Minimum separating circle for bichromatic points in the plane

Steven Bitner and Yam Cheung and Ovidiu Daescu
Department of Computer Science
The University of Texas at Dallas
Richardson, TX USA
Email: {spb063000,ykcheung,daescu}@utdallas.edu

***Abstract*—Consider two point sets in the plane, a red set of size n, and a blue set of size m.**

In this paper we show how to find the minimum separating circle , which is the smallest circle that contains all points of the red set and as few points as possible of the blue set in its interior. If multiple minimum separating circles exist our algorithm finds all of them. We also give an exact solution for finding the largest separating circle that contains all points of the red set and as few points as possible of the blue set in its interior. Our solutions make use of the farthest neighbor Voronoi Diagram of point sites.

***Keywords*-set separation; bichromatic separation; circular separation; Voronoi diagram**

I. Introduction

Let $\mathcal{R}$ and $\mathcal{B}$ be two sets of points in the plane ($\mathbb{R}^2$), of size $|\mathcal{R}| = n$ and $|\mathcal{B}| = m$, respectively. We call $\mathcal{R}$ the red set and $\mathcal{B}$ the blue set.

We say that $\mathcal{R}$ cannot be separated from $\mathcal{B}$ by a circle if there exists no circle that contains all points of $\mathcal{R}$ and has no point of $\mathcal{B}$ in its interior. Notice that it is possible that $\mathcal{B}$ can be separated from $\mathcal{R}$ even though $\mathcal{R}$ cannot be separated from $\mathcal{B}$.

Assume that $\mathcal{R}$ cannot be separated from $\mathcal{B}$ by a circle. Let $\mathcal{S}$ denote the set of circles such that each circle in $\mathcal{S}$ contains all points in $\mathcal{R}$ while minimizing the number of points from $\mathcal{B}$ contained within its interior. In this paper we show how to find the smallest circle in $\mathcal{S}$. We denote this circle by $C_{\mathcal{B}}(\mathcal{R})$ and call it the *minimum separating circle*. If multiple minimum separating circles exist our algorithms find all of them.

It is easy to see that this problem is different from that of finding the minimum enclosing circle for a set of points. The minimum separating circle, as illustrated in Figure I, may be much larger than the minimum enclosing circle.

The problem we study arises in military applications, for example for wireless communication amongst a red force while trying to minimize the chances of transmissions being intercepted by the blue force. Another application could be in determining the location and amount of explosives needed to destroy an enemy force. The amount of explosives can determine the blast radius. If we model the enemy forces as the red set and the unarmed civilian locations as the blue set we can find the best location and amount of explosives to drop in order to destroy all enemy forces while minimizing civilian casualties.

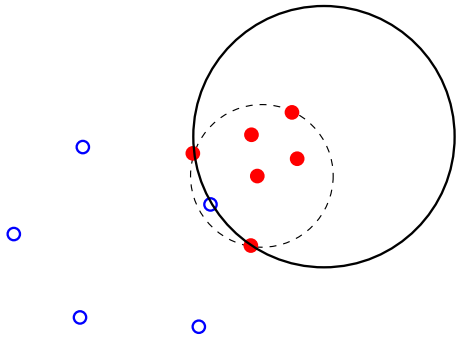

Figure 1. Minimum red enclosing (dashed) and red-blue separating (solid) circles.

A great deal of work has been done on the separation of point sets with geometric objects.

In [13], the author gave the first $O(n \log n)$ algorithm for solving the largest empty circle problem in the plane, which given a set S of n points asks to find the largest circle such that its interior contains no points from S and the center of the circle lies within the convex hull $CH(S)$ of S.

A related problem is the k-enclosing circle problem where given a set S of n points in the plane, we seek to find the minimum radius circle that contains at least k points from S. A long standing result from [9] solves this problem in expected $O(n \log n + nk)$ time using $O(nk)$ space and in $O(n \log n + nk \log k)$ time using linear space. This time was improved to $O(nk)$ in [8] using $O(n + k^2)$ space. Interestingly, this result beats the lower bound of $\Omega(n \log n)$ when k is small.

