\lg(d_j + 1)) \subset O(n + n\lg(1 + \lambda/n))$ score compositions, by convexity of the logarithm and because $\sum_{j=1}^{n} d_j + 1 \le \lambda + n$. A simple upper bound when considering all the n passes of the algorithm can be obtained as follows.

Lemma 5 *There exists an algorithm that finds an $f()$-optimal box in $O(n^2(1+\lg(1+\lambda/n)))$ score compositions and $O(n^2(1 + \lg(1 + \lambda/n)) + n \lg n)$ coordinate comparisons, where $\lambda \le n^2$ is the local insertion complexity of the sequence of x-coordinates of the points sorted by y-coordinates.*

In the worst case this boils down to the $O(n^2 \lg n)$-worst-case algorithm, whereas in the best case $\lambda = 0$ and the cost corresponds to $O(n^2)$ operations.

We can upper bound λ by using other two measures of disorder in permutations. For example, let us consider Inv, the number of *inversions* in the permutation π, or said another way, the number of pairs out of order in the sequence [12]. The measure Inv corresponds to a cost where the "finger" is always at the end of the sequence. This can be as small as $(\lambda - n)/2$, for example consider the permutation $\pi = (m, m-1, m+1, m-2, m+2, \dots, 1, 2m-1)$ for $m = (n+1)/2$ and odd n. However, Inv can be much larger than λ because it is not symmetric on decreasing sequences, for example when the points are semi-aligned in a decreasing diagonal and the permutation is $\pi = (n, n-1, n-2, \dots, 1)$. Thus replacing λ by Inv in Lemma 5 yields a valid upper bound in terms of big-O complexity.

Another well-known measure of permutation complexity is the number of *increasing runs* ρ, that is, the minimum number of contiguous monotone increasing subsequences that cover π [9]. Let $r_1, \dots, r_\rho$ be the lengths of the runs, computed in $O(n \lg n)$ comparisons. Then the sum of the values $|\pi_{j+1} - \pi_j|$ within the i-th run telescopes to at most n, and so does the sum of the d_j values. Therefore $\sum_{j=1}^{n} \lg(d_j + 1) \leq \sum_{i=1}^{\rho} r_i \lg(1+n/r_i) \leq n + \sum_{i=1}^{\rho} r_i \lg(n/r_i)$ by convexity. This leads to the following alternative upper bound.

Lemma 6 *There exists an algorithm that finds an $f()$-optimal box in $O(n \lg n)$ coordinate comparison and $O(n^2(1+\mathcal{H}(r_1, \dots, r_\rho)) \subset O(n^2 \lg(\rho+1))$ score compositions, where $r_1, \dots, r_\rho$ are the lengths of ρ maximal contiguous increasing subsequences that cover the sequence of x-coordinates of the points sorted by y-coordinate.*

6 Taking Advantage of both Stripes and Alignments

The combination of the techniques of Sections 4 and 5 can be elegantly analyzed. A simple result is that we need to start only from δ different p_i values, and therefore an upper bound to our complexity is $O(n\delta((1 + \lg(1 + \lambda/n)))$. We can indeed do slightly better by sorting the points by increasing x-coordinates within each monochromatic stripe. While the measure λ' resulting from this reordering may be larger than λ, the upper bounds related to Inv and ρ, namely Inv$'$, ρ', and $\mathcal{H}(n'_1, \dots, n'_{\rho'})$, do not increase. In particular it is easy to see that the upper bound of Theorem 4 is dominated by the combination since $\rho' \leq \delta$ and $\mathcal{H}(r'_1, \dots, r'_{\rho'}) \leq \mathcal{H}(n_1, \dots, n_\delta)$ (because no run will cut a monochromatic stripe once the latter is reordered).

