The Bichromatic Square and Rectangle Problems[⋆]

Jonathan Backer and J. Mark Keil

Department of Computer Science,
University of Saskatchewan,
Saskatoon, SK, Canada
S7N 5C9

Abstract We examine a variant of the maximum empty square (or rectangle) problem: Find an axis-aligned square (or rectangle) that contains as many blue points as possible without containing any red points. Let n be the total number of red and blue points. We solve the bichromatic square problem in $O(n \log n)$ time and $O(n)$ space. We also solve the bichromatic rectangle problem in $O(n \log^3 n)$ time and $O(n \log n)$ space.

1 Introduction

In this paper, we consider instances of the bichromatic problem: Find a figure of a certain shape that contains no red points and as many blue points as possible. Specifically, we present efficient solutions for the the bichromatic axis-aligned square and axis-aligned rectangle problems. For brevity, *all rectangles (and hyperrectangles) are assumed to be axis-aligned* unless otherwise noted.

Recently, Aranov and Har-Peled posed the bichromatic problem for a variety of shapes [3], including squares and rectangles. In their paper, Aranov and Har-Peled present an $(1+\varepsilon)$-approximation algorithm for the bichromatic ball problem that runs in $O(n^{\lceil d/2 \rceil}(\varepsilon^{-2} \log n)^{\lceil d/2+1 \rceil})$ time, for dimensions $d \geq 3$. They conjecture that the bichromatic circle problem is 3SUM-hard.

Eckstein et al. explore the bichromatic problem for hyperrectangles in high-dimensions, which they claim is useful for data-analysis [8]. They show that the bichromatic hyperrectangle problem is NP-hard, for arbitrarily high dimensions. They also present a $O(n^{2d+1})$ time algorithm, for any fixed dimension d. As this runtime grows exponentially in d, it is impractical for high dimensions, which motivates the heuristic approach that they develop. In a subsequent paper, two of the authors, Liu and Nediak, propose a $O(n^2 \log n)$ time and $O(n)$ space algorithm for the two-dimensional bichromatic rectangle problem [12]. We improve their runtime bounds by solving this problem in $O(n \log^3 n)$ time and $O(n \log n)$ space.

The bichromatic problem is a natural variant of the maximum empty shape problem: Find a figure of a certain shape that contains no red points and has as large a volume as possible. The algorithms that we propose for the bichromatic

⋆ This research is supported by the Natural Sciences and Engineering Research Council (NSERC).

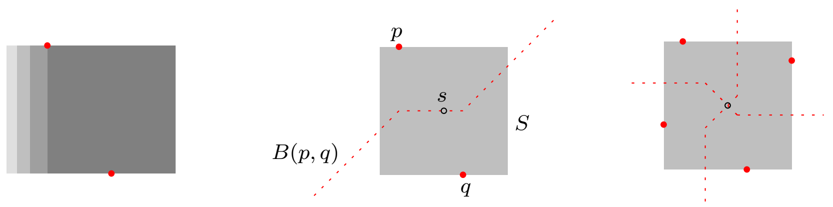
problem use techniques that were successfully applied to the maximum empty square and rectangle problems [4,2,15]. Edmonds et al. describe a solution to the maximum empty rectangle problem that is more suitable for data mining large data sets because it typically requires less space [9].

We note that the bichromatic problem is related to the bichromatic discrepancy problem: Find a figure of a certain shape that maximises the number of blue points contained minus the number of red points contained. Dobkin et al. solve the rectangle bichromatic discrepancy problem in $O(n^2 \log n)$ time [7]. In their paper, they relate various discrepancy problems to problems in machine learning and computer graphics.

2 Maximum Empty Square Problem

We want to find a square that does not contain any red points and contains as many blue points as possible. We call this the bichromatic square (BS) problem. The BS problem is a variant of the maximum empty square (MES) problem: Given a rectangle E, find a maximum area square that is contained in E that does not contain any red points. Solutions to the MES problem have been described in a single line (e.g. “The special case in which a largest empty square is desired has been solved ... using Voronoi diagrams” [4]). We elaborate this approach to the MES problem before addressing the BS problem because it forms the foundation of our approach to the BS problem.

