# An Instance-Optimal Algorithm for Bichromatic Rectangular Visibility

**Jean Cardinal**
Université libre de Bruxelles (ULB), Brussels, Belgium

**Justin Dallant**
Université libre de Bruxelles (ULB), Brussels, Belgium

**John Iacono**
Université libre de Bruxelles (ULB), Brussels, Belgium

## Abstract

Afshani, Barbay and Chan (2017) introduced the notion of instance-optimal algorithm in the order-oblivious setting. An algorithm $A$ is instance-optimal in the order-oblivious setting for a certain class of algorithms $\mathcal{A}$ if the following hold:

- $A$ takes as input a sequence of objects from some domain;
- for any instance $\sigma$ and any algorithm $A' \in \mathcal{A}$, the runtime of $A$ on $\sigma$ is at most a constant factor removed from the runtime of $A'$ on the worst possible permutation of $\sigma$.

If we identify permutations of a sequence as representing the same instance, this essentially states that $A$ is optimal on every possible input (and not only in the worst case).

We design instance-optimal algorithms for the problem of reporting, given a bichromatic set of points in the plane $S$, all pairs consisting of points of different color which span an empty axis-aligned rectangle (or reporting all points which appear in such a pair). This problem has applications for training-set reduction in nearest-neighbour classifiers. It is also related to the problem consisting of finding the decision boundaries of a euclidean nearest-neighbour classifier, for which Bremner et al. (2005) gave an optimal output-sensitive algorithm.

By showing the existence of an instance-optimal algorithm in the order-oblivious setting for this problem we push the methods of Afshani et al. closer to their limits by adapting and extending them to a setting which exhibits highly non-local features. Previous problems for which instance-optimal algorithms were proven to exist were based solely on local relationships between points in a set.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** computational geometry, instance-optimality, colored point sets, empty rectangles, visibility

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2021.24

**Related Version** *Full Version*: `https://arxiv.org/abs/2106.05638` [12]

**Funding** *Justin Dallant*: This work was supported by the French Community of Belgium via the funding of a FRIA grant.
*John Iacono*: Supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.

## 1 Introduction

In the theoretical study of algorithms one often quantifies the performance of an algorithm in terms of the worst case or average running time over a distribution of inputs of a given size. Sometimes, more precise statements can be made about the speed of an algorithm on certain instances by expressing the running time in terms of some parameter depending on the input. One class of such algorithms are the so-called output-sensitive algorithms, where the parameter is the size of the output. In computational geometry, a classical example is computing the convex hull of a set of $n$ points in the plane in $\mathcal{O}(n \log h)$ time, where $h$ is the size of the convex hull [15, 5]. More recently, Afshani et. al. [1] introduced a specific notion of *instance optimality in the order-oblivious setting* and designed algorithms with this property for different problems on point sets. Roughly speaking, an algorithm is instance optimal in a certain class of algorithms $\mathcal{A}$ if on any input sequence $S$, its performance on $S$ is at most a constant factor removed from the best performance of any algorithm in $\mathcal{A}$ on $S$. In the order-oblivious setting, performance is defined as the worst-case runtime over all permutations of the input sequence (the motivation behind this will be made clear below). A common characteristic of most problems solved in [1] is that they are based around local relations between points in $S$, in the sense that the relation between two points $p, q \in S$ depends solely on their coordinates and not on those of any other point in $S$. This is important because it allows one to decompose certain queries into independent queries on a partition of $S$. In this paper we push these methods closer to their limits by adapting and applying them to a problem which does not fit in this framework.

**The studied problem.** Here we consider the following problem: given a set $S$ of points $n$ in the plane with distinct $x$ and $y$ coordinates, each colored red or blue, report the red/blue pairs of points such that the rectangles they span contain no point of $S$ in their interior. This again has applications to machine learning and more specifically nearest-neighbour classifiers. Indeed, by solving this problem and discarding all points which do not not appear in such a pair, we obtain a (possibly much smaller) set of points where for any point $y$ in the plane, the color of its nearest neighbour is the same as in the original set of points for the $L_1$ distance (as pointed out in [10], where this problem was perhaps first studied). This remains true even when *a priori* unknown (and possibly non-linear) scalings might be applied to the $x$ and $y$ axis after this preprocessing step. In fact, the resulting set of points consists of exactly those necessary for this to hold, so this constitutes an optimal reduction of the training set in that sense. Note that this problem does not fit in the general framework of [1], as whether two point span an empty rectangle depends on the position of all other points.

**Related works.** Pairs of points spanning empty rectangles and the corresponding graphs have been studied at numerous occasions in the past, under various names. They are called *rectangular influence graphs* in [10], which discusses applications to data clustering and classification. In [9] a similar relation is called *direct dominance*, and a worst-case optimal algorithm to report all pairs of related points is given. This algorithm runs in $O(n \log n + h)$ time, where $h$ is the size of the output. A straightforward adaptation yields an algorithm with the same running time for the bichromatic problem studied here. In [18] this relation is called *rectangular visibility* and a different algorithm with the same running time is given as well as algorithms for the dynamic query setting. The expected size of a largest independent set in this graph is studied in [7] (where they call such graphs *Delaunay graphs with respect to rectangles*). Generalizations and variations of this type of relation between points have also been widely studied [17, 16, 4, 2, 11, 14, 13, 19, 8]. Another problem of note which is closely related to the one we study here is the following: given a set $S$ of $n$ points in the plane, each colored red or blue, compute the subset of edges of the Voronoï diagram of $S$ which are adjacent to both a site corresponding to a blue point and a site corresponding to a red point. This problem has some relevance to machine learning as we can equivalently state it as finding the boundaries of a nearest-neighbour classifier with two classes in the plane. A third formulation is finding the pairs of red and blue points such that there is an empty disk whose boundary passes through both. In [3], Bremner et. al. show that this problem can be solved in output-sensitive optimal $\mathcal{O}(n \log h)$ time, where $h$ is the size of the output. It is an interesting open problem to find instance-optimal algorithms for this problem in the order-oblivious setting (or prove that no such algorithm exists).