In [12], they solve the problem of finding the minimum radius circle that contains a set of n points in the plane, where the center of the circle is required to lie on a query line segment. Using $O(n \log n)$ and $O(n)$ preprocessing time and space, the algorithm is able to return the minimum radius circle with its center on the query line segment in $O(\log^2 n)$ time. The query time was improved in [5] to $O(\log n)$ while maintaining the same preprocessing bounds.

In [1], given n points of k different types (or colors) they find the minimum area or perimeter axis parallel

978-0-7695-4112-9/10 $26.00 © 2010 IEEE
DOI 10.1109/ISVD.2010.14

rectangle that contains at least one point from each set in $O(n(n-k)\log^2 k)$ time. They also find the narrowest unbounded region defined by two parallel lines in the plane spanning all k types in $O(n^2\alpha(k)\log k)$ time, where $\alpha(k)$ is the slowly growing inverse of the Ackermann function.

Bichromatic separability problems have also been widely studied, although typically in a setting which we shall refer to as purely separable. By purely separable, we mean that the two sets ($\mathcal{R}$ and $\mathcal{B}$) can be separated by some object (line, circle, rectangle, etc.) where all elements in one set, say $\mathcal{R}$, lie on one side of the boundary while all elements of $\mathcal{B}$ lie on the other side.

A near linear time algorithm to report whether two sets of points are purely separable by a prism, as well as near quadratic results for separability of point sets by a slab or wedge are given in [2]. That paper also gives a near cubic time for finding the separation of two sets of points by a double wedge.

In [4], the authors give an optimal $\Theta(n\log n)$ algorithm for a circular separation problem. Specifically, given two sets of line segments of size $O(n)$ that intersect only at their endpoints, they find all largest circles whose interior contains one set of line segments and no segments from the other set.

The problem of separating two input sets $\mathcal{R}$ and $\mathcal{B}$ of points in the plane using the smallest circle was solved in $O(|\mathcal{R}||\mathcal{B}|)$ time and space in [7]. This result was improved upon in [11] where the time and space requirements are $O(|\mathcal{R}|+|\mathcal{B}|)$. They also prove a set of properties of any finite largest separting circle that allow them to find the largest separating circle in optimal $O(n\log n)$ time.

In [3] they address the problem of a linear separator of two point sets in the plane. In their paper however, they do not address a purely separable set, but instead they seek to define metrics for the error of the separator based on the number of violations. They solve the problem for four such metrics. One is the MinMax metric which looks to minimize the maximum distance from a misclassified point to the separating line. They offer a $\Theta(n\log n)$ solution in the plane. Two others are the MinSum (resp. MinSum^2) criterion, which minimizes the sum (resp. sum of squares) of the Euclidian distances between the separator and all misclassified points. For the planar MinSum version, they give an $O(n\log n + n^{4/3}\log^{1+\epsilon} n)$ algorithm, as well as a randomized algorithm with an expected time of $O(n\log n + n^{4/3})$. They give an algorithm that solves the planar MinSum^2 problem in quadratic time. Lastly, they solve the MinMis problem, in which the goal is to minimze the number of misclassified points, in quardatic time in the plane.

We emphasize that our algorithm is intended to be used in the case where $\mathcal{R}$ cannot be purely separated from $\mathcal{B}$, which can be decided in linear time [11]. Thus, we address a more general problem, where we allow the smallest possible number of (blue) outliers within the circle containing the red points.

II. Results

We present two algorithms for finding the minimum separating circle. The first algorithm takes $O(nm\log m + n\log n)$ time, uses $O(n+m)$ space, and is based on a sweep of the farthest point Voronoi diagram of the red points. The second algorithm takes $O(nm) + O^*(m^{1.5})$ time using $O(n) + O^*(m^{1.5})$ space, and is based on circular range queries, where the notation $O^*()$ ignores a polylogarithmic factor.