We call a square S *viable*, if it is contained in E and does not contain any red points. We call S *relevant* if it is viable and not properly contained in any viable square. To solve the MES problem, we search the set of relevant squares. Our search is based on one basic observation: If two *adjacent* sides of a viable square do not touch an *obstacle* (a red point or the boundary of E), we can inflate the square while keeping it viable by sliding out the corner common to the obstacle-free sides. Equivalently, two *opposite* sides of a relevant square must touch an obstacle (see Figure 1a).



(a) Range of relevant squares.

(b) Relevant square centred on horizontal segment of bisector.

(c) Relevant square centred on Voronoi point.

Figure 1: Connection to Voronoi diagrams of the red points. Voronoi edges are illustrated with dotted lines.

This opposite side observation establishes a connection to Voronoi diagrams. Note that the L_∞ distance between two points (x_1, y_1) and (x_2, y_2) is $\max(|x_1 - x_2|, |y_1 - y_2|)$. Let p and q be points where a relevant square S touches two obstacles (see Figure 1b). If s is the centre of S, then the L_∞ distance between p and s is exactly the L_∞ distance between q to s. Hence, s lies on the L_∞ bisector of p and q, denoted $B(p, q)$. Typically, $B(p, q)$ is composed of two diagonals and one horizontal or vertical segment. The centre s lies on the non-diagonal portion of $B(p, q)$ because p and q lie on opposite sides of S. Moreover, s lies on an edge of the L_∞-Voronoi diagram because s has more than one nearest obstacle.

Remark 1. The centre s of a relevant square lies on a non-diagonal edge of the L_∞-Voronoi diagram of the obstacles (red points and boundary of E).

Unfortunately, there are degenerate cases where the above observation does not hold (see Figure 1c). When we solve the BS problem, we address these exceptions, which leads to a careful reformulation of Remark 1 as Lemma 3. Our approach to handling these degeneracies also applies to the MES problem. Ignoring these technicalities, the MES problem can be solved by (a) constructing the L_∞-Voronoi diagram (VD) of the obstacles and (b) examining each non-diagonal segment of the VD. Step (a) can be executed in $O(n \log n)$ time and $O(n)$ space [11]. Step (b) can be executed in $O(n)$ time because each Voronoi edge has at most one non-diagonal segment and the VD has $O(n)$ edges [11]. Therefore, the MES problem can be solved in $O(n \log n)$ time and $O(n)$ space.

3 Bichromatic Square Problem

We now consider the bichromatic square problem. In this case, we slightly alter our notion of viability: A square is *viable* if it does not contain any red points. Remark 1 still applies, so our approach to the BS problem is similar: (a) construct the VD of the red points and (b) examine each non-diagonal segment of the VD.

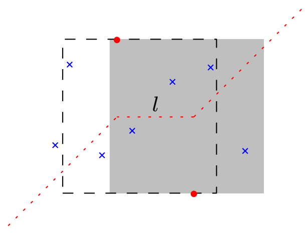
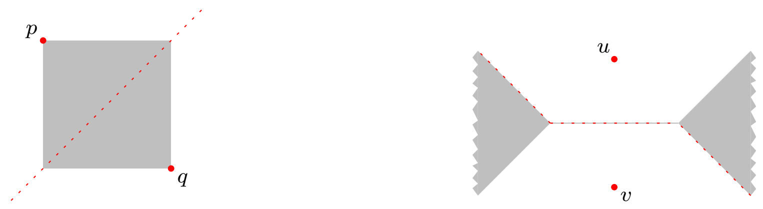


Figure 2: Two relevant squares centred on l. Red points are marked with a $\bullet$ and blue points with a $\times$.

