Guarding a Treasury

Svante Carlsson[*] Håkan Jonsson[*]

Abstract

We present a preliminary investigation of problems concerning the guarding of treasures within a simple polygon where values are assigned to the treasures. In particular, we consider the case of finding the location of a guard so that the value of the visible treasures is maximized. We obtain efficient algorithms for variations of the problem where we restrict the placement of the guard or the location of the treasures. We are also able to prove tight lower bounds for some of these variations.

1 Introduction

Original art gallery problems ask questions on the lower bounds of the number of guards needed such that every points of the polygon is visible to at least one guard, and where these guards should be placed to achieve this. In 1975 Chvátal [3] proved that $\lfloor \frac{n}{3} \rfloor$ guards are sufficient and sometimes necessary to guard the interior of a polygon. Related problems with a flavour of art galleries, such as guarding the exterior of polygons, guarding line segments in the plane, using mobile watchmen and many more, are nicely presented in O'Rourke's monograph [9] and in Shermer's survey [11]. In this paper we deal with a variation of the problem with *treasures in an art gallery* named by Deneen and Joshi [4]. They define a *treasury* to be a simple polygon containing t treasures and present an algorithm for computing an approximation of the smallest set of guards needed such that all treasures are in sight by at least one guard. We extend their definition, by assigning a value (a weight) to each treasure and consider the placement of a single guard such that the sum of the values of the treasures visible to the guard is maximized. In addition to placing a guard in a general position inside the treasury, we also study the cases where the guard is restricted to corners or to the boundary of the treasury. The treasures are assumed to be in general positions, but we also treat the special case where they are placed at the corners of the treasury. We present solutions to all problems considered and a lower bound for guarding a treasury from the boundary.

[*]Division of Computer Science, Luleå University of Technology, S-971 87 LULEÅ, Sweden.

2 Preliminaries

A *treasury* is a simple polygon P bounded by n straight line *edges* $e_1, e_2, \ldots, e_n$, where e_i and e_{i+1} are joined by the *vertex* p_i of P. To each of t *treasures* positioned at *sites* $s_1, s_2, \ldots, s_t$ in P there is a value $w_1, w_2, \ldots, w_t$ associated. Our guard is supposed to be posted at a point in P where the sum of the values of the treasures visible to the guard is maximized. The *boundary* of P is the union of all edges, and the *interior* is the area surrounded by the boundary. We assume that P does not contain holes. Let r be the number of *reflex* vertices, i.e. vertices that join edges of P which forms angles greater than 180 degrees inside P. We use the algebraic tree model as our model of computation [10], and will mainly use the frame-work of visibility presented by Bose [1].

We say that two points p and q in P are *visible* to each other if the straight line segment $\overline{pq}$ is a subset of P. A *visibility polygon* of a point $x \in P$ contains all points of P visible to x. This polygon can be computed in $\Theta(n)$ time using the algorithm of El-Gindy and Avis [5], a fact we will use extensively in this paper. Visibility of treasures in a treasury can

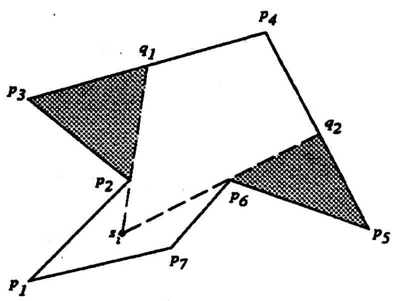

Figure 1: A site s_i and its associated pockets (the shaded areas). The dashed line segment $\overline{p_2 q_1}$ and $\overline{p_6 q_2}$ are windows generated by s_i.

be dealt with using *windows*. A window of a point x (the *generator*) belonging to P is an edge of the visibility polygon of x not contained in any of the edges of P, delimited by the two *event points* b (a reflex vertex called the *base*) and the point q on the boundary (the *end*). Windows constitutes borders a guard have to pass when being moved in order to loose (or gain) sight of x, since inside P only reflex vertices might block the view (Figure 1). A maximal connected subset of P from which the generator is not visible is called a *pocket*. A pocket is joined to the visibility polygon by a window. Some care has to be taken to avoid ambiguities due to collinearities. We refer to Bose [1] for further details.