We then give an $O\left(nm\log m + k\,(n+m)\log\,(n+m)\right)$ time and $O(n+m)$ space algorithm for finding the largest separating circle, where k is the number of separating circles containing the smallest possible number of points from $\mathcal{B}$.

The layout of the paper is as follows. In Section III we prove a few properties of the minimum separating circle. Then, in Section IV we use additional information related to the farthest neighbor Voronoi diagram to achieve our first result. In Section V we describe the second algorithm for finding the minimum separating circle. In Section VI, we discuss briefly how the first minimum separating circle algorithm can be used to find the largest separating circle. We conclude the paper in Section VII.

III. Preliminaries

We start by discussing some properties of the minimum separating circle. For easiness of exposition we assume that the points in $\mathcal{R}\cup\mathcal{B}$ are in general position, that is, no four of them lie on the same circle. However, all of our arguments hold without this assumption.

Let $CH(\mathcal{R})$ denote the convex hull of the points in $\mathcal{R}$. Notice that the blue points within $CH(\mathcal{R})$ are included in any circle enclosing the red points and can be counted in $O(n\log n + \min\{m\log n, m\log m\})$ time in a preprocessing step. Thus, from now on, we assume the blue points are outside $CH(\mathcal{R})$. As it will become clear later, in Section IV, only the red points that are vetices of $CH(\mathcal{R})$ matter. Thus, the points of $\mathcal{R}$ interior to $CH(\mathcal{R})$ can also be eliminated in the preprocessing step.

Lemma 1. *The smallest separating circle must pass through at least two points from $\mathcal{R}$.*

Proof: We make the proof by contradiction. Assume that $C_{\mathcal{B}}(\mathcal{R})$ does not pass through two or more points in $\mathcal{R}$. This means that it is either defined by two blue points and one red point, or by three blue points. First, we analyze the case when one red point lies on the circle. Consider the red point that lies on C and let us call that point r_i. If we move the center of $C_{\mathcal{B}}(\mathcal{R})$ directly toward r_i some small distance ϵ, then we get a new circle C' which is smaller than $C_{\mathcal{B}}(\mathcal{R})$. We choose ϵ to be a distance small enough so that the new circle C' still contains all points from $\mathcal{R}$. Since, C' is smaller than $C_{\mathcal{B}}(\mathcal{R})$ we have obtained a contradiction.

A similar approach can be applied in the case where all three points defining $C_{\mathcal{B}}(\mathcal{R})$ are members of $\mathcal{B}$. We simply use one of the three blue points, call it b_i, in the same manner used for r_i. ■

It follows that the minimum separating circle can be broken into two types. The first type (I) is the smallest enclosing circle of $\mathcal{R}$, which can be found in linear time [10]. The second type (II) is a circle which passes through two points from $\mathcal{R}$ and one point from $\mathcal{B}$.

Lemma 2. *There are n minimum separating circles in the worst case.*

Proof: Firstly, we establish the upper bound. We shall provide a proof by contradiction. Assume that there exist l minimum separating circles, where $l > n$. The minimum enclosing circle for a set of points has its center c on an edge of the farthest point Voronoi diagram (FVD) [12]. Let $FVD(\mathcal{R})$ denote the farthest point Voronoi diagram of $\mathcal{R}$. As the center of an enclosing circle of $\mathcal{R}$ moves away from c, the size of the circle increases monotonically along the branches of the tree. Clearly, we can not have more than one circle center on any unbounded edge of $FVD(\mathcal{R})$. Since there are n unbounded edges, we can have at most n circle centers that lie on unbounded edges of $FVD(\mathcal{R})$. This implies that any additional centers must lie on internal, bounded edges of $FVD(\mathcal{R})$. However, if a minimum separating circle has a center c' on an internal edge of the $FVD(\mathcal{R})$ then the points in the subtree rooted at c' and not containing c can not be the center of a minumum separating circle, since such circles would have a larger radius, a contradiction.