Theorem 7 *There exists an algorithm that finds an $f()$-optimal box in $O(n \lg n)$ coordinate comparisons and $O(n\delta(1 + \min(\lg(1 + \texttt{Inv}/n), \mathcal{H}(r_1, \dots, r_\rho)))) \subset O(n\delta \lg(\rho + 1))$ score compositions, where δ is the minimum number of monochromatic stripes in which the points, sorted by increasing y-coordinate, can be partitioned; X is the corresponding sequence of x-coordinates once we (re-)sort by increasing x-coordinate the points within each monochromatic stripe; $\texttt{Inv} \leq n^2$ is the number of out-of-order pairs in X; and $r_1, \dots, r_\rho$ are the lengths of the minimum number $\rho \leq \delta$ of contiguous increasing runs that cover X. A similar result holds by exchanging x and y axes.*

Note that if these new measures are not particularly favorable, the formula boils down to the $O(n\delta \lg \delta)$ time complexity of Section 4.

7 Taking Advantage of Diagonals of Blocks

In this section we present an approach orthogonal to the previous ones, which considers partitions of the point set into subsets and yields to a new adaptive algorithm which solves the Optimal Planar Box problem in each of them and combine the solutions. Its difficulty measure depends on such a partition.

For any subset $A \subseteq P$, a *diagonalization* of A is a partition $\{A_1, A_2\}$ of A induced by two lines ℓ_1 and ℓ_2, respectively parallel to axes x and y, so that the elements of A_1 and the elements of A_2 belong to opposite quadrants with respect to the point $\ell_1 \cap \ell_2$. Note that if $p_1, p_2, \dots, p_m$ denote the elements of A sorted by x-coordinate, then any diagonalization of A has the form $\{\{p_1, \dots, p_k\}, \{p_{k+1}, \dots, p_m\}\}$ for some index $k \in [1..m-1]$. Not all point sets admit a diagonalization, the simplest case consists of four points placed at the four corners of a square whose sides are slightly rotated from the axes. We call such a point set a *windmill*, due to the characteristic position of its points. Given any bounded set $S \subset \mathbb{R}^2$, let $\texttt{Box}(S)$ denote the smallest box enclosing S and let the *extreme* points of any subset $A \subseteq P$ be those belonging to the boundary of $\texttt{Box}(A)$.

Lemma 8 *Let A be a point set that does not admit a diagonalization. Then A has exactly four extreme points. Furthermore, A has a windmill which contains at least one extreme point of A.*

Definition 9 *A diagonalization tree of P, D-tree, is a binary tree such that: (i) each leaf u contains a subset $S(u) \subseteq P$ which does not admit a diagonalization, (ii) set $\{S(u) \mid u \text{ is a leaf}\}$ is a partition of P, and (iii) each internal node v has exactly two children v_1 (the left one) and v_2 (the right one) and satisfies that $\{A(v_1), A(v_2)\}$ is a diagonalization of $A(v)$, where for each node v $A(v)$ denotes the union of the sets $S(u)$ for all leaves u descendant of v (See Figure 1).*

Lemma 10 *Let P be a set of n points in the plane. Every D-tree of P has the same number of leaves. Furthermore, the i-th leaves from left to right of any two D-trees of P contain the same subset $S(\cdot)$ of P.*

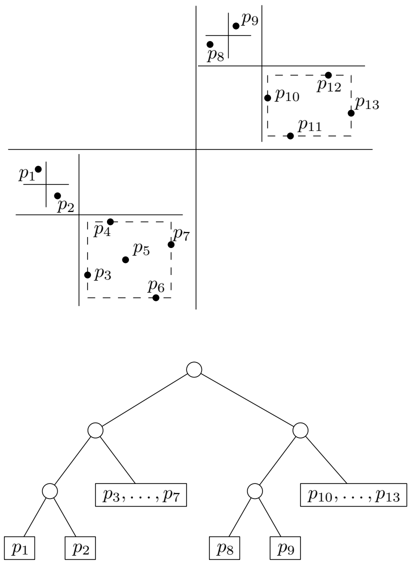

Figure 1: A D-tree of the point set $\{p_1, \ldots, p_{13}\}$.