Our BS approach differs from the MES approach in the execution of step (b). Let l be a non-diagonal segment of the VD (see Figure 2). To solve the MES problem, it suffices to calculate the width of the relevant squares that are centred on l. To solve the BS problem, we must find the maximum number of blue points contained in a relevant square that is centred on l. As Figure 2 illustrates, two different squares that are centred on l may contain a different number of blue points. In general, there may be $\Omega(n)$ relevant squares that are centred on l and are important to consider. Later, we show how to examine all relevant squares centred on l with a plane sweep. The structure of the BS problem allows us to sweep over all non-diagonal segments in a total of $O(n \log n)$ time and $O(n)$ space.



(a) Points on opposite corners of a square.

(b) Points with the same x-coordinate.

Figure 3: Bisector degeneracies.

Two types of bisector degeneracies may occur when constructing the VD. Suppose that two points p and q lie on opposite corners of a square (see Figure 3a). In this case, we simplify our analysis by treating $B(p, q)$ as two diagonals separated by a degenerate horizontal segment at the midpoint of p and q. Another degeneracy occurs when two points u and v have the same x-coordinate. In this case, $B(u, v)$ has infinite area (see the shaded region in Figure 3b). When two points have the same x-coordinate, we treat the point with the lower y-coordinate as infinitesimally to the left of the other point. Using this tie-breaking scheme, the bisector in Figure 3b is the dotted curve. We handle the case where two points have the same y-coordinate similarly. By addressing bisector degeneracies in this way, bisectors have the following important property.

Property 1. As a point b traces the bisector $B(p, q)$ between two points, the distance from b to q (equivalently p) changes unimodally, reaching its minimum at the non-diagonal segment.

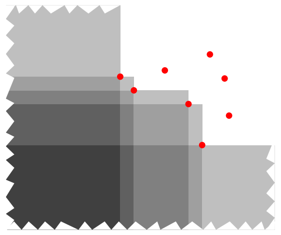
There are exceptions to Remark 1. We will shortly identify $O(n)$ relevant squares that cover all exceptions. First, we describe how to efficiently process these exceptions.

Lemma 1. *Given a set $\mathcal{T}$ of $O(n)$ rectangles (possibly unbounded), we can count the number of blue points inside each rectangle in $\mathcal{T}$ using $O(n \log n)$ time and $O(n)$ space.*

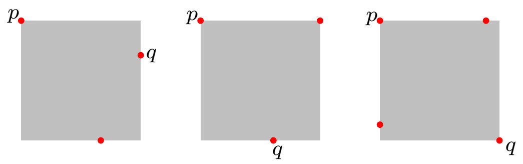
Proof. We sweep a horizontal line from $-\infty$ to ∞. We store the blue points that are below the sweep line in a balanced binary tree sorted by x-coordinate. For each node in the tree, we maintain the number of blue points stored in the subtree rooted at that node. Blue points can be inserted into the tree in $O(\log n)$ time. This simple data structure allows us to count the number of blue points contained in $[a, b] \times (-\infty, y]$ in $O(\log n)$ time, when the sweep line is at y. Two queries of roughly this type suffice to count the number of blue points in a rectangle because $[a, b] \times [c, d] = ([a, b] \times (-\infty, d]) \setminus ([a, b] \times (-\infty, c))$. $\square$

Every unbounded relevant square is not covered by Remark 1. These squares are either full planes, half planes or quarter planes (see Figure 4a). There is at most one relevant full plane, four relevant half planes, and $O(n)$ relevant quarter planes. We can enumerate all relevant quarter planes with four plane sweeps (e.g. we can handle the case illustrated in Figure 4a in one sweep).

A *Voronoi point* occurs where the boundaries of three or more distinct Voronoi cells meet. As Figure 1c illustrates, relevant squares centred at Voronoi points may not be covered by Remark 1. Fortunately, there are only $O(n)$ Voronoi points [11], and they are represented explicitly in the VD. Hence, we can efficiently enumerate them.



(a) Unbounded squares extending to $x = -\infty$ and $y = -\infty$.



(b) Squares with a red corner.

Figure 4: Atypical relevant squares.