A lower bound example is shown in Figure 2. If the n red points lie at the vertices of a regular polygon, with the same number of blue points at some distance δ from the midpoint of each edge of the polygon, then we can easily construct n smallest separating circles that contain all of the other blue points near the edges, except the one being used to define the circle. Slight perturbations along the arcs of the circles allow us to maintain this example without the condition of general position for all points. ■

Using the lemmas above, a minimum separating circle can be found in $O\left(n^2 m\left(m+n\right)\right)$ time by checking all $m\binom{n}{2}$ possible choices of vertices defining a circle of type II. We start with the minimum enclosing circle for $\mathcal{R}$ as our best result. In each iteration, we must verify that the circle contains all red points. This can be done in $O(n)$ time. Then, in $O(m)$ time we count the number of blue points contained within the circle. We compare this number with the previous best result. If the number of blue points is smaller in this iteration, then this circle becomes our new best result encountered thus far. If the number of blue points is larger, then this circle is not a candidate for $C_{\mathcal{B}}(\mathcal{R})$. If the number of blue points is equal, then we compare the size of the previous best result with the size of the currently defined circle. If the current circle is smaller, then it becomes the new best result, otherwise we move to the next iteration.

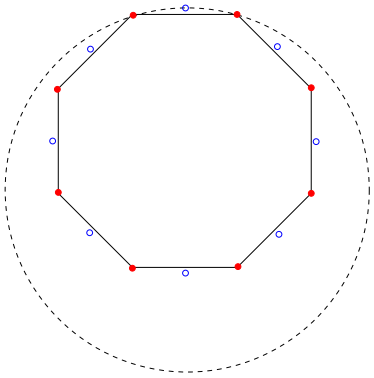

Figure 2. An example of a configuration of two sets with $O(n)$ smallest separating circles

IV. A First Algorithm for Minimum Separating Circle

We begin with the following lemma.

Lemma 3. *The center of a minimum separating circle lies on an edge of the farthest neighbor Voronoi diagram of $\mathcal{R}$.*

Proof: From Lemma 1, we know that $C_{\mathcal{B}}(\mathcal{R})$ must pass through at least two red points. Given the fact that the locus of the centers of enclosing circles of $\mathcal{R}$ through two red points defines an edge of $FVD(\mathcal{R})$ and the center of an enclosing circle of $\mathcal{R}$ through three red points defines a vertex of $FVD(\mathcal{R})$, the proof follows. ■

Using Lemma 3 we can compute $C_{\mathcal{B}}(\mathcal{R})$ by executing a sweeping procedure on the edges of $FVD(\mathcal{R})$, and obtain the following results.

Theorem 1. *$C_{\mathcal{B}}(\mathcal{R})$ can be computed in $O(nm\log m + n\log n)$ time.*

Proof: For an edge e_{ij} of $FVD(\mathcal{R})$, let r_i and r_j be the red points in $\mathcal{R}$ defining e_{ij}. That is, for any point $c \in e_{ij}$, the smallest enclosing circle of $\mathcal{R}$ centered at c passes through r_i and r_j. Notice that the radius of the circle increases as we move along e_{ij} from one endpoint to the other one. The line segment $\overline{r_i r_j}$ divides the enclosing circle into two regions. Let A be the region that decreases as the center of the circle moves on e_{ij} and let B be the region that increases. See Figure 3 for an illustration.

We initialize our sweeping algorithm on e_{ij} by constructing an enclosing circle C of $\mathcal{R}$ which passes through r_i and r_j and has the smallest possible area, that is, we start at the nearest point to r_i and r_j on e_{ij}. Let $c \in e_{ij}$ be the center of C. The point c is almost always a vertex of $FVD(\mathcal{R})$ unless the smallest enclosing circle for $\mathcal{R}$ is defined by r_i and r_j. To handle the degenerate case, we can either run the sweeping procedure twice in both directions along e_{ij} or

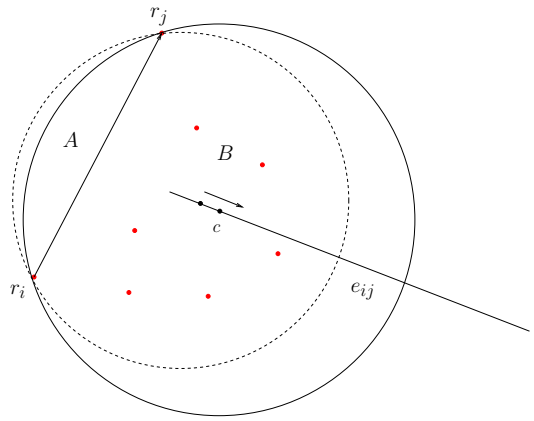

Figure 3. Expanding the enclosing circle by moving its center c on e_{ij}.

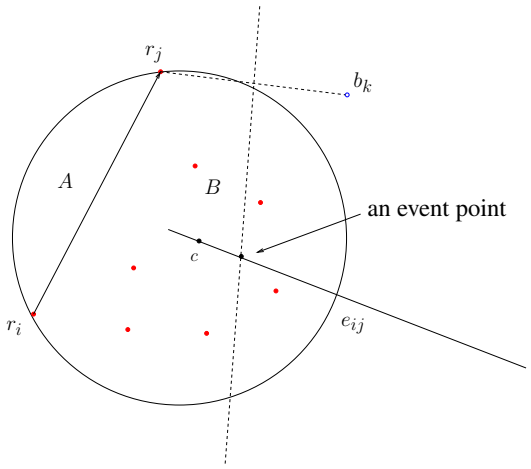

Figure 4. Finding an event point on edge e_{ij}.

divide e_{ij} into two separate edges by treating c as a vertex. We grow C by sweeping c along e_{ij} (see Figure 3 for an illustration). A point $E \in e_{ij}$ is an event point if, when c sweeps through E, the circle C sweeps through a blue point. We compute the number of blue points enclosed by C at each event point. The sweeping procedure on e_{ij} terminates when all event points have been analyzed.

Correctness: C remains an enclosing circle of $\mathcal{R}$ during the process, since r_i and r_j are the farthest red points to c. By executing the sweeping procedure on each edge, we compute the number blue points enclosed by every enclosing circle of $\mathcal{R}$ defined by two red points and one blue point, as well as $C(\mathcal{R})$.

Time: The construction of $FVD(\mathcal{R})$ takes $O(n \log n)$ time. For the sweeping procedure on an edge $e_{ij} \in FVD(\mathcal{R})$, all event points can be computed in $O(m)$ time by finding the intersection between e_{ij} and the perpendicular bisector of r_i and each blue point (see Figure 4). Since the bisector of two points is a straight line, each blue point defines at most one event point on each edge of $FVD(\mathcal{R})$. We need to sort all event points on e_{ij} with respect to their order along e_{ij}, which takes $O(m \log m)$ time. We can compute the number of blue points enclosed by C at the beginning of the sweep by brute force, in $O(m)$ time. Observe that during the sweeping procedure region A contracts while region B expands. If c sweeps through an event point defined by a blue point b_k on A (resp. B) b_k is removed from (resp., enclosed by) C. Hence, the number of blue points enclosed by C at the current event point can be computed in constant time by decrementing or incrementing the number of blue points enclosed by C at the previous event point. The sweeping procedure, including sorting, takes $O(m \log m)$ time for each edge. The overall algorithm runs in $O(nm \log m + n \log n)$ time. ■

Naturally, we would like to impose a bound, be it ammortized or for each edge of $FVD(\mathcal{R})$, on the number of possible event points. A small bound would allow us to save time in our sorting step and reduce the runtime of our algorithm. The next lemma states, however, that we cannot hope for better in the worst case.

Lemma 4. *The number of event points is $\Omega(nm)$ in the worst case.*

Proof: Recall that $CH(\mathcal{R})$ denotes the convex hull of the points in $\mathcal{R}$. Refer to Figure 5 for an illustration of our following description. We start our worst case construction by analyzing the upper and lower chains of $CH(\mathcal{R})$. As a circle which remains tangent to the same two red points grows toward infinity, the interior of that circle approaches the halfspace defined by the line which passes through those two points. We can grow the circle defined by each pair of points on the upper chain of $CH(\mathcal{R})$ and eventually we will contain all blue points that lie in the halfspace containing $\mathcal{R}$. We do this for all edges on the upper chain, and then find the intersection of all those halfspaces, which is a superset of the grey region in Figure 5.

An edge of $FVD(\mathcal{R})$ defined by two points $r_i, r_j \in \mathcal{R}$ that are neighbors on the convex hull will have an endpoint at the center of the circle which passes through both of these points as well as one other point in $\mathcal{R}$, and which contains all other points from $\mathcal{R}$. The edge is unbounded at the other end, since the infinte circle passing through r_i and r_j contains all points in $\mathcal{R}$. If we begin our sweep for a given unbounded edge of $FVD(\mathcal{R})$ at the vertex with all blue points lying outside the corresponding circle (called *initial* circle) and continue outward, we will eventually encounter all blue points. With this in mind, we can choose all blue points to be outside of all initial circles, yet within the intersection of the halfspaces defined above, as depicted by the grey region of Figure 5.

By this example, we show that for each unbounded edge of $FVD(\mathcal{R})$, we can have $\Omega(m)$ event points, resulting in

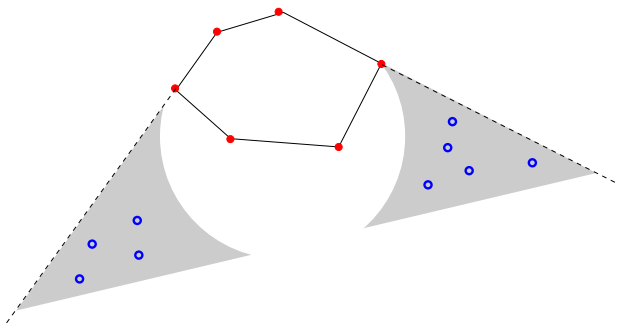

Figure 5. An illustration of the upper bound for the number of event points

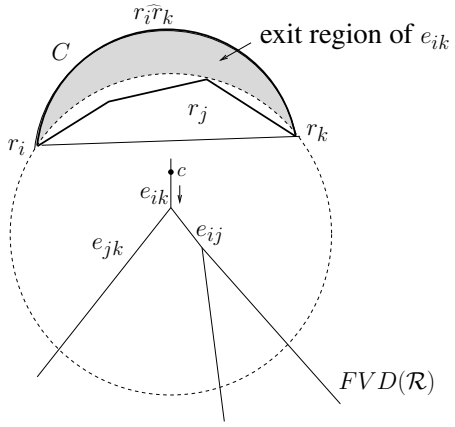

Figure 6. An illustration of the exit region associated with a Voronoi edge.

$\Omega(nm)$ event points in the worst case.

One may argue that for the case in Figure 5 we should not perform the sweep, since the red and blue sets can clearly be separated by a circle. However, we can simply place some of the blue points in the portion of the circle passing through r_i and r_j that decreases in size as the size of the circle increases. The blue points from that region would cause removal of event points, making the sweep necessary. We can place $O(m)$ points in each portion of the circle in order to make this argument complete. ■

V. A Second Algorithm for Minimum Separating Circle

The natural question at this point is whether or not we must consider all possible event points. The answer is no. Given that we start with the smallest possible circle for a given edge of $FVD(\mathcal{R})$ and strictly increase the radius of that circle looking for events where we have fewer points from $\mathcal{B}$ in the interior of the circle, we only need to check when a blue point exits the circle. Thus, we don't need to analyze all possible event points for a given edge, but rather only the **exit** event points. Clearly, several blue points may be exit event points for a given edge of $FVD(\mathcal{R})$. We have to check those event points since there may be **enter** event points in between any of them. We note that our solution in the previous section relies on a sweeping algorithm that maintains the number of blue points contained in a circle in constant time per update. Without such a sorted event point sweep we can no longer calculate the number of blue points in a circle in constant time. However, using the following lemma, we can afford more expensive queries while reducing the overall running time of our algorithm, in some cases.

Lemma 5. *A point from $\mathcal{B}$ is an exit event point for at most one edge of $FVD(\mathcal{R})$*

Proof: Let circle C be an enclosing circle of $\mathcal{R}$ which passes through two red points r_i and r_k. Let e_{ik} be the Voronoi edge in $FVD(\mathcal{R})$ associated with r_i and r_k. See Figure 6 for an illustration. If we perform the sweeping procedure described in the previous section by moving the center of C along e_{ij} and away from $\overline{r_ir_k}$, the radius of C strictly increases. We define the region where points are excluded from C during the sweeping procedure as the *exit region*, $Exit(e_{ik})$, associated with e_{ik}. That is, $Exit(e_{ik})$ is the region swept by a circular arc between r_i and r_k on C when the center of C moves along e_{ik}. It is easy to see that C will sweep over each point in $Exit(e_{ik})$ exactly once. Hence, each point from $\mathcal{B}$ has at most one exit event point on each Voronoi edge.

To prove the lemma, it is sufficient to show that all exit regions are piecewise disjoint. We say that two exit regions are *disjoint* if they share no interior point. We view $FVD(\mathcal{R})$ as a tree with the center of the minimum enclosing circle $C(\mathcal{R})$ of $\mathcal{R}$ as its root. Let e_{ik} be an edge incident to the root with two descent edges e_{ij} and e_{jk} (see Figure 6) for an illustration. Observe that $Exit(e_{ik})$ is disjoint from all exit regions of its descendant edges, which must lie within the region bounded by the circular arc $\widehat{r_ir_jr_k}$ and the chain of $CH(\mathcal{R})$ from r_i to r_k that contains r_j. Moreover, the exit regions of its two direct descendant edges are also disjoint, since $Exit(r_ir_j)$ is bounded by $\widehat{r_ir_jr_k}$ and the subchain of $CH(\mathcal{R})$ connecting r_i and r_j, while $Exit(r_jr_k)$ is bounded by $\widehat{r_ir_jr_k}$ and $\overline{r_jr_k}$. Using a similar argument, it follows that the exit regions of edges which are not descendant edges of e_{ik} are disjoint from the exit regions associated with e_{ik} as well as its descendant edges. ■

Given that there are at most m exit event points, we use the following approach. For each point $b_i \in \mathcal{B}$, we find the edge e_{jk} in $FVD(\mathcal{R})$ for which it is an exit event point, if such an edge exists. We then perform a circular range counting query for the query circle that passes through b_i, r_j and r_k. We compare this count with the best count found so far. If the count is equal, then we store the circle with the smaller radius as the current solution. If the count is less than in the current solution, then this becomes the new

solution. If the count is higher than the current solution, we move to the next exit event point for analysis.

We now give a more datailed analysis of this approach. For each point $b_i \in \mathcal{B}$, we locate the associated edge in $FVD(\mathcal{R})$, if it exists, in $O(n)$ time, resulting in $O(mn)$ time over all event points. For each event point b_i and its associated edge e_{jk} of $FVD(\mathcal{R})$ we perform a circular range counting query with the circle that passes through b_i, r_j and r_k. This gives us the number of blue points within that circle. Using the data structure in [6], over all event points, this results in $O^*(m + m^{1/2}k^{1/2})$ query time, where k is the total number of points returned by all of the $O(m)$ range counting queries, and $k < m^2$. In the worst case, this amounts to $O^*(m^{1.5})$ total time for all of the m queries, where the $O^*()$ notation ignores some polylogarithmic factor. The preprocessing requires $O^*(m^{1.5})$ time and space.

Theorem 2. *The minimum separating circle can be found in $O(mn) + O^*(m^{1.5})$ time using $O(n) + O^*(m^{1.5})$ space.*

We note that both of our algorithms can be easily modified to return all smallest circles by appending circles that are the same size as $C_{\mathcal{B}}(\mathcal{R})$ and contain the same number of blue points to a list.

VI. Largest Separating Circle

For the largest separating circle problem we make use of our result from Section IV. While we are sweeping along the edges of $FVD(\mathcal{R})$ searching for a smallest separating circle, we can store all circles which contain the minimum number of blue points in a list C. For each of these circles C_i, we can use the algorithm from [11] for finding the largest separating circle of two purely seperable sets by modifying our input. We give as input the set $\mathcal{R}$ as well as the set $\mathcal{B}'$ where $\mathcal{B}'$ is $\mathcal{B}$ less the blue points in C_i. If there are k circles in C, then we must perform this step k times, $k = m$ in the worst case. Since the algorithm given in [11] runs in $O\left((n+m)\log(n+m)\right)$ time, our running time is $O(nm\log m + k\,(n+m)\log(n+m))$, where the first term comes from the smallest enclosing circle algorithm given in Section IV.

Theorem 3. *The largest separating circle can be found in $O(m\,(n+m)\log(n+m))$ time using $O(n+m)$ space.*

VII. Conclusion

We have presented two algorithms for computing the smallest separating circle and an algorithm for finding the largest separating circle of two sets of points.

We have defined the separating circle above to be the circle containing all red points and as few blue points as possible. If it is desired to have the overall smallest separating circle, the algorithm can be run a second time with the red and blue point sets swapped. Then, the comparison of the second run with the first shall give the desired answer.

Although we proved tight bounds for the number of smallest separating circles, the equivalent bounds on the number of largest separating circles remains as an open problem.

References

[1] M. Abellanas, F. Hurtado, C. Icking, R. Klein, E. Langetepe, L. Ma, and B. Palop. Smallest color-spanning objects. In *Proceedings of the 9th Annual European Symposium on Algorithms*, pages 278–289. Springer-Verlag, 2001.

[2] P. Agarwal, B. Aronov, and V. Koltun. Efficient algorithms for bichromatic separability. *ACM Transactions on Algorithms*, 2(2):209–227, 2006.

[3] B. Aronov, D. Rappaport. C. Seara D. Garijo, Y. Núñez, and J. Urrutia. Measuring the error of linear separators on linearly inseparable data. In *XIII Encuentros de Geometria Computacional, Zaragoza, España*, 2009.

[4] J. Boissonnat, J. Czyzowicz, O. Devillers, J. Urrutia, M. Yvinec, and F. Preparata. Computing largest circles separating two sets of segments, 2000.

[5] P. Bose, S. Langerman, and S. Roy. Smallest enclosing circle centered on a query line segment. In *Canadian Conference on Computational Geometry*, 2008.

[6] T. Chan. On enumerating and selecting distances. In *Proceedings of the 14th annual symposium on Computational geometry*, pages 279–286, New York, NY, USA, 1998. ACM.

[7] S. Fisk. Separating point sets by circles, and the recognition of digital disks. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 554–556, July 1986.

[8] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

[9] J Matoušek. On enclosing k points by a circle. *Information Processing Letters*, 53(4):217–221, 1995.

[10] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[11] J. O'Rourke, S. Kosaraju, and N. Megiddo. Computing circular separability. *Discrete Computational Geometry*, 1:105–113, 1986.

[12] S. Roy, A. Karmakar, S. Das, and S. Nandy. Constrained minimum enclosing circle with center on a query line segment. *Computational Geometry: Theory and Applications*, 42(6-7):632–638, 2009.

[13] G. Toussaint. computing largest empty circles with location constraints. *International Journal of Parallel Programming*, 12(5):347–358, 1983.