From Lemma 10 we can conclude that every D-tree T of P induces the same partition $\{S(u_1), \ldots, S(u_\beta)\}$ of P, where $u_1, \ldots, u_\beta$ are the leaf nodes of T.

Lemma 11 *A D-tree of P requires $O(n)$ space and can be built in $O(n \lg n)$ comparisons.*

Proof. (Sketch) Let $p_1, p_2, \ldots, p_n$ be the elements of P sorted by x-coordinate, and let $p_{\pi_1}, p_{\pi_2}, \ldots, p_{\pi_n}$ be the elements of P sorted by y-coordinate. Considering the computation of permutation π a preprocessing, we can show that: If P admits a diagonalization $\{\{p_1, \ldots, p_k\}, \{p_{k+1}, \ldots, p_n\}\}$ then it can be determined in $O(\min\{k, n-k\})$ comparisons. Otherwise, if P does not admit a diagonalization, then it can be decided in $O(n)$ comparisons. We can then build a D-tree of P recursively as follows. If a diagonalization $\{\{p_1, \ldots, p_k\}, \{p_{k+1}, \ldots, p_n\}\}$ of P exists, which was determined in $O(t)$ comparisons where $t = \min\{k, n-k\}$, then create a root node and set as left child a D-tree of $\{p_1, \ldots, p_k\}$ and as right child a D-tree of $\{p_{k+1}, \ldots, p_n\}$. Otherwise, if P does not admit a diagonalization, which was decided in $O(n)$ comparisons, then create a leaf node whose set $S(\cdot)$ is equal to P. This results in the next recurrence equation for the total number $T(n)$ of comparisons, where $1 \leq t \leq \lfloor n/2 \rfloor$:

$$T(n) = \begin{cases} O(t) + T(t) + T(n-t) & n > 1, \text{ a diagonalization exists.} \\ O(n) & \text{otherwise.} \end{cases}$$

By applying induction and using the binary entropy function $H(x) = x \lg(1/x) + (1-x) \lg(1/(1-x))$, with the fact $x \leq H(x)$ for $x \leq 1/2$, it can be proved that $T(n)$ is $O(n \lg n)$. $\square$

Definition 12 *For any non-empty subset $A \subseteq P$ the set of ten $f()$-optimal boxes of A, denoted by $\texttt{Ten}(A)$, consists of the following $f()$-optimal boxes of A, all contained in $\texttt{Box}(A)$:*

1. $\texttt{Box}(A)$;
2. $\texttt{B}_{opt}(A)$, *without restriction;*
3. $\texttt{B}_1(A)$, *with the bottom-left vertex of* $\texttt{Box}(A)$;
4. $\texttt{B}_2(A)$, *with the bottom-right vertex of* $\texttt{Box}(A)$;
5. $\texttt{B}_3(A)$, *with the top-right vertex of* $\texttt{Box}(A)$;
6. $\texttt{B}_4(A)$, *with the top-left vertex of* $\texttt{Box}(A)$;
7. $\texttt{B}_{1,2}(A)$, *with the bottom vertices of* $\texttt{Box}(A)$;
8. $\texttt{B}_{2,3}(A)$, *with the right vertices of* $\texttt{Box}(A)$;
9. $\texttt{B}_{3,4}(A)$, *with the top vertices of* $\texttt{Box}(A)$;
10. $\texttt{B}_{4,1}(A)$, *with the left vertices of* $\texttt{Box}(A)$.

Lemma 13 *For any non-empty subset $A \subseteq P$ and any diagonalization $\{A_1, A_2\}$ of A, $\texttt{Ten}(A)$ can be computed in $O(1)$ score compositions from $\texttt{Ten}(A_1)$ and $\texttt{Ten}(A_2)$.*

Theorem 14 *There exists an algorithm that finds an $f()$-optimal box of P in $O(n \lg n + \sum_{i=1}^{\beta} h_c(n_i))$ comparisons (on coordinates and indices) and $O(\sum_{i=1}^{\beta} h_s(n_i) + \beta)$ score compositions, where $\{P_1, \ldots, P_\beta\}$ is the partition of P induced by any D-tree of P and β is the size of this partition, n_i is the cardinality of P_i, and $h_c(n_i)$ and $h_s(n_i)$ are the numbers of coordinate comparisons and score compositions used, respectively, to compute the ten $f()$-optimal boxes of P_i.*

Proof. Build a D-tree T of P in $O(n \lg n)$ comparisons (Lemma 11). Let $u_1, \ldots, u_\beta$ be the leaves of T which satisfy $S(u_i) = P_i$ for all $i \in [1..n]$. Compute the set $\texttt{Ten}(S(u_i)) = \texttt{Ten}(P_i)$ in $h_c(n_i)$ coordinate comparisons and $h_s(n_i)$ score compositions. By using a post-order traversal of T, for each internal node v of T compute $\texttt{Ten}(A(v))$ from $\texttt{Ten}(A(v_1))$ and $\texttt{Ten}(A(v_2))$, where v_1 and v_2 are the children nodes of v, in $O(1)$ score compositions (Lemma 13). The $f()$-optimal box of P is the box $\texttt{B}_{opt}(A(r))$, where r is the root node of T and satisfies $A(r) = P$. In total, this algorithm runs in $O(n \lg n) + \sum_{i=1}^{\beta} h_c(n_i) = O(n \lg n + \sum_{i=1}^{\beta} h_c(n_i))$ coordinate comparisons and $\sum_{i=1}^{\beta} h_s(n_i) + \sum_{i=1}^{\beta-1} O(1) = O(\sum_{i=1}^{\beta} h_s(n_i) + \beta)$ score compositions. $\square$

Corollary 15 *There exists an algorithm that finds an $f()$-optimal box of P in $O(n \lg n + \sum_{i=1}^{\beta} n_i \lg n_i)$ comparisons and $O(\sum_{i=1}^{\beta} n_i^2 \lg n_i + \beta)$ score compositions, where β is the size of the partition $\{P_1, \ldots, P_\beta\}$ of P induced by any D-tree of P, and $n_i = |P_i|$ for all i.*

8 Dealing with Windmills

In this section we use Lemma 8 to obtain a variant of the algorithm in Theorem 14. The set $S(u)$ of every leaf node u of any D-tree of P does not admit a diagonalization and has a windmill containing an extreme point of $S(u)$. The idea is to remove the extreme points of $S(u)$ and then recursively build a D-tree of the remaining points. This approach yields a diagonalization in depth of the point set, potentially reducing the number of score compositions.

Definition 16 *An extended diagonalization tree of P, D^*-tree, is defined recursively as follows: Each leaf node u of a D-tree of P satisfying $|S(u)| > 1$ is replaced by a node u' containing the set $X(u)$ of the four extreme points of $S(u)$, and if the set $S(u) \setminus X(u)$ is not empty then u' has as its only one child a D^*-tree of $S(u) \setminus X(u)$.*

Lemma 17 *Every D^*-tree of P has the same number σ of one-child nodes, contains $n - 4\sigma$ leaves nodes, and every leaf node u satisfies $|S(u)| = 1$ or $|S(u)| = 4$. A D^*-tree of P requires $O(n)$ space and can be built in $O(n \lg n + \sigma n)$ comparisons.*

Proof. The first part of the lemma can be seen from Lemma 10 and Definition 16. A D^*-tree of P can be built in $O(n \lg n + \sigma n)$ comparisons by following the same algorithm to build a D-tree of P until finding a leaf node u such that $S(u)$ does not admit a diagonalization. At this point we pay $O(n)$ comparisons in order to continue the algorithm with the set $S(u) \setminus X(u)$ according to Definition 16. Since this algorithm finds σ nodes u, the total comparisons are $O(n \lg n + \sigma n)$. The D^*-tree has n nodes of bounded degree and hence can be encoded in linear space. □