**Paper organization.** In Section 2 we motivate and state more precisely the notion of instance-optimality we work with in this paper. In Section 3 we define the problem formally and give an instance lower bound in the order-oblivious model by adapting the adversarial argument of [1]. The key new ingredients are a new definition of safety and a way to deal with the fact that here the adversary cannot necessarily change the expected output of the algorithm by moving a single point inside a so-called non-safe region. In Section 4 we give an algorithm and prove that its runtime matches the lower bound. The main observation which makes this work is that while the algorithms in [1] require the safety queries to be decomposable (which they are not here), we can afford to do some preprocessing to make them behave as if they were decomposable, as long as the amount of work done stays within a constant of the lower bound. In section 5 we mention that when competing against algorithms which can do linear queries, instance-optimality in the order-oblivious setting is impossible. Some details and proofs have been left out of this version and can be found in the full paper [12].

## 2 Instance optimality in the order-oblivious setting and model of computation

Ideally, we would like to consider a very strong notion of optimality, where an algorithm is optimal if on every instance its runtime is at most a constant factor removed from the algorithm with the smallest runtime on that particular instance. There is an obvious flaw with this definition, as for every instance we can have an algorithm "specialized" for that instance, which simply checks if the input is the one it is specialized for then returns the expected output without any further computation when it is the case (and spends however much time it needs to compute the correct output otherwise). For problems which are not solvable in linear time in the worst-case, this prohibits the existence of such optimal algorithms. One way to get around this issue in some cases and get a meaningful notion of instance-optimality is the following, taken from [1].

▶ **Definition 1.** *Consider a problem where the input consists of a sequence of $n$ elements from a domain $\mathcal{D}$. Consider a class $\mathcal{A}$ of algorithms. A* correct *algorithm refers to an algorithm that outputs a correct answer for every possible sequence of elements in $\mathcal{D}$. For a set $S$ of $n$ elements in $\mathcal{D}$, let $T_A(S)$ denote the maximum running time of $A$ on input $\sigma$ over all $n!$ possible permutations $\sigma$ of $S$. Let $\mathrm{OPT}(S)$ denote the minimum of $T_{A'}(S)$ over all correct algorithms $A' \in \mathcal{A}$. If $A \in \mathcal{A}$ is a correct algorithm such that $T_A(S) \leq O(1) \cdot \mathrm{OPT}(S)$ for every set $S$, then we say $A$ is* instance-optimal in the order-oblivious setting.

By measuring the performance of an algorithm on an instance as the maximum runtime over all permutations of the instance elements, the algorithm can no longer take advantage of the order in which the input elements are presented. In particular, simply checking if the input is a specific sequence is no longer a good strategy. Here, the domain $\mathcal{D}$ consists of all points in the plane, colored red or blue. An instance is a sequence of points, no two sharing the same $x$ or $y$ coordinate. However, we really want to consider this sequence as a set of points, as the order in which the points are presented does not change the instance conceptually. Thus, it makes sense for us to consider a performance metric for which algorithms cannot take advantage of this order. For the class of algorithms $\mathcal{A}$, we will consider algorithms in a restricted real RAM model where the input can only be accessed through comparison queries. That is, the algorithms can compare the $x$ or $y$ coordinates of two points but not, for example, evaluate arbitrary arithmetic expressions on these coordinates. We refer to such algorithms as *comparison-based algorithms*. The lower bound works even for a stronger model of computation, comparison-based decision trees (assuming at least a unit cost for every point returned in the output). We could also allow the comparison of the $x$ coordinate of a point with the $y$ coordinate of another without changing any of the results of this paper.

## 3 Lower-bound for comparison-based algorithms in the order-oblivious setting

### Some basic notations and definitions

Throughout this section we consider a set $S$ of $n$ red and blue points in the plane. We assume that $S$ is non-degenerate, in the sense that no two points in $S$ share the same $x$ or $y$ coordinate (in particular, all points are distinct). If $p$ is a point, we will denote its $x$ and $y$ coordinates as $x(p)$ and $y(p)$ respectively and its color as $c(p)$.

▶ **Definition 2.** *A point $p$* dominates *$q \neq p$ if $x(p) \geq x(q)$ and $y(p) \geq y(q)$. A point is* maximal *(resp.* minimal*) in $S$ if is dominated by (resp. dominates) no point in $S$.*

▶ **Definition 3** (Visible and participating points)**.** *Let $p, q$ be two points in $S$. We say that $q$ is* visible *from $p$ in $S$ (or that $p$* sees *$q$ in $S$) if the axis-aligned box spanned by $p$ and $q$ contains no point of $S$ in its interior. We say that $p \in S$* participates *in $S$ if it is visible from a point in $S$ of the opposite color. We will omit the set $S$ when it is clear from context.*

The problems we want to solve can thus be restated as follows.

▶ **Problem 4** (Reporting participating points)**.** *Report all points which participate in $S$.*

▶ **Problem 5** (Reporting red-blue pairs of visible points)**.** *Report all red-blue pairs of points $(p, q)$ such that $p$ and $q$ see each other in $S$.*

The following definitions will also be useful for us.