A bounded relevant square with a red point on a corner is called a *corner square* (see Figure 4b). We enumerate corner squares in order to reduce the number of cases that we must consider in the Lemma 3. This does not affect the asymptotic performance of our algorithm because there are only $O(n)$ such squares: A corner square is uniquely defined by a red point p and the corner on which it must lie (e.g. upper left, lower left, upper right, or lower right). Let UL(p) denote the unique bounded relevant square with a red point p on its the upper-left corner. As illustrated in Figure 4b, UL(p) has a red point q (not necessarily unique) on the lower right half of its boundary (the bottom and right edges excluding the lower-left and upper-right corners). This observation allows us to enumerate all corner squares efficiently.

Lemma 2. *We can enumerate all $O(n)$ corner squares in $O(n \log n)$ time and $O(n)$ space.*

Proof. We describe how to find all $\mathrm{UL}(p)$ with a red point on its bottom edge. The case where a red point is on the right edge is analogous. The basic idea is to sweep out the bottom edge of $\mathrm{UL}(p)$ until it hits q. The area swept out by the bottom edge forms a cone (see Figure 5). This cone contains the diagonal, but not the left edge. We simultaneously grow all the cones so that the amortised time per cone is low.

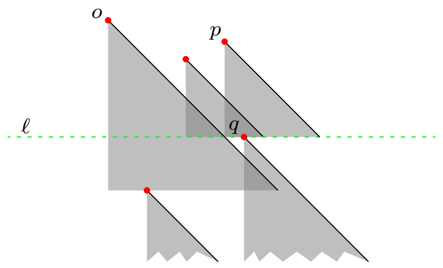


Figure 5: Sweep line approach to identifying corner squares.

We sweep a horizontal line ℓ from ∞ to $-\infty$ while maintaining a list of growing cones that reach ℓ. We store the apex of each such cone in a balanced binary tree sorted by x-coordinate. If two such apexes p and q have the same x-coordinate, we treat the point with smaller y-coordinate as infinitesimally to the left of the other point.

When ℓ passes over a red point q, we update the balanced binary tree: First we remove all of the cones whose base touches q (they can grow no further), and then we add q. The apexes of all the cones that touch q occur consecutively in the sorted list of points maintained by the sweep line. To find the rightmost apex in the consecutive range, we search the tree for the apex p that lies just to the left of q (see Figure 5). Then we walk to the left through the list of apexes until we find the apex o of a cone that does not touch q (see Figure 5). The total time to find all of the cones that touch q is $O(\log n + k)$, where k is the number of such cones. This algorithm takes $O(n \log n)$ because each cone is added and removed once. □

The relevant squares that are not enumerated (i.e. bounded, non-corner squares that are not centred at a Voronoi point) are called *floating*. We say that a side of a relevant square is supported, if its interior contains a red point. The next lemma is a precise reformulation of Remark 1.

Lemma 3. *Each floating square has two opposite supported sides and two opposite unsupported sides. Hence, its centre lies on the interior of a non-diagonal segment of the VD.*

Proof. Let S be the floating square centred at s. Clearly, S has two supported opposite sides because S is bounded and S is not a corner square.

Suppose for contradiction that S has at least three supported sides. We assume without loss of generality that the unsupported side of S (if it exists) is the right side. Note that s lies on the boundary of the VD because S has more than one supported side. Since s is not a Voronoi point, it must lie in the interior of some Voronoi edge e.

Case 1: Suppose that s lies on a diagonal segment d. Then we can slide s parallel to d and increase the distance between s and its nearest red point (see Property 1). However, all four diagonal directions move s closer to the top or bottom of S, which decreases the distance between s and its nearest red point, a contradiction.

Case 2: Suppose that s lies on the interior of a non-diagonal segment l. Then we can slide s parallel to l without changing the distance between s and its nearest red point (see Property 1). However, sliding s up or to the left moves it closer to a supported side of S, which decreases the distance between s and its nearest red point, a contradiction.

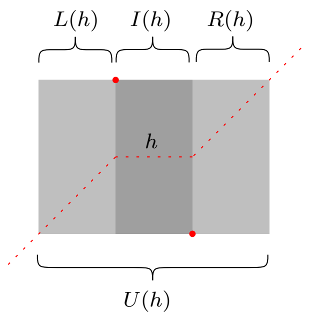
Thus, S has two opposite supported sides and two opposite unsupported sides. We can slide s parallel to the supported sides without changing the distance between s and its nearest red point or changing the number of its nearest red points. Hence, s lies on a segment of the boundary of the VD parallel to the supported sides of S. □

We now describe how to find the floating square that contains the most blue points. We do this with two plane sweeps: one sweep over the horizontal segments of the VD and a second sweep over the vertical segments of the VD. We just describe the first sweep because the second sweep is very similar.

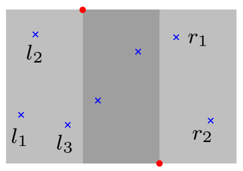
Let h be a horizontal segment of the VD. We call a floating square *h-restricted* if its centre lies on h. To simplify our analysis, we assume without loss of generality that (a) every h-restricted square has the same width (for a fixed h) by splitting segments at Voronoi points and (b) each point on h corresponds to the centre of a floating square by splitting segments at centres of corner squares. There are $O(n)$ resulting horizontal segments because there are $O(n)$ edges in the VD, $O(n)$ Voronoi points, and $O(n)$ corner squares [11].

Any blue point contained in an h-restricted square must lie in the union of all h-restricted squares, denoted $U(h)$ (see Figure 6a). We partition $U(h)$ into three regions: the intersection $I(h)$ of all h-restricted squares, the region $L(h)$ of $U(h) \setminus I(h)$ to the left of $I(h)$, and the region $R(h)$ of $U(h) \setminus I(h)$ to the right of $I(h)$. Note that $L(h)$ is the region swept out by the left side of all h-restricted squares. Two observations lead to an efficient algorithm for finding the floating square containing the most blue points: (a) the blue points in $I(h)$ have no effect on which h-restricted square contains the maximum number of blue points, and (b) the regions swept out by different horizontal segments are disjoint, as stated in the following key lemma.

Lemma 4. *If h and h' are distinct horizontal segments of the Voronoi diagram, then $L(h) \cap L(h') = \emptyset$.*



(a) Partition of swept region.



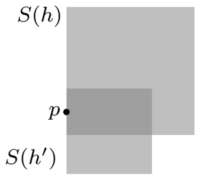
(b) Partition of covered blue points.

Figure 6: Sweep of h-restricted squares.

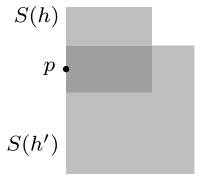
Proof. For contradiction, consider $p \in L(h) \cap L(h')$. Let $S(h)$ be the floating square whose left side touches p and whose centre lies on h, let $s(h)$ be the centre of $S(h)$, and let $t(h)$ denote the y-coordinate of the top side of a square $S(h)$. Without loss of generality, we assume $t(h) \geq t(h')$

Case 1: Suppose that $t(h) = t(h')$. Then $S(h)$ and $S(h')$ have the same upper-left corner. Hence $S(h) = S(h')$ because both squares are relevant. Thus, $s(h) = s(h')$, which implies that $s(h)$ is at an endpoint of h. Hence, either $S(h)$ is a corner square or $s(h)$ is a Voronoi point. So $S(h)$ is not floating, which contradicts that $p \in L(h)$.

Case 2: Suppose that $t(h) > t(h')$. If $S(h)$ is wider than $S(h')$ as illustrated in Figure 7a, then the red points supporting the top of $S(h')$ are contained in $S(h)$, contradicting that $S(h)$ is viable. So suppose that $S(h)$ is no wider than $S(h')$ as illustrated in Figure 7b. If the right side of $S(h)$ is unsupported, we can inflate $S(h)$ by pushing out the lower-right corner of $S(h)$, which contradicts that $S(h)$ is relevant. If the right side of $S(h)$ is supported, then at least three sides of $S(h)$ are supported. This contradicts Lemma 3 because $S(h)$ is a floating square. □



(a) $S(h)$ is wider than $S(h')$.