Theorem 18 *There exists an algorithm that finds an $f()$-optimal box of P in $O(n \lg n + \sigma n)$ coordinate comparisons and $O(\sigma n \lg n)$ score compositions, where σ is the number of one-child nodes of every D^*-tree of P.*

Proof. Build a D^*-tree T of P in $O(n \lg n + \sigma n)$ comparisons (Lemma 17). For each of the $n - 4\sigma$ leaves nodes u of T compute $\texttt{Ten}(S(u))$ in constant score compositions. Then, using a post-order traversal of T, compute $\texttt{Ten}(S(u))$ for each internal node u as follows: If v has two children v_1 (the left one) and v_2 (the right one), then $\{A(v_1), A(v_2)\}$ is a diagonalization of $A(v)$ and $\texttt{Ten}(A(v))$ can be computed in $O(1)$ score compositions from $\texttt{Ten}(A(v_1))$ and $\texttt{Ten}(A(v_2))$ (Lemma 13). Otherwise, if v is one of the σ one-child nodes, then $\texttt{Ten}(A(v))$ can be computed in $O(n \lg n)$ worst-case comparisons and score compositions. Namely, if a box of $\texttt{Ten}(A(v))$ contains a at least one point of $X(u)$ in the boundary then it can be found in $O(n \lg n)$ comparisons and score compositions [5]. Otherwise, it is a box of $\texttt{Ten}(A(v'))$, where v' is the child of v. We pay $O(1)$ score compositions for each of the $O(n)$ two-child nodes and $O(n \lg n)$ score compositions for each of the σ one-child nodes. Then the total score compositions is $O(n + \sigma n \lg n)$. □

9 Future work

The definition of $\texttt{Ten}(\cdot)$ can be used for further results: Suppose the point set P can be partitioned into subsets $P_1, P_2, \ldots, P_k$ so that bounding boxes $\texttt{Box}(P_1), \texttt{Box}(P_2), \ldots, \texttt{Box}(P_k)$ are pairwise disjoint and any axis-parallel line stabs at most one of them. Once $\texttt{Ten}(P_1), \texttt{Ten}(P_2), \ldots, \texttt{Ten}(P_k)$ have been computed, an optimal box of P can be found in $O(k \lg k)$ comparisons and $O(k^2 \lg k)$ score compositions. Finding an optimal decomposition $P_1, P_2, \ldots, P_k$ is our main issue.

References

[1] G. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. In *Proc. of the USSR Academy of Sciences*, volume 146, pages 263–266, 1962.

[2] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Oper. Res. Lett.*, 39(4):246–251, 2011.

[3] J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, 1984.

[4] R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay sorting log n-block sequences. *SIAM J. Comp.*, 30(1):1–43, 2000.

[5] C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, and I. Ventura. Bichromatic separability with two boxes: A general approach. *J. Algorithms*, 64(2-3):79–88, 2009.

[6] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *J. Comput. Syst. Sci.*, 52(3):453–470, 1996.

[7] D. P. Dobkin and S. Suri. Dynamically computing the maxima of decomposable functions, with applications. In *FOCS*, pages 488–493, 1989.

[8] J. Eckstein, P. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Comput. Optim. App.*, 23(3):285–298, 2002.

[9] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1968.

[10] Y. Liu and M. Nediak. Planar case of the maximum box and related problems. In *CCCG*, pages 14–18, 2003.

[11] H. Mannila. Measures of presortedness and optimal sorting algorithms. In *IEEE Trans. Comput.*, volume 34, pages 318–325, 1985.

[12] A. Moffat and O. Petersson. An overview of adaptive sorting. *Australian Comp. J.*, 24(2):70–77, 1992.

[13] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[14] T. Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electronic Notes in Theoretical Computer Science*, 61:191–200, 2002. CATS'02.