3 Guarding Treasures of a Treasury

In this section we present algorithms for solving the problem of placing a single stationary guard in a treasury such that the value of the treasures visible to the guard is maximized. We start by investigating the cases where the guard is restricted to stand in the corners or on the boundary of the treasury. We end this section by not having any restrictions as to where in the treasury we are allowed to place the guard. In each case we begin by assuming that the treasures can be placed in arbitrary positions. We then consider the special case where the treasures are placed at the vertices of the treasury. In the latter case we will assume that the number of treasures is at most n, since two treasures with values w_i and w_j in the same corner can be treated as one treasure with value $w_i + w_j$. When the treasures are placed at the corners of the treasury we will use the term vertex to denote both the treasure located at the vertex as well as the vertex itself.

The main observation used in this paper is that the number of visible sites changes by exactly one as a window is passed. Using the windows we partition P into regions, ***visibility regions***, where all the points in the region view the same treasures. Thus, it suffice to consider only one point of each visibility region.

3.1 A Guard Restricted to Corners

Consider the problem of placing a guard at one of the n corners of the treasury such that the value of the treasures visible to the guard is maximized. One way of solving the problem is to compute the visibility polygon, in linear-time, from each treasure site s_i and adding the value of the treasure to the sum of values of the vertices of the visibility polygon. Then, choose in linear time the vertex with the largest accumulated sum of treasure values to be the place to position the guard in $\Theta(n)$ time. Computing t visibility polygons one at a time takes $\Theta(tn)$ time and $\Theta(n)$ space.

If the number of treasures is much larger than n we use another approach. We compute the visibility regions formed by the windows generated by all vertices of P in $O(nr \log n + n^2 r)$ time using the algorithm of Chazelle and Edelsbrunner [2] and preprocess this subdivision of P for planar point location as done by Kirkpatric in $O(n^2 r \log n)$ time. Now, we can locate the region containing a treasure using $\Theta(\log n)$ time. As we locate the treasures we replace all treasures of a region with a representative treasure having the accumulated value of all the other. Traverse the collection of visibility regions while keeping track of which vertices are visible. When we enter a region containing a treasure we add the value of this treasure to the sum of values of each vertex currently visible. The trivial upper bound on the number of summations is $O(n^3 r)$, but we believe that it can be reduced further. To preprocess the decomposition of P, to locate t treasures and to traverse the regions takes $O(n^2r\log n + t\log n + n^3r)$ time, i.e. this algorithm is faster then the algorithm above if $t \in \Omega(n^2r)$.

Theorem 3.1 *In a treasury containing t treasures in arbitrary positions, a vertex v with the maximal value of the treasures visible from v could be found in time $O(\min(tn, t\log n + n^3r))$ time.*

Now, assume that the treasures are placed at the corners of the treasures. In this case, we consider the *visibility graph.* Nodes of the visibility graph represents vertices of P and arcs connect vertices that are visible to each other. Hersberger [7] has devised an algorithm for computing the visibility graph of P in output-sensitive time $O(m)$ where m is the number of mutually visible vertices. Given this graph we can compute the sum of values of the treasures visible to a vertex in time proportional to the degree of the node corresponding to the vertex, or in $O(m)$ time for all vertices. If the number of treasures is much less than n we use the algorithm above that runs in $\Theta(tn)$ time for small values of t.

Theorem 3.2 *If the treasures are placed at the corners, a vertex from where the value of the visible treasures is maximized can be found in $\Theta(min(m, tn))$ time.*

We believe that this is close to the lower bound, since this is close to the time it takes to count the number of visible vertices of every vertex.

3.2 A Guard Along the Walls of a Treasury

Consider the problem of placing a guard somewhere along the wall of a treasury such that the value of the treasures visible is maximized. Compared to being restricted to corners, this problem might at a first glance seem much harder since the boundary of P contains an infinite number of points. However, a guard being moved along the boundary views the same treasures between two event points adjacent to each other. At an event point the number of visible treasures changes by exactly one. Due to this, we only need to consider one point of each part of the boundary delimited by two adjacent event points. Trivially, we have:

Lemma 3.1 *t treasures generates $O(tr)$ event points (windows) on the boundary of a treasury P.*

To compute all the event points, we use the sites as generators one at a time.

Lemma 3.2 *It is possible to compute all event points on the boundary generated by a site s_i in time $O(n)$.*

Proof: The visibility polygon contains enough information for computing the event points as they appear in angular order. □

Corollary 3.1 *In a treasury containing t treasures at arbitrary positions, all event points can be computed in $O(tn)$ time.*

Theorem 3.3 *In a treasury containing t treasures in arbitrary positions, a point on the boundary where the total value of the treasures visible is maximized can be found in $O(tn + W\log t)$ time using $\Theta(W)$ space, where W is the number of windows generated by the treasures.*

Proof: We find the place from which the value of the visible treasures is maximized by computing the event points, sorting them along the boundary and then visiting them in sorted order. By Corollary 3.1, we can compute the event points generated from all treasure sites and keep them in t separate lists containing $O(r)$ event points each, in $O(tn)$ time. In each list, the event points will be sorted in angular order along the boundary. Note that at reflex vertices there might be many event points. Merging t sorted lists containing a total of W event points take time $O(W\log t)$. We start the walk along the boundary at a reflex vertex, where the treasures visible can be computed at the same time as all the event points are computed. As we pass an event point we can determine in constant time whether the number of visible treasures increases or decreases. We can do this by examining how the window of the event point is positioned relative the boundary. Thus, by Lemma 3.1 we can determine which treasures are visible at all event points in $\Theta(W)$ time. □

Restrict the treasures to the corners of the treasury and consider the problem of placing a guard somewhere along the wall such that the value of the treasures seen by the guard is maximized. We will show how the result in Theorem 3.3 could be improved in the case where the treasures are all placed at the corners, and also provide a non-trivial lower bound on how fast an algorithm that finds the place on the boundary where the value of the treasures visible is maximized can be. First, we consider a given edge e_i of P and show that the best position on e_i in order to guard as large a value as possible could be computed in $\Theta(n \log n)$ time. Then we return to the problem of placing a guard somewhere on the boundary.

Suppose that we are given an edge e_i of P and asked where on e_i the value of the visible treasures are maximized. To find this we compute the event points on e_i and then visit them in sorted order along e_i. From Lemma 3.7 in Bose [1] we have:

Lemma 3.3 *There are $O(n)$ end points on an edge of P.*

To compute the event points in Lemma 3.3 we use the algorithm of Lee and Lin [8] that computes the *weak-visibility polygon* from an edge of P, i.e. the part of an polygon visible from at least one point along the edge, in time $O(n \log n)$ making some minor changes. We can sort the $O(n)$ end points and visit them in sorted order in $\Theta(n \log n)$ as we did earlier in this section. This gives us:

Theorem 3.4 *Given a treasury with treasures incident to the vertices, the place on an given edge where the value of the visible sites is maximized can be found in $\Theta(n \log n)$ time.*

We show that the algorithm presented above is optimal by showing a lower bound of $\Omega(t \log t)$ on the time complexity on the problem. The lower bound is obtained by reduction to the problem of determining whether any two members of a set of n numbers differ from each other by less than ε, which is known to require $\Omega(n \log n)$ in the algebraic decision tree model [10, Corollary 8.1, pp. 326].

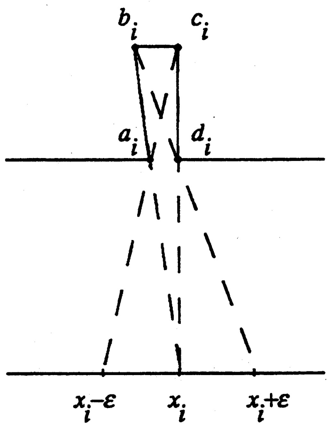

Figure 2: A part of P_Ω showing the chimney of the number x_i.

Theorem 3.5 *In a treasury containing $O(n)$ treasures at the vertices, finding the point on an edge e_i where the value of the visible sites is maximized requires $\Omega(t \log t)$ time.*

Proof: We start the proof by constructing a polygon P_Ω which we will use in the reduction. Consider a set, $\mathcal{X}$, of $\frac{t}{2}$ numbers $\{x_1, x_2, \ldots, x_{\frac{t}{2}}\}$, and let min and max be the smallest and the largest of all x_i respectively. Construct P_Ω in the following way:

1. Build a rectangular polygon with vertices $p_1 = (min-1, 0), p_2 = (min-1, 1), p_3 = (max+1, 1)$ and $p_4 = (max+1, 0)$. Call the edge $\overline{p_1p_4}$ the *floor* of the polygon and the edge $\overline{p_2p_3}$ the *roof.* Let $\Delta x = max - min$.

2. For each x_i cut the roof between $a_i = \frac{\Delta x}{n+1} i - \frac{\delta}{2}$ and $d_i = \frac{\Delta x}{n+1} i + \frac{\delta}{2}$ for some sufficiently small gap $\delta = |a_i - d_i| < \varepsilon$, and add 3 new edges $\overline{a_ib_i}, \overline{b_ic_i}$ and $\overline{c_id_i}$ to P_Ω at this cut by building a *chimney* (a pocket of p_2 as shown in Figure 2). Place b_i at the intersection point between the line through $(x_i + \varepsilon, 0)$ and d_i and the line through $(x_i, 0)$ and a_i. Position c_i at the intersection point between the line through $(x_i - \varepsilon, 0)$ and a_i and the line through $(x_i, 0)$

and d_i. Choose the gap δ sufficiently small so that no chimneys intersect. Place a treasure with value 1 at b_i and c_i.

At the end of this construction step, P_Ω consists of $2t+4$ vertices. If and only if there are two numbers x_i and x_j such that $|x_i - x_j| < \varepsilon$ the number of treasures visible from $(x_i, 0)$ will be larger than two. Hence, if we are able to find the place of maximal visibility on the base we are able to solve the problem of determining whether any two members of a set of n numbers differ from each other by less than ε. □

We are now able to conclude that our algorithm, based on Lee and Lin's method is optimal if the number of treasures is $\Theta(n)$. Using this algorithm on each edge and then in linear time selecting the best solution yields an $O(n^2 \log n)$ time algorithm, which is no better than the result of Section 3.2. We improve this to a complexity tied to the number of mutually visible vertices by observing that only vertices visible to a generator may generate windows. This means that the visibility graph and P together contains all information we need in order to find all the windows and event points.

Lemma 3.4 *Given a treasury containing t treasures at the vertices, all event points on the boundary can be computed in $O(m)$ time.*

Proof: Computing the visibility graph using the algorithm of Hersberger [7] requires $O(m)$ time, and by slightly modifying it we can get all visibility polygons of all vertices in the same amount of time. Given the visibility polygon of v, computing all event points in P generated by v requires no more than time proportional to the number of vertices in the visibility polygon. □

If the number of treasures is much less than n we use the general algorithm of Section 3.2 which in this case runs in $O(tn + tr \log t)$ time. We get the following theorem as a consequence of Theorem 3.3:

Theorem 3.6 *In a treasury containing treasures at the vertices, it is possible to compute the point on the boundary where the value of the treasures visible is maximized in $O(min(m + W \log t, tn + W \log t))$ time.*

The lower bound of $\Omega(t \log t)$ on an edge in treasures containing $O(n)$ treasures placed at the vertices (Theorem 3.5) also applies to the boundary. However, $\Omega(m)$ is not an obvious lower bound on the problem of placing a guard anywhere on the boundary. It is not sure that we have to consider all event points.

3.3 A Guard in a General Position in a Treasury

We end this section by considering the case where the guard is allowed to be positioned anywhere inside the treasury. As before, the set of treasures visible to a guard being moved changes only when a window is passed. To place a stationary guard at an arbitrary position within the treasury is thus equivalent to finding a region $\mathcal{R}$ in P bounded by windows and edges of the polygon such that the value of the treasures visible in $\mathcal{R}$ is maximal. Given n line segments in the plane, it is possible to compute the arrangement of these line segments in time $O(n \log n + k)$, where k is the number of intersections [2]. Corollary 3.1 states that the $O(tr)$ windows can be computed in $O(tn)$ time. Given the windows, we can use the algorithm in [2] to compute the partition of P into visibility regions induced by the windows. $O(tr)$ line segments in general positions can form $O(t^2r^2)$ intersections. This is not the case if we consider windows in a treasury. In a treasury the $O(tr)$ windows will form $O(t^2r)$ intersections [6] which, since the set of intersections can be viewed as a planar graph, gives us:

Lemma 3.5 *A treasury can be partition into $O(t^2 r)$ visibility regions.*

Computing the visibility regions using [2] takes $O(tr \log tr + k)$ where k is the number of intersections. Within a region, the same set of treasures are visible from all points. As we compute the windows, we can compute the treasures that are visible from a reflex vertex without affecting the upper bound. By traversing the arrangement of regions in a depth-first fashion we can find the region that guards treasures of the largest value in $O(tn + tr \log tr + k)$ time.

Theorem 3.7 *Finding a point inside a treasury where the total value of the visible treasures is maximