Optimally Placing a Star on a Point Set

Prosenjit Bose* Jason Morrison*

Abstract

We consider the problem of placing a *star* $\vec{R}$ on a set $\mathcal{S}$ of n points in the plane in order to maximize a given objective function. A *star* $\vec{R}$ is a set of m rays $\{r_1, \ldots, r_m\}$ in $\mathbb{R}^2$, emanating from a point p such that the angle between two consecutive rays is $2\pi/m$. The cone defined by two consecutive rays with apex p is k-occupied if it contains at least k points of $\mathcal{S}$. We design algorithms to compute placements of $\vec{R}$ that maximize or minimize the number of k-occupied cones of $\vec{R}$. Our main result is an $O(nm^3 \log(m) + nm \log(nm)))$ time algorithm for finding a placement of $\vec{R}$ that maximizes the number of k-occupied cones.

1 Introduction

This paper studies a series of star placement problems whose goal is to partition a point set $\mathcal{S}$ while optimizing a given objective function. To simplify the analysis, all stars $\vec{R}$ have m rays that are equally spaced around a central point. We assume that the order of the rays around the central point is fixed and the only variable is the position of the central point. We note that most of the results presented can be extended to stars whose rays are not equally spaced, with appropriate (and only minor) modifications to running time and space.

The small sized description of an oriented star allows us to partition a point set $\mathcal{S}$ into possibly many unique sub-sets while allowing quick query time to decide what sub-set a given query point may belong. This partitioning is a generalization of the partitioning to construct static quad-trees and k-d trees[1]. Due to the generalization, the recursive use of the partitioning techniques described constructs data structures similar to quad-trees with degree m.

Returning to the mathematical definitions, a cone of $\vec{R}$ is said to be k-occupied if it contains at least k points. A placement of a star $\vec{R}$ has *occupancy* h if at least h of its m cones are 1-occupied. The placement of a star $\vec{R}$ has a *k-occupancy* of c if c of its cones are k-occupied. We present algorithms to find star placements that solve the following problems:

Problem 1 *Find a placement of $\vec{R}$ that maximizes the occupancy of $\vec{R}$.*

Problem 2 *Given a fixed integer k, find a placement of $\vec{R}$ that minimizes/maximizes the k-occupancy of $\vec{R}$.*

Problem 3 *Given a fixed integer k find a placement of $\vec{R}$ (if one exists) such that no cone of $\vec{R}$ is k-occupied.*

Problem 4 *Find the smallest integer k such that there exists a placement of $\vec{R}$ with k-occupancy 0.*

Problem 5 *Given a fixed integer c, find the largest integer k such that there exists a placement of $\vec{R}$ with k-occupancy at least c.*

The solutions presented in this paper are based on computing the region of possible placements of $\vec{R}$ that ensure that a specific cone of $\vec{R}$ is at most k-occupied. By computing these regions for the different cones and different values of k it is possible to determine the solutions for each of the above problems.

For the remainder of this paper we assume that the points contained in $\mathcal{S}$ are in general position with respect to the star. Specifically all lines l, parallel to any of the m rays, contain at most 1 point of $\mathcal{S}$.

2 k^{th} Maximal Regions

Given a cone $\vec{C}$ with a specific orientation, the *k^{th} maximal region* $M_k(\mathcal{S}, \vec{C})$ is the region of the plane where the placement of $\vec{C}$ contains at most k points of $\mathcal{S}$. We refer to the boundary of the k^{th} maximal region as the k^{th} maximal layer. This section provides an $O(n \log n)$ time algorithm for computing the k^{th} maximal layer of a given cone.

The k^{th} maximal layer and k^{th} maximal region depend on the orientation of the input cone $\vec{C}$. We assume without loss of generality that $\vec{C}$ has one ray (r_{i1}) oriented along the positive y axis and a second ray (r_{i2}) $\alpha < \pi$ radians c.w. from the first.

2.1 Properties of maximal layers and regions

Several properties are useful in constructing the k^{th} maximal region. First, the boundary (i.e., the k^{th} maximal layer hereafter denoted $\partial M_k(\mathcal{S}, \vec{C})$) is not part of the region. $\partial M_k(\mathcal{S}, \vec{C})$ partitions the plane into two open regions. One region has the property that the

*Research supported in part by NSERC. School of Computer Science, Carleton University, Ottawa, Canada, jit, morrison@scs.carleton.ca

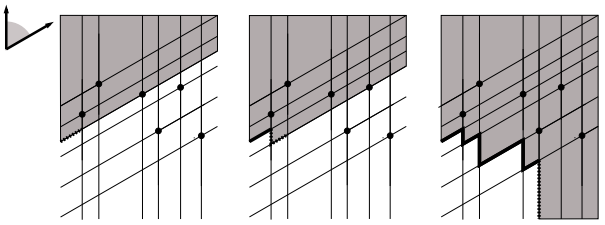

Figure 1: a) initializing $\partial M_k(\mathcal{S}, \vec{C})$ b) 1^{st} update of $\partial M_k(\mathcal{S}, \vec{C})$ c) post-processing of $\partial M_k(\mathcal{S}, \vec{C})$ Maximal region boundary is shown for $k = 2$ and oriented cone as displayed, updates to $\partial M_k(\mathcal{S}, \vec{C})$ are shown with thick dotted segments.

placement of the apex of $\vec{C}$ in the region results in $\vec{C}$ containing at most k points. The placement of the apex in the other region results in $\vec{C}$ containing more than k points. Thus any placement of $\vec{C}$ on $\partial M_k(\mathcal{S}, \vec{C})$ must have at least one point on the boundary of $\vec{C}$, and contain at most k points in its interior.

Second, note that $\partial M_k(\mathcal{S}, \vec{C})$ consists of straight segments parallel to the bounding rays of $\vec{C}$ (details are given in the full version of the paper [4]).

2.2 Construction

We now show how to compute the boundary of the k^{th} maximal region. First, the points of $\mathcal{S}$ are sorted in the direction perpendicular to r_{i2}. We call this the r_{i2} ordering. The general position assumption guarantees that this is a total order on the points of $\mathcal{S}$. The points are processed in a plane sweep using the r_{i2} ordering from maximum to minimum. After each event point p_i, the boundary of the k^{th} maximal region, $\partial M_k(n)$, might be updated farther along the r_{i1} or r_{i2} ordering. A priority queue holds $k+1$ of the points contained in a cone $\vec{C}$ placed at the rightmost event point of $\partial M_k(n)$. This ensures the algorithm's invariant that there are more than k points within any cone placed on the boundary constructed and at most k of those points are strictly interior.

The algorithm begins by initializing the priority queue with the first $k + 1$ event points. The boundary $\partial M_k(\mathcal{S}, \vec{C})$ is updated by adding the ray of the line passing through the $k + 1^{st}$ event point and parallel to r_{i2} from $-\infty$ in the x-direction to the lowest r_{i1} ordering in the queue (See Figure 1a). The point q is the end point of the newly added ray.

When a new event point p_{bottom} is swept over it is inserted into the priority queue and one of the following cases is occurs:

1. If the r_{i1} ordering of the inserted point is strictly less than all of the other points, then it is removed from the queue and $\partial M_k(\mathcal{S}, \vec{C})$ remains the same.

2. Otherwise the point p_{left} at the top of the queue having the minimum r_{i1} ordering should be deleted from the queue and value p_{left} should be updated. The boundary $\partial M_k(\mathcal{S}, \vec{C})$ is subsequently updated as follows:

   (a) Add the vertical segment from q to the line parallel to r_{i2} passing through p_{bottom}. Set q to this new endpoint.

   (b) Add a segment parallel to r_{i2} from q to the line parallel to r_{i1} passing through the new p_{left}. Set q to this new endpoint.

When the sweep is completed $\partial M_k(\mathcal{S}, \vec{C})$ has to be updated with the vertical ray from q to $-\infty$ in the y direction.

Theorem 1 *For a given point set $\mathcal{S}$ an arbitrary integer $k < |\mathcal{S}| = n$, and an oriented cone $\vec{C}$ the k^{th} maximal region of $\mathcal{S}$ with respect to $\vec{C}$ is computable in $O(n \log n)$ time and $O(n)$ space.*

Proof. First we show that the algorithm maintains the invariant that there are more than k points within a cone placed anywhere along the constructed boundary and at most k of those points are strictly interior. We prove this invariant by induction on the number of event points generating segments in the boundary.

The base case is the insertion of the initial ray. By definition this ray any cone $\vec{C}$ placed on the ray must contain $k + 1$ points with one on r_{i2}.

Assume that the invariant is maintained after the i^{th} event point generating a segment in the boundary. Now consider the $i + 1^{st}$ such event point. If the event point has an r_{i1} ordering less than q then it will be to the left of the remainder of boundary, by construction, and therefore cannot be contained in any cones placed on that portion of the boundary. Furthermore due to its r_{i2} ordering it is below the boundary previously constructed and therefore cannot be contained in any cones placed on that portion of the boundary. This proves that discarding the point in Case 1) does not affect the invariant.

For Case 2, a cone $\vec{C}$ placed at q or along the added vertical segment will contain p_{left} on its boundary. Of the remaining $k + 1$ points in the queue at most k can be strictly interior to $\vec{C}$ because the point p_{bottom} can only be on the boundary of $\vec{C}$ at the new q after step 2a). For any cone placed along the new segment parallel to r_{i2} the point p_{bottom} must be on the boundary by construction and the old p_{left} can only be on the boundary of the cone. This leaves at most k points in the queue that can be strictly interior to any $\vec{C}$ placed on the boundary.

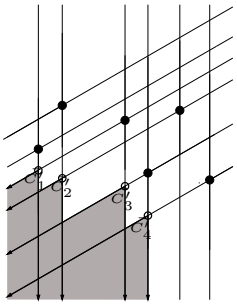

Figure 2: A set of cones representing $\overline{M_k(\mathcal{S}, \vec{C})}$

Since the cone always has at least a single point on the boundary and the remaining k points can be in the interior the invariant must be true. This proves the inductive case and thereby proves that the invariant is kept by the algorithm.

It remains to be shown that by maintaining the invariant the algorithm constructs the correct boundary. This proof is direct as the constructed boundary has properties identical to the desired boundary as proven by the invariant. Since we have proven that there is only one, connected boundary then the constructed boundary is correct.

Analyzing the algorithm complexity we note that the sorting phase requires $O(n \log n)$ and, the $O(n)$ insertions and deletions in the priority queue use $O(n \log k)$ time. Finally, the size of $\partial M_k(\mathcal{S}, \vec{C})$ is $O(n)$ because at most two segments are added for each deletion and at most $O(n)$ deletions are performed.

□

As noted above, the k^{th} maximal region divides the plane into two regions: the region ($M_k(\mathcal{S}, \vec{C})$) and its complement ($\overline{M_k(\mathcal{S}, \vec{C})} = \mathbb{R}^2 - M_k(\mathcal{S}, \vec{C})$). An important property to note is that $\overline{M_k(\mathcal{S}, \vec{C})}$ can be represented as the union of a set of $O(n)$ cones each of whose angle is the same as the angle of $\vec{C}$. The proof of correctness of this representation follows directly from discussion above. Formally, $\overline{M_k(\mathcal{S}, \vec{C})} = \{\vec{C'_1} \cup \ldots \cup \vec{C'}_{O(n)}\}$ where $\vec{C'}$ is the cone $\vec{C}$ rotated by π radians (see Figure 2). Since $\partial M_k(\mathcal{S}, \vec{C})$ can be calculated in $O(n \log n)$ time and $O(n)$ space, the cone representation of $\overline{M_k(\mathcal{S}, \vec{C})}$ can be computed in the same time.

3 Solutions

The first observation about Problems 1-5 is that a simple brute force approach exists for each of the problems. Consider the arrangement resulting by placing a star $\vec{R}$ on each point in $\mathcal{S}$. It is easy to see that one can restrict their search to star placements on vertices of this arrangement. However, the resulting arrangement has a complexity of $O(n^2 m^2)$. In the following section we achieve a better result by using geometric properties of k maximal regions. We begin by presenting a general technique for solving placement problems of stars (Section 3.1). In Section 3.2 we apply this technique to Problems 1-5 thus providing algorithmic solutions for each problem.

3.1 A Maximal Region Approach

The star placement problems of this paper are occupancy based problems. Our approach to solve these problems is to construct the complement of the k^{th} maximal region for each of the m cones in $\vec{R}$. Once the regions are calculated they are overlayed and the resulting arrangement is examined. From this arrangement it is possible to tell exactly which cones contain strictly more than k points. Depending on the problem either one or more values of k may have to be examined. The remainder of this section explains the details of the approach outlined above for an arbitrary value k.

First, each of the m regions $\overline{M_k(\mathcal{S}, \vec{C}_i)}$ are calculated using the algorithm described in Section 2.2. This requires $O(mn \log n)$ time and $O(nm)$ space. The boundary of each region $\overline{M_k(\mathcal{S}, \vec{C}_i)}$ is represented by the upper envelope of the $O(n)$ cones with fixed angle $\frac{2\pi}{m}$ and identical orientation. A result by Efrat et al. [3] shows that the complexity of the arrangement of such boundaries is $O(nm^3 \log m)$ and is computable using Chazelle and Edelsbrunner's algorithm [2] in $O(nm^3 \log m + nm \log(nm))$ time.

Once the arrangement has been computed, each edge can be visited in depth first search order. Changing from one edge to the next is equivalent to maintaining the dynamic partitioning of the point set. This can be done in $O(1)$ time per change and with at most $O(nm^3 \log m)$ such changes.

Theorem 2 *Given a planar point set $\mathcal{S}$, a star $\vec{R}$ comprised of m cones $\{\vec{C_1}, \ldots, \vec{C_m}\}$ and an integer k, the arrangement of all regions $\overline{M_k(\mathcal{S}, \vec{C}_i)}$ can be examined in $O(nm \log nm + nm^3 \log m)$ time and $O(nm^3 \log m)$ space.*

Corollary 3 *Given a planar point set $\mathcal{S}$, an oriented star $\vec{R}$ comprised of a constant number of oriented cones $\{\vec{C_1}, \ldots, \vec{C_{O(1)}}\}$ and an integer k, the arrangement of all regions $\overline{M_k(\mathcal{S}, \vec{C}_i)}$ can be examined in $O(n \log n)$ time and $O(n)$ space.*

3.2 Solutions and Analyses

This section details the application of the maximal region approach to star placement problems. Five occupancy based problems are studied and full analyses of

the solutions are given. Each problem assumes that we are given a planar point set $\mathcal{S}$ and a star $\vec{R}$.

Problem 1 *Find a placement of $\vec{R}$ that maximizes the occupancy of $\vec{R}$.*

The goal is to find a placement of $\vec{R}$ for which the most number of cones contain at least one point. This is the same as finding a point at which the most number of $M_0(\mathcal{S}, \vec{C}_i)$'s overlap. Thus a solution can be computed by examining the arrangement of $M_0(\mathcal{S}, \vec{C}_i)$'s as described in the previous section and maintaining the maximum overlap. The total complexity of this algorithm is $O(nm^3 \log m + nm \log nm)$ time and $O(nm^3 \log m)$ space. The minimization version of this problem is omitted as it is always possible to achieve an occupancy of 1 in any desired cone.

Problem 2 *Given a fixed integer k, find a placement of $\vec{R}$ that minimizes/maximizes the k-occupancy of $\vec{R}$.*

The second problem is a generalization of the first. The difference between the solutions for these problems is that problem 2 examines the arrangement of $M_{k-1}(\mathcal{S}, \vec{C}_i)$'s. As before the solution is computed by examining the arrangement and maintaining where the (least/most) number of overlapping regions occurs. This is accomplished in $O(nm^3 \log m + nm \log nm)$ time and $O(nm^3 \log m)$ space.

Problem 3 *Given a fixed integer k find a placement of $\vec{R}$ (if one exists) such that no cone of $\vec{R}$ is k-occupied.*

This problem is presented because our solution for the next problem uses this as a sub-problem. This problem is solved by finding the placement with minimum k-occupancy and determining if that k-occupancy is zero. Since that adds only $O(1)$ time to Problem 2 this problem is also solvable in $O(nm^3 \log m + nm \log nm)$ time and $O(nm^3 \log m)$ space.

Problem 4 *Find the smallest integer k such that there exists a placement of $\vec{R}$ with k-occupancy 0.*

An alternative expression of this problem is to find the placement of $\vec{R}$ that minimizes the maximum number of points in any cone. The goal is to search through possible values of k and find the minimum value for which there exists a placement with k-occupancy 0.

We claim that the optimal value of k must lie in the interval $[\lceil \frac{n}{m} \rceil, \lceil \frac{n}{2} \rceil]$. This is because each point must lie in at least one cone and at worst all of the points can be split into at least two cones by placing the star on the median of an arbitrary ordering r_i.

Furthermore, determining k is possible using binary search on the interval $[\lceil \frac{n}{m} \rceil, \lceil \frac{n}{2} \rceil]$. This is validated by observing that the k-occupancy of any placement that minimizes the maximum number of points in any cone the k-occupancy of such a point decreases monotonically as k increases. This leads to a $O((nm^3 \log m + nm \log nm) \log n)$ time and $O(nm^3 \log m)$ space algorithm. With a more careful binary search this is reducible from an $O(\log n)$ factor to $O(\log \delta)$ where the optimum value of $k = \lceil \frac{n}{m} \rceil + \delta$.

Problem 5 *Given a fixed integer c, find the largest integer k such that there exists a placement of $\vec{R}$ with k-occupancy at least c.*

This is a clustering problem (that creates c equally large clusters each of size at least k) using the star placement as the clustering tool. The solution to this problem is similar to the solution for the previous problem. Since n points must be partitioned between at least c cones we have that $k \leq \lfloor \frac{n}{c} \rfloor$. As before we are able to use binary search with each iteration selecting a value k and determining if the maximum k-occupancy is at least c. Thus the problem is solvable in $O((nm^3 \log m + nm \log nm) \log n)$ time and $O(nm^3 \log m)$ space.

We note that for values of $n > m \log m$ our algorithms are better than the brute force technique. But for large values of m the brute force technique is preferable.

4 Conclusion

We have provided a new technique to partition planar point sets. In doing so we studied a generalization of maximal layers and provided an algorithm for their construction. The results presented here do generalize to non-equally spaced stars with modifications to the running time and space. The algorithm for maximal layers also generalizes to handle degeneracies when the points are not in general position.

References

[1] M. de Berg, M. van Kreveld, M. Overmars and O. Schwartzkopf. *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 1997.

[2] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, January 1992.

[3] A. Efrat, G. Rote, and M. Sharir. On the union of fat wedges and separating a collection of segments by a line. *Computational Geometry: Theory & Applications*, 3(5):277–288, November 1994.

[4] J. Morrison. *Geometric Placement Problems*. PhD thesis, School of Computer Science, Carleton University, Ottawa, Canada, May 2002.