(b) $S(h)$ is narrower than $S(h')$.

Figure 7: Cases when $t(h) < t(h')$.

Let $\langle l_1, l_2, \ldots \rangle$ be the blue points in $L(h)$ sorted by increasing x-coordinate (see Figure 6b). Similarly, let $\langle r_1, r_2, \ldots \rangle$ be the blue points in $R(h)$ sorted by increasing x-coordinate. We can simultaneously walk through $\langle l_i \rangle$ and $\langle r_i \rangle$ to find the h-restricted square containing the most blue points in time linear in the sum of the sizes of the two lists.

Lemma 5. *We can compute all of the sequences $\langle l_i \rangle$ associated with all of the horizontal segments in $O(n \log n)$ time and $O(n)$ space.*

Proof. We can compute all $\langle l_i \rangle$ in a single plane sweep. While we sweep a vertical line from $-\infty$ to ∞, we maintain a sorted list in a balanced binary tree of the topmost point where some $L(h)$ intersects the sweep line. Associated with each point in the tree is a reference to its corresponding rectangle $L(h)$. Adding and removing a rectangle is an $O(\log n)$ operation. When we hit a blue point b as we sweep right, we find the y-coordinate in the binary tree immediately above b in $O(\log n)$ time. If b is contained in the corresponding box, we add it to that box's list. $\square$

The next theorem follows.

Theorem 1. *We can solve the bichromatic square problem in $O(n \log n)$ time and $O(n)$ space.*

Proof. As described earlier, we can identify the $O(n)$ non-floating squares in $O(n \log n)$ time and $O(n)$ space (e.g. see Lemma 2). By Lemma 1, we can count the number of blue points in each non-floating square in $O(n \log n)$ time and $O(n)$ space.

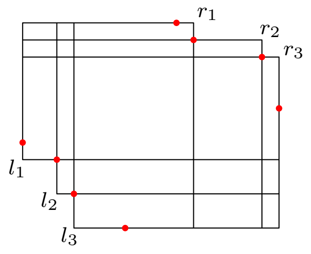
We now describe how to handle the floating squares. We just consider squares restricted to horizontal segments because the vertical segment case is similar. First, we count the number of blue points in $I(h)$ for every horizontal segment h in $O(n \log n)$ time and $O(n)$ space by Lemma 1. Then we compute $\langle l_i \rangle$ and $\langle r_i \rangle$ for each horizontal edge in $O(n \log n)$ time and $O(n)$ space by Lemma 5. Finally, we sweep across each horizontal segment h to find the h-restricted square containing the most blue points. This last step takes a total of $O(n \log n)$ time and $O(n)$ space because each blue point belongs to at most one sequence $\langle l_i \rangle$ by Lemma 4. $\square$

4 Bichromatic Rectangle Problem

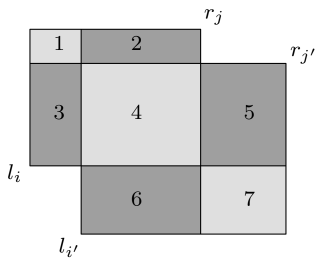
In this section, we show how to find a rectangle that contains no red points and contains as many blue points as possible. We call this the bichromatic rectangle (BR) problem. Our approach to solving the BR problem is a direct application of the techniques used to solve the maximum empty rectangle (MER) problem: Given a set of points red points and a bounding rectangle E, find the largest area rectangle that is contained in E and contains no red points.

We say that a rectangle R is *viable* if it is contained in E and contains no red points. We say that R is *relevant* if it is viable and not properly contained in any viable rectangle. When looking for a MER, we can restrict our attention to relevant rectangles because every MER is relevant. A common approach to finding a MER is to enumerate all relevant rectangles. This can be accomplished in $O(n \log n + k)$ time and $O(n)$ space [15], where n is the number of red points and k is the number of relevant rectangles. This approach works well in practice because the expected value of k is $O(n \log n)$ under modest assumptions about the red point set [14]. However, in the worst case $k \in \Theta(n^2)$.

The MER problem and BR problem are closely related: Let E be a rectangle enclosing the red and blue points. If R is a solution to the BR problem, then $R \cap E$ is also a solution. Hence, we can solve the BR problem by enumerating all relevant rectangles and counting the blue points in each one. Counting the blue points in a rectangle is an orthogonal range counting query. Using $O(n \log n)$ preprocessing time and $O(n \log n)$ space, we can answer such queries in $O(\log n)$ time by using a range tree with fractional cascading [6]. The resulting algorithm takes $O(k \log n)$ time (because $k \in \Omega(n)$) and $O(n \log n)$ space. This direct approach is asymptotically faster in expectation (but not worst case) than the $\Theta(n^2 \log n)$ time algorithm proposed by Liu and Nediak [12].



(a) Every l_i and r_j define a relevant rectangle.



(b) McKenna's Lemma.

Figure 8: Structure in the worst case behaviour.

The number of relevant rectangles is $\Theta(n^2)$ in the worst case. Figure 8a illustrates one arrangement of points that generates $\Theta(n^2)$ relevant rectangles: The red points form two well separated chains of approximately equal length, where the points on each chain decrease in y-coordinate as they increase in x-coordinate. Consecutive points on a chain define a corner: a lower-left corner for consecutive points on the lower chain and an upper-right corner for consecutive points on the upper chain. We label the corners on the lower chain $l_1, l_2, \ldots$ from top to bottom. Similarly, we label the corners on the upper chain $r_1, r_2, \ldots$ from top to bottom. We use the term *corner rectangle* to refer to the relevant rectangle with a l_i as a lower-left corner and a r_j as an upper-right corner. Chazelle et al. [4] reduce the MER problem to finding the largest empty corner rectangle (LECR). Their reduction works by first explicitly considering all of the

$O(n)$ relevant rectangles not considered by the LECR problem. Hence, this same approach reduces the BR problem to finding the corner rectangle with the most blue points (BCR problem).

Aggarwal and Suri provide an elegant solution to the LECR problem [2]. They exploit the fact that the areas of the rectangles in this subproblem are related via a simple inequality. Let $A_{i,j}$ denote the area of the corner rectangle associated with l_i and r_j. Then $A_{i,j} + A_{i',j'} > A_{i,j'} + A_{i',j}$, for $i < i'$ and $j < j'$ [13]. This inequality implies that we can find the LECR by comparing the areas of just $O(n)$ rectangles by using the monotone matrix searching technique of Aggarwal et al. [1]. The LECR problem is used to merge the results of subproblems in a divide-and-conquer approach to the MER problem. So although the LECR problem can be solved in $O(n)$ time and $O(n)$ space, solving the MER problem requires $O(n \log^2 n)$ time and $O(n)$ space.

We note that if $A_{i,j}$ denotes the number of blue points in the corner rectangle associated with l_i and r_j, then the inequality $A_{i,j} + A_{i',j'} > A_{i,j'} + A_{i',j}$ still holds. This can be verified by considering the seven regions associated with the relevant corner rectangles illustrated in Figure 8b. Let R_k denote the number of blue points in the k^{th} region. Then $A_{i,j} = R_1 + R_2 + R_3 + R_4$. Using this style of counting, it can be verified that $A_{i,j} + A_{i',j'} = A_{i,j'} + A_{i',j} + R_1 + R_7$. Hence, $A_{i,j} + A_{i',j'} > A_{i,j'} + A_{i',j}$ because $R_1 \geq 0$ and $R_7 \geq 0$. Calculating $A_{i,j}$ is an orthogonal range counting query. With appropriate preprocessing, $A_{i,j}$ can be computed in $O(\log n)$ time. Hence, the BCR problem can be solved in $O(n \log n)$ time and $O(n \log n)$ space. By combining this method of solving the BCR problem with the Aggarwal and Suri divide-and-conquer approach to the MER problem [2], we obtain the following theorem.

Theorem 2. *A rectangle containing no red points and as many blue points as possible can be found in $O(n \log^3 n)$ time and $O(n \log n)$ space.*

5 Discussion

In this paper, we solve the bichromatic square problem in $O(n \log n)$ time and $O(n)$ space. This algorithm is simple and uses no data structure more complex than a balanced binary tree. We also solve the bichromatic rectangle problem in $O(n \log^3 n)$ time and $O(n \log n)$ space. Aggarwal and Suri's solution to the maximum empty rectangle problem requires $O(n \log^2 n)$ time and $O(n)$ space [2]. Removing the additional logarithmic factors from our adaptation of their approach is an open problem.

There is a substantial interest in the problem of colour range queries (see [10] for a survey): Given a set of coloured points, preprocess the points to efficiently count the number of different colours contained in a series of query ranges. We note that the techniques in this paper can be used to find a square or rectangle that contains as many different colours as possible with roughly the same time and space bounds (i.e. we add a new term for the number of different colours).

It is natural to ask if our results can be generalised to higher dimensions. It is straightforward to solve the bichromatic cube problem in $O(n^3 \log n)$ time using the observation that relevant cubes typically have two opposite sides that touch a red point. The bichromatic three-dimensional hyperrectangle problem can be solved in $O(n^3 \log^2 n)$ time by enumerating all relevant hyperrectangles [5] and counting the number of blue points in each one. In both cases, breaking the cubic runtime barrier is an interesting open problem.

References

1. A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.
2. A. Aggarwal and S. Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the third annual symposium on Computational geometry*, pages 278–290. ACM New York, NY, USA, 1987.
3. B. Aronov and S. Har-Peled. On Approximating the Depth and Related Problems. *SIAM Journal on Computing*, 38(3):899–921, 2008.
4. B. Chazelle, R.L. Drysdale, and D.T. Lee. Computing the largest empty rectangle. *SIAM Journal on Computing*, 15:300, 1986.
5. A. Datta and S. Soundaralakshmi. An efficient algorithm for computing the maximum empty rectangle in three dimensions. *Information Sciences*, 128(1-2):43–65, 2000.
6. M. de Berg, O. Schwartskopf, M. Overmars, and M. van Kreveld. *Computational geometry.* Springer-Verlag Berlin, 2000.
7. D.P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *Journal of Computer and System Sciences*, 52(3):453–470, 1996.
8. J. Eckstein, P.L. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002.
9. J. Edmonds, J. Gryz, D. Liang, and R.J. Miller. Mining for empty spaces in large data sets. *Theoretical Computer Science*, 296(3):435–452, 2003.
10. P. Gupta, R. Janardan, and M. Smid. Computational geometry: generalized intersection searching. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, chapter 64, pages 1–17. Chapman & Hall/CRC, 2005.
11. D.T. Lee and C.K. Wong. Voronoi Diagrams in $L_1 (L_\infty)$ Metrics with 2-Dimensional Storage Applications. *SIAM Journal on Computing*, 9:200, 1980.
12. Y. Liu and M. Nediak. Planar case of the maximum box and related problems. In *Proceeding of the Canadian Conference on Computational Geometry*, pages 11–13, 2003.
13. M. McKenna, J. O'Rourke, and S. Suri. Finding the largest rectangle in an orthogonal polygon. In *Proceedings of the 23rd Allerton Conference on Communication, Control, and Computing*, pages 486–495, 1985.
14. A. Naamad, D.T. Lee, and W.L. Hsu. Maximum empty rectangle problem. *Discrete Appl. Math.*, 8(3):267–277, 1984.
15. M. Orlowski. A new algorithm for the largest empty rectangle problem. *Algorithmica*, 5(1):65–73, 1990.