▶ **Definition 6.** *We call the set of minimal points of $S$ the* NE-minimal-set *of $S$ (for "North-East-minimal set"). The* NW-minimal-set*,* SE-minimal-set *and* SW-minimal-set *of $S$ are defined symmetrically (see Figure 1). In particular, the* SE*-minimal set of $S$ is the set of maximal points in $S$.*

▶ **Definition 7.** *Let $B$ be an axis-aligned box. We denote the $x$ coordinate of the right boundary (resp. left boundary) of $B$ as $x_{\max}(B)$ (resp. $x_{\min}(B)$). We denote the $y$ coordinate of the top boundary (resp. bottom boundary) of $B$ as $y_{\max}(B)$ (resp. $y_{\min}(B)$).*

*The* cross *of $B$, denoted as $cross(B)$ is the set of points $p$ in the plane such that $x_{\min}(B) \leq x(p) \leq x_{\max}(B)$ or $y_{\min}(B) \leq y(p) \leq y_{\max}(B)$.*

*The* quadrants *of $B$ are the connected components of $\mathbb{R}^2 \setminus cross(B)$. We call the four components the* NE-quadrant*,* NW-quadrant*,* SE-quadrant *and* SW-quadrant*, denoted as* NE$(B)$*,* NW$(B)$*,* SE$(B)$ *and* SW$(B)$ *respectively (see Figure 2).*

All boxes we consider are axis-aligned boxes in the plane. For ease of exposition, we assume that all boxes we consider have no point of $S$ on the boundaries of their four quadrants.

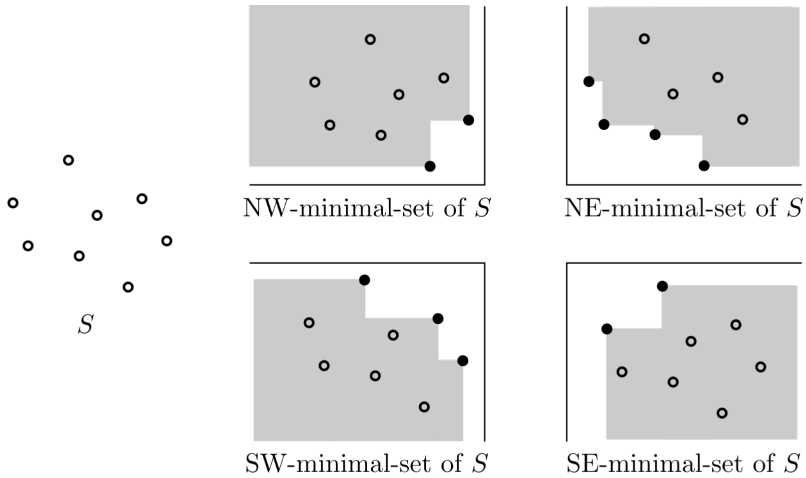


**Figure 1** A set of points $S$ and the four minimal-sets of $S$. No point in the shaded regions can appear on the corresponding minimal-sets.

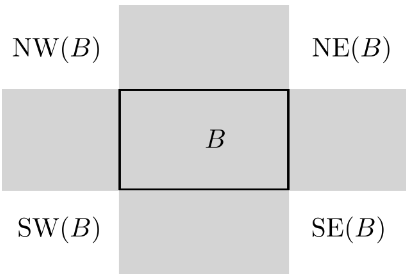


**Figure 2** The four quadrants of an axis-aligned box $B$. The shaded region corresponds to $cross(B)$.

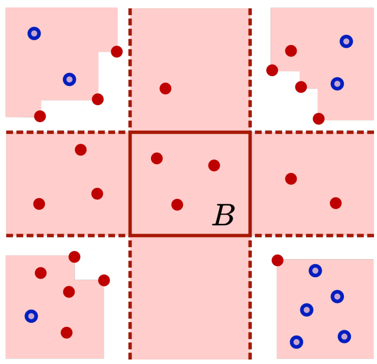


**Figure 3** A red-safe box $B$.

## Lower Bound

We prove an entropy-like lower bound on the number of comparisons which have to be done to solve the problems of reporting participating points in the order-oblivious setting. The proof and terminology are largely inspired from the lower bound proofs in [1] but some additional arguments, which we underline later, are required. We need a few definitions in order to state our lower bound.

▶ **Definition 8.** *An axis-aligned box $B$ is* red-cross-safe *(resp.* blue-cross-safe*) for $S$ if all points in $S \cap cross(B)$ are red (resp. blue). It is* cross-safe *if it is red-cross-safe or blue-cross-safe.*

*It is* red-safe *if it is red-cross-safe and the* NE*-minimal (resp.* NW*-minimal,* SE*-minimal,* SW*-minimal) set of $S \cap \mathrm{NE}(B)$ (resp. $S \cap \mathrm{NW}(B)$, $S \cap \mathrm{SE}(B)$, $S \cap \mathrm{SW}(B)$) is red (see Figure 3). We define* blue-safe *boxes similarly. A box is* safe *if it is red-safe or blue-safe.*

*A subset $S' \in S$ has one of these properties if it can be enclosed by a box with the property.*

Notice that if a subset $S' \in S$ is safe, then no point in in $S'$ participates in $S$. Thus, in an intuitive sense, a partition of non-participating points into safe subsets can be seen as a certificate for the fact that these points do not participate. The minimal entropy of such a partition is then the minimal amount of information required to encode such a certificate.

▶ **Definition 9.** *A partition* $\Pi$ *of* $S$ *is* respectful *if every member* $S_k \in \Pi$ *is a singleton or a safe subset of* $S$*. The* entropy *of a partition* $\Pi$ *is* $\mathcal{H}(\Pi) := \sum_{S_k \in \Pi} \frac{|S_k|}{n} \log \frac{n}{|S_k|}$*. The* structural entropy $\mathcal{H}(S)$ *of* $S$ *is the minimum of* $\mathcal{H}(\Pi)$ *over all respectful partitions* $\Pi$ *of* $S$*.*

We can now state our lower bound:

▶ **Theorem 10.** *For the problem of reporting participating points in the order-oblivious comparison-based model,* $\mathrm{OPT}(S) \in \Omega(n(\mathcal{H}(S)+1))$*. Consequently, for reporting red-blue pairs of visible points,* $\mathrm{OPT}(S) \in \Omega(n(\mathcal{H}(S)+1)+h)$*, where* $h$ *is the size of the output.*

The proof is similar to what can be found in [1]. In their case however, if a point can be moved anywhere inside a non-safe box then it can be moved in a way that changes the expected output of the algorithm. In our case, we can do something similar but sometimes need to move multiple points to affect the expected output (but a constant number are enough). We use a simple argument about the maximum number of chips which can be placed on a tree in a constrained way to show that this has no impact on the lower bound. The full proof, which is too long for this version of the paper, can be found in the full paper [12]. From this lower bound one can also deduce that the existence of an instance-optimal algorithm in the order-oblivious setting for reporting participating points implies the existence of such an algorithm for reporting red-blue pairs of visible points (using known results about worst-case optimal algorithms). Thus, we will focus on the former problem from now on.

## 4 Instance optimal comparison-based algorithm in the order-oblivious setting

In this section we present a comparison-based algorithm for reporting participating points with a runtime matching the lower bound we saw in the previous section (note that worst-case optimal $\mathcal{O}(n \log n + h)$ algorithms are easy to obtain by the method of [9]). Once again, the main algorithm will be very similar to what is done in [1], however their results do not directly apply here. The main difficulty in adapting their algorithm to our case is that the relation we consider here is not decomposable. More precisely, if we know that some point $p$ does or does not participate in $S'$ and $S''$, we cannot use this to decide if $p$ participates in $S' \cup S''$. The bulk of the work here will thus be to preprocess the set $S$ in order to make the safety tests decomposable, while keeping our preprocessing time within $O(n(\mathcal{H}(S)+1))$.

### 4.1 The main algorithm

Before we go into detail about how to preprocess the points let us see how, if we can build the right data structure, we can use it to report the participating points in $S$ in $O(n(\mathcal{H}(S)+1))$ time. We will need the following definition and observations:

▶ **Definition 11.** *Let* $S$ *be a set of red and blue points. A subset* $S' \subset S$ conforms with $S$ *if it contains all points which participate in* $S$*.*

▶ **Observation 12.** *Let* $S$ *be a set of of red and blue points and let* $S'$ *be a subset which conforms with* $S$*. Then an axis-aligned box is safe for* $S$ *if and only if it is safe for* $S'$*. Moreover, a point participates in* $S$ *if and only if it participates in* $S'$*.*

▶ **Observation 13.** *If* $B$ *is a safe box for* $S$*, then any sub-box of* $B$ *is safe for* $S$*.*

▶ **Observation 14.** *Let* $p$ *be a point in* $S$ *and let* $B$ *be some axis-aligned box bounding* $p$*. If* $B$ *is safe, then* $p$ *does not interact in* $S$*.*

We have the following algorithm and theorem, adapted from [1], where $\delta$ is a constant to be chosen later:

**Algorithm 1** Reporting participating points.

**Input:** A point set $S$ of size $n$

1 Set $Q = S$.
2 **for** $j = 0, 1, \ldots \lfloor \log(\delta \log n) \rfloor$ **do**
3     Partition the points in $Q$ using a kd-tree to get $r_j = 2^{2^j}$ subsets $Q_1, \ldots, Q_{r_j}$ of size at most $\lceil |Q|/r_j \rceil$, along with corresponding bounding boxes $B_1, \ldots, B_{r_j}$.
4     **for** $i = 0, 1, \ldots, r_j$ **do**
5         **if** *$B_i$ is safe for $Q$* **then**
6             Prune all points in $Q_i$ from $Q$.
7 Solve the reporting problem for the remaining set $Q$ directly in $O(|Q| \log |Q|)$ time.

▶ **Theorem 15.** *Let $S$ be a set of $n$ points in general position. Suppose we have preprocessed $S$ such that for any subset $S' \subset S$ containing all points which participate in $S$ we can test if an axis-aligned box is safe for $S'$ in $O(n^{1-\alpha})$ time, plus the cost of a constant number of range-emptiness queries on $S'$.*
*Then Algorithm 1 can report all points which participate in $S$ in $O(n(\mathcal{H}(S) + 1))$ time.*

We reiterate the proof for the sake of completeness and to underline how Observation 12 and our additional assumptions on preprocessing factor into it:

**Proof.** By Observation 12 and Observation 14, we only ever prune points which do not participate in the original set $S$ and we never modify which points participate among those that remain. Thus the algorithm invoked at line 6 will compute the correct output.

By assumption, testing a box for safety in $Q$ can be done in $O(n^{1-\alpha})$ time plus the cost of a constant number of range-emptiness queries on $Q$. Using a simple and ingenious trick by T. Chan [6], we can perform $r$ orthogonal range emptiness queries on a set of size $m$ in $O(m \log r + r^{O(1)})$ time. Thus, the $r_j$ tests of lines 3 and 4 can be done in $O(|Q| \log r_j + r_j^{O(1)} + r_j n^{1-\alpha})$ time. As $r_j < n^{\delta}$, by taking $\delta$ small enough the $r_j^{O(1)} + r_j n^{1-\alpha}$ term can be made sublinear. As the outer loop of the algorithm is only executed $O(\log(\log n))$ times the total contribution of these terms over the whole algorithm can also be made sublinear and thus negligible. Line 2 can be done in $O(|Q| \log r_j)$ time by the classical recursive algorithm to compute kd-trees.

Now let $n_j$ be the number of points in $Q$ just after iteration $j$. The runtime of the algorithm is in $O(\sum_j n_j \log r_{j+1})$. This includes the final step at line 6, as for $j = \lfloor \log(\delta \log n) \rfloor$ (i.e. after the last iteration of the outer loop) we have $O(|Q| \log |Q|) \subset O(n_j \log n^{2\delta}) = O(n_j \log r_{j+1})$. Let $\Pi$ be a respectful partition of $S$ and consider $S_k \in \Pi$. At iteration $j$ all subsets $Q_i$ lying entirely inside the bounding box of $S_k$ are pruned by Observation 13. Since the bounding box of $S_k$ intersects at most $O(\sqrt{r_j})$ cells of the kd-tree, the number of points in $S_k$ remaining after iteration $j$ is $min\{|S_k|, O(\sqrt{r_j} \cdot n/r_j)\} = min\{|S_k|, O(n/\sqrt{r_j})\}$. The $S_k$'s cover the entire point set so by double summation we have:

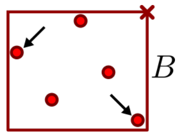


**Figure 4** A cell of red points and the corresponding box $B$. The relevant points are indicated by arrows. The box point is indicated by a cross.

$$\begin{aligned}
\sum_j n_j \log r_{j+1} &\le \sum_j \sum_k \min\left\{|S_k|, O\left(n/\sqrt{2^{2^j}}\right)\right\} \cdot 2^{j+1} \\
&= \sum_k \sum_j \min\left\{|S_k|, O\left(n/2^{2^{j-1}}\right)\right\} \cdot 2^{j+1} \\
&\in O\left(\sum_k \sum_{j \le \log(2\log(n/|S_k|))} |S_k| \cdot 2^j + \sum_{j > \log(2\log(n/|S_k|))} n \cdot 2^j / 2^{2^{j-1}}\right) \\
&\in O\left(\sum_k |S_k|(\log(n/|S_k|) + 1)\right) = O(n(\mathcal{H}(S) + 1)).
\end{aligned}$$ ◀

## 4.2 Cross-safety tree

In order to solve our problem instance-optimally (in the order-oblivious comparison-based model), we want to design a data-structure which allows us to quickly test if a given axis aligned box $B$ is safe for $S$. To make the presentation clearer, we focus on testing if $B$ is red-cross-safe and the NE-minimal-set of $\mathrm{NE}(B) \cap S$ is red. This can then be repeated symmetrically for $\mathrm{NW}(B), \mathrm{SE}(B)$ and $\mathrm{SW}(B)$ to test if $B$ is red-safe (and similarly for testing if $B$ is blue-safe). We will see how to build and query the following data structure:

▶ **Definition 16.** *A* cross-safety tree $T_S$ on $S$ *is a recursive partitioning on the plane similar to a kd-tree where we stop subdividing the points once we have reached a cross-safe subset of points. The root of $T_S$ corresponds to $S$. If $S$ is not cross-safe, we split the points around a vertical line $L$ such that the two open halfplanes defined by $L$ partition $S$ into two sets $S_1$ and $S_2$ of size at most $\lceil |S|/2 \rceil$. The children of the root will then correspond to $S_1$ and $S_2$. For every newly created node we repeat the procedure, partitioning the set of points by median $x$ coordinates at even levels of the tree and median $y$ coordinates at odd levels, until the points contained are a cross-safe subset of $S$.*

*The* cell *of a point $p$, denoted as $C_p$ is the the subset of points contained in the same leaf as $p$. A cell of $T_S$ is red (resp. blue) if the points it contains are red (resp. blue).*

*The* box of a point $p$, *denoted as $B_p$ is the smallest axis-aligned box containing all points in $C_p$ (slightly extended to enforce our assumption of only considering boxes for which there are no points on the boundary of their four quadrants). A box of $T_S$ is red (resp. blue) if the points it contains are red (resp. blue).*

*The* box-point of $p$ *is the top-right point of the box of $p$, and has the same color as $p$.*

*A point is* relevant *if it has the minimum $x$ or $y$ coordinate among all points in its box (or equivalently, in its cell).*

*(See Figure 4 for an illustration of these definitions.)*

*Each node $u$ in the tree stores:*

- *The set of point it contains, which we denote as $P_u$.*

- *The smallest axis-aligned bounding box of $P_u$, which we denote as $B_u$.*
- *The red box-points of minimum $x$ and $y$ coordinates among all box-points associated with a red point in $P_u$.*
- *The subset of all relevant blue points in the minimal set of $P_u$, sorted by $x$-coordinate.*

Note that if the points in a minimal set are sorted by $x$ coordinate then they are also sorted (in reverse order) by $y$ coordinate.

## 4.3 Querying a cross-safety tree

Before we see how to build a cross-safety tree on $S$ efficiently, let us see how we make queries on a node $u$ of $T_S$. A query consists of a lower range $R_L = [x_L, +\infty] \times [y_L, +\infty]$ and an upper range $R_U = [-\infty, x_U] \times [-\infty, y_U]$ such that $R_L \cap R_U \neq \emptyset$ and neither the boundaries of $R_L$ nor $R_U$ intersect any blue box of $T_S$. It returns:

- $rx_u$, the minimum $x$-coordinate of any red box-points associated with a red point in $P_u \cap R_L$ (set to $+\infty$ if there are no red points in $P_u \cap R_L$).
- $ry_u$, the minimum $y$-coordinate of any red box-points associated with a red point in $P_u \cap R_L$ (set to $+\infty$ if there are no red points in $P_u \cap R_L$).
- $bx_u$, the minimum $x$-coordinate of any blue points in the minimal set of $P_u \cap R_L \cap R_U$ (set to $+\infty$ if there are no blue points in the minimal set of $P_u \cap R_L \cap R_U$).
- $by_u$, the minimum $y$-coordinate of any blue points in the minimal set of $P_u \cap R_L \cap R_U$ (set to $+\infty$ if there are no blue points in the minimal set of $P_u \cap R_L \cap R_U$).

Observe the following:

▶ **Observation 17.** *If a horizontal or vertical line passes through a red point then it does not intersect any blue-cross-safe box. The same applies when "red" and "blue" are swapped.*

▶ **Observation 18.** *Let $B$ be a red-cross-safe box. Then all points in $B$ dominate (resp. are dominated by) the same subset of blue points in $S$. The same applies when "red" and "blue" are swapped.*

We will need a few additional lemmas.

▶ **Lemma 19.** *The points corresponding to $bx_u$ and $by_u$ are relevant points of $T_S$.*

**Proof.** Let $P = P_u \cap R_L \cap R_U$, and suppose there is a blue point on the minimal set of $P$. Let $p$ be the leftmost point in $P$ which does not dominate any red point in $P$. Suppose that $bx_u$ is not equal to $x(p)$. The only way this can happen is if $p$ is not on the minimal set of $P$, meaning that there is a blue point $q$ which is dominated by $p$ (as $p$ does not dominate any red point). In particular, $q$ lies to the left of $p$. By definition of $p$, $q$ thus dominates a red point. But if $q$ dominates a red point and $p$ dominates $q$, then $p$ dominates a red point, which contradicts the definition of $p$. Thus $bx_u = x(p)$. Moreover, if $p$ does not dominate any red point, then no point in its box dominates a red point, as the box is blue-cross-safe. Because the boundaries of $R_L$ and $R_U$ do not intersect any blue box, the whole box of $p$ is contained in $P$. Thus, by definition of $p$, it is the leftmost point in its box and it is relevant. The same reasoning shows that $by_u$ is the $y$ coordinate of a relevant point. ◀

▶ **Lemma 20.** *If the bounding box $B_u$ of points in $P_u$ lies entirely in $R_L$, then we can return the necessary information in $O(\log n)$ time.*

**Proof.** In this case, $rx_u$ and $ry_u$ are already stored in the node, so we can return them in constant time. By Lemma 19, the point giving $bx_u$ is the relevant blue point with minimum $x$ coordinate among all blue points in the minimal set of $P_u \cap R_U$. This is also the relevant blue point with maximum $y$ coordinate among all blue points in the minimal set of $P_u \cap R_U$.

We can do a binary search through the relevant blue points in the minimal set of $P_u$ to find the point $p$ below the line $y = y_U$ with maximum $y$ coordinate. If this point is not in $R_U$ (because $x(p) > x_U$) then no relevant blue point in the minimal set of $P_u$ is in $R_U$, as all other relevant blue points $q$ in the minimal set of $P_u$ with $y(q) \leq y_U$ will have $x(q) > x(p) > x_U$. In this case we set $bx_u$ to $+\infty$. Otherwise $bx_u = x(p)$.

We can find $by_u$ similarly with a single binary search. ◀

▶ **Lemma 21.** *If $u$ is not a leaf of $T_S$ and $B_u$ intersects the boundary of $R_L$, then we can return the necessary information after querying the children of $u$ with the same lower range $R_L$ (but possibly different upper ranges) and a constant amount of additional work.*

**Proof.** Suppose without loss of generality that the children of $u$ split $P_u$ by a vertical line (the situation for a horizontal line is similar). Let $v$ be the child corresponding to the left half-plane and $w$ the one corresponding to the right half-plane. Let us focus on computing $bx_u$, as this is the one requiring the most care.

Querying $v$ with the same $R_L$ and $R_U$ returns some values $rx_v$, $ry_v$, $bx_v$ and $by_v$. A blue point $p$ on the minimal set of $P_w \cap R_L \cap R_U$ is a blue point on the minimal set of $P_u \cap R_L \cap R_U$ if and only if $y(p) \leq by_v$ and $p$ does not dominate any red point in $P_v \cap R_U$. By Observation 18 this is equivalent to saying that $y(p) \leq by_v$ and $y(p) \leq ry_v$.

Thus we can compute $bx_u$ by setting $R'_U = [-\infty, x_U] \times [-\infty, \min\{y_U, ry_v\}]$, querying $w$ with $R_L$ and $R'_U$ to get values $rx_w$, $ry_w$, $bx_w$ and $by_w$, then setting $bx_u = \min\{bx_v, bx_w\}$. Notice that by Observation 17, we are allowed to query $w$ with $R'_U$ as its boundary does not intersect any blue box of $T_S$. It is then easy to see that $rx_u = \min\{rx_v, rx_w\}$, $ry_u = \min\{ry_v, ry_w\}$ and $by_u = \min\{by_v, by_w\}$. ◀

▶ **Lemma 22.** *If $u$ is a leaf of $T_S$ and $B_u$ intersects the lower boundary or the left boundary of $R_L$ but not both simultaneously, then we can return the necessary information in constant time.*

**Proof.** In this case, we know that $B_u$ is a red box, as by assumption the boundary of $R_L$ does not intersect any blue box of $T_S$. Thus $B_u$ contains no blue points and we know $bx_w = by_w = +\infty$. Suppose without loss of generality that $B_u$ intersects the left boundary of $R_L$. Then the rightmost point of $B_u$ is in $R_L$, and thus $P_u \cap R_L$ is non-empty. Because all points in $P_u$ have the same box-point $p = (p_x, p_y)$, we have $rx_u = p_x$ and $ry_u = p_y$. ◀

▶ **Lemma 23.** *If $u$ is a leaf of $T_S$ and $B_u$ intersects both the lower boundary and the left boundary of $R_L$, then we can return the necessary information after a single orthogonal range-emptiness test.*

**Proof.** Again, we know that $B_u$ is a red box and all points in $P_u$ have the same box point $p = (p_x, p_y)$. To know if we need to set $rx_u = ry_u = +\infty$ or $rx_u = p_x$ and $ry_u = p_y$, we simply need to test if there is a (necessarily red) point in $B_u \cap R_L$. This requires a single range-emptiness test. ◀

By applying the relevant result among Lemmas 20, 21, 22 and 23 recursively we get:

▶ **Theorem 24.** *We can query the root of a cross-safety tree $T_S$ in $O(\sqrt{n} \log n)$ time plus the cost of a single range-emptiness test.*

As a corollary, we get:

▶ **Corollary 25.** *Let $B$ be an axis aligned box. We can query a cross-safety tree $T_S$ to test if $B$ is red-*NE*-safe for $S$ in $O(\sqrt{n}\log n)$ time plus the cost of a constant number of orthogonal range-emptiness tests on $S$.*

**Proof.** With two orthogonal range-emptiness queries on the blue points of $S$ and one on the red points of $S$ we can test if $B$ contains at least one red point and cross($B$) contains only red points (that is, test if $B$ is red-cross-safe for $S$). If $B$ is not red-cross-safe for $S$ we can immediately return "No". We assume from now on that it is.

Let $R_L$ be the range corresponding to NE($B$), and let $R_U = [-\infty, +\infty] \times [-\infty, +\infty]$. Because $R$ is red-cross-safe, it is easy to see that the boundary of $R_L$ does not intersect any blue box of $T_S$ (this is also trivially true for $R_U$). Thus, we can query the root $u$ of $T_S$ with $R_L$ and $R_U$ to get the $x$ coordinate $bx_u$ of the blue point with minimum $x$ coordinate among all blue points on the minimal set of $\text{NE}(B) \cap S$, or $+\infty$ if no such point exists. In particular, this allows us to test if such a point exists. ◀

We also have the following:

▶ **Lemma 26.** *Let $S' \subset S$ be a subset which conforms with $S$ and let $B$ be an axis-aligned box. If $S' \cap B \neq \emptyset$, then we can replace all orthogonal range-emptiness tests on $S$ with the same tests on $S'$ in the procedure described in Corollary 25 (including the tests done while querying $T_S$) without affecting the outcome.*

**Proof.** If there are both red and blue points in $S \cap \text{cross}(B)$, then at least one of these blue points participates in $S$. Because $S'$ conforms with $S$, this blue point is also in $S'$, so $S'$ is not red-cross-safe. The converse is trivially true. Thus, the initial three range-emptiness tests return the same results on $S$ and $S'$.

Now consider the query done on $T_S$. If the corner of the lower range $R_L$ does not intersect a red leaf box of $T_S$, then no range-emptiness test is performed and the claim holds. Now suppose $R_L$ intersects a red leaf box $B$ of $T_S$. Let $S''$ be the set of points in $S$ where we remove all points in $S \cap B \cap R_L$ which are not in $S' \cap B \cap R_L$. Notice that by replacing the range-emptiness query on $S$ with one on $S'$, the procedure behave exactly like querying a cross-safety tree on $S''$. Because $S' \subset S'' \subset S$ we know that $S''$ conforms with $S$ and thus by Observation 12 the claim holds. ◀

Thus, this data-structure fits the prerequisites of Theorem 15, and we can use it to get an algorithm solving the problem in $O(n(\mathcal{H}(S) + 1))$ time after having built it. The only missing ingredient to get an instance-optimal algorithm is building the data-structure within the same asymptotic runtime. We show in the following section that we can indeed do this.

## 4.4 Construction in $O(n(\mathcal{H}(S) + 1))$ time

Rather than focusing on constructing the cross-safety tree specifically, we start with a bit more general setting.

▶ **Theorem 27.** *Let $S$ be a set of points. Let $C$ be a property on axis-aligned bounding boxes of the plane such that for boxes $B_2 \subset B_1$, if $C(B_1)$ is true then $C(B_2)$ is true. (Note that $C$ can depend on $S$).*

*A partition $\Pi$ of $S$ is $C$-respectful if every set in $\Pi$ is a singleton or can be enclosed by an axis aligned bounding box $B$ such that $C(B)$ is true.*

*Let $\mathcal{H}_C(S)$ be the minimum of $\mathcal{H}(\Pi)$ over all $C$-respectful partitions of $S$.*

*If property $C(B)$ can be tested in $O(|S \cap B|)$ time when given access to $S \cap B$, then a kd-tree $T_S^C$ with stopping condition $C$ on the leaves can be built in $O(n(\mathcal{H}_C(S)+1))$ time.*

*This remains true if for each node in the tree we do an additional linear amount of work (in the number of points considered at that node).*

The proof is similar in spirit to that of Theorem 15, although simpler.

**Proof.** Consider the classical top-down recursive approach to construct a kd-tree on $S$ (with linear-time median, selection), where we stop subdividing points once we have reached a bounding-box $B$ with property $C$. Consider any $C$-respectful partition $\Pi$ of $S$. Let $S_k \in \Pi$ and let $B_k$ be a corresponding bounding box with the property $C$. At the $j$'th level of the recursion, we have partitioned the plane into $O(2^j)$ boxes each containing at most $\lceil n/2^j \rceil$ points still to be considered. Any box $B$ of $T_S^C$ which is entirely contained in $B_k$ has property $C$ and can be set as a leaf. In other words the box $B$ does not need to be recursed on and the points in $B \cap S$ are not considered at level $j$ or lower. Because $B_k$ intersects at most $O(\sqrt{2^j})$ boxes of $T_S^C$ at level $j$, the number of points in $S_k$ to consider at level $j$ is $min\{|S_k|, O(\sqrt{2^j} \cdot n/2^j)\} = min\{|S_k|, O(n/\sqrt{2^j})\}$. At each level, the amount of work to be done is linear in the number of points to consider. The $S_k$'s cover the entire point set so by double summation we get that the runtime is in

$$\begin{aligned} O\left(\sum_j \sum_k \min\left\{|S_k|, n/\sqrt{2^j}\right\}\right) &= O\left(\sum_k \sum_j \min\left\{|S_k|, n/\sqrt{2^j}\right\}\right) \\ &= O\left(\sum_k \sum_{j \le 2\log(n/|S_k|)} |S_k| \;+ \sum_{j > 2\log(n/|S_k|)} n/\sqrt{2^j}\right) \\ &= O\left(\sum_k |S_k|(\log(n/|S_k|)+1)\right) \\ &\subset O\left(n(\mathcal{H}_C(S)+1)\right). \end{aligned}$$ ◀

Note that the proof generalizes easily to any constant dimension $d > 0$. We can apply this theorem to get the following (omitted proofs can be found in the full paper [12]):

▶ **Lemma 28.** *A set of points $S$ can be preprocessed in $O(n(\mathcal{H}(S)+1))$ time so that for any subset $S_k \subset S$, we can test if all points in $S_k$ lie in a common vertical slab containing only points of $S$ of the same color in $O(S_k)$ time.*

Which in turn implies:

▶ **Theorem 29.** *A cross-safety tree on $S$ can be constructed in $O(n(\mathcal{H}(S)+1))$.*

Finally, putting this together with 26 and Theorem 15 we get the main result.

▶ **Theorem 30.** *All points participating in $S$ can be reported in $O(n(\mathcal{H}(S)+1))$ time. In other words, there is an instance-optimal algorithm for this problem in the order-oblivious comparison-based model.*

One thing to note here is that while this guarantees that the algorithm is optimal with respect to any parameter of the instance which does not depend on the order of the input points, it is not immediately obvious that it runs in $O(n \log h)$ time, where $h$ is the number of points to report (we only know that if there is an algorithm in the comparison-based model running within this time bound, then so does ours). The following results shows that the runtime of our algorithm is indeed within this bound.

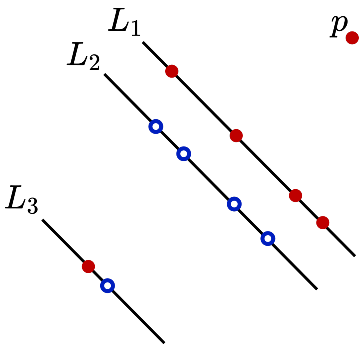


**Figure 5** Type of instance considered in the impossibility proof.

▶ **Theorem 31.** *Let $S$ be an instance of the Reporting participating points problem and let $h$ be the number of points which participate in $S$. Then $n(\mathcal{H}(S)+1) \in O(n \log h)$.*

## 5 Instance-optimality is impossible with linear queries

In the previous section, we have shown that there is a comparison-based algorithm to report participating points which is instance-optimal in the order-oblivious runtime against all comparision-based algorithm solving the problem. We also show that if we "compete" against algorithms which can do queries of the form $x(p) - x(q) \geq y(p) - y(q)$, then such a result is no longer possible, even if we allow our algorithm to be in a much stronger model of computation, such as algebraic computation trees. The full proof, in which we consider instences of the type illustrated in Figure 5, can be found in the full paper [12]. One caveat of this proof is that it relies on special instances with linear degeneracies (three points can be collinear). It is not clear if instance-optimality is possible when restricted to non-degenerate instances.

## References

1 Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *J. ACM*, 64(1), 2017. doi:10.1145/3046673.
2 Pankaj K. Agarwal and Jirí Matousek. Relative neighborhood graphs in three dimensions. *Comput. Geom.*, 2:1–14, 1992. doi:10.1016/0925-7721(92)90017-M.
3 David Bremner, Erik D. Demaine, Jeff Erickson, John Iacono, Stefan Langerman, Pat Morin, and Godfried T. Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. *Discret. Comput. Geom.*, 33(4):593–604, 2005. doi:10.1007/s00454-004-1152-0.
4 Jean Cardinal, Sébastien Collette, and Stefan Langerman. Empty region graphs. *Comput. Geom.*, 42(3):183–195, 2009. doi:10.1016/j.comgeo.2008.09.003.
5 Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discret. Comput. Geom.*, 16(4):361–368, 1996. doi:10.1007/BF02712873.
6 Timothy M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discret. Comput. Geom.*, 16(4):369–387, 1996. doi:10.1007/BF02712874.
7 Xiaomin Chen, János Pach, Mario Szegedy, and Gábor Tardos. Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. *Random Struct. Algorithms*, 34(1):11–23, 2009. doi:10.1002/rsa.20246.
8 Olivier Devillers, Jeff Erickson, and Xavier Goaoc. Empty-ellipse graphs. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms,*

*SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1249–1257. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347218.

9 Ralf Hartmut Güting, Otto Nurmi, and Thomas Ottmann. Fast algorithms for direct enclosures and direct dominances. *J. Algorithms*, 10(2):170–186, 1989. doi:10.1016/0196-6774(89)90011-4.

10 Manabu Ichino and Jack Sklansky. The relative neighborhood graph for mixed feature variables. *Pattern Recognit.*, 18(2):161–167, 1985. doi:10.1016/0031-3203(85)90040-8.

11 J.W. Jaromczyk and G.T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992. doi:10.1109/5.163414.

12 Justin Dallant Jean Cardinal and John Iacono. An instance-optimal algorithm for bichromatic rectangular visibility. *arXiv preprint arXiv:2106.05638*, 2021. arXiv:2106.05638.

13 J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discret. Comput. Geom.*, 7:13–28, 1992. doi:10.1007/BF02187821.

14 David G. Kirkpatrick and John D. Radke. A framework for computational morphology. In Godfried T. TOUSSAINT, editor, *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 217–248. North-Holland, 1985. doi:10.1016/B978-0-444-87806-9.50013-X.

15 David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(1):287–299, 1986. doi:10.1137/0215021.

16 J. Ian Munro, Mark H. Overmars, and Derick Wood. Variations on visibility. In D. Soule, editor, *Proceedings of the Third Annual Symposium on Computational Geometry, Waterloo, Ontario, Canada, June 8-10, 1987*, pages 291–299. ACM, 1987. doi:10.1145/41958.41989.

17 Mark H. Overmars. Connectability problems. In Rolf G. Karlsson and Andrzej Lingas, editors, *SWAT 88, 1st Scandinavian Workshop on Algorithm Theory, Halmstad, Sweden, July 5-8, 1988, Proceedings*, volume 318 of *Lecture Notes in Computer Science*, pages 105–112. Springer, 1988. doi:10.1007/3-540-19487-8_11.

18 Mark H. Overmars and Derick Wood. On rectangular visibility. *J. Algorithms*, 9(3):372–390, 1988. doi:10.1016/0196-6774(88)90028-4.

19 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982. doi:10.1137/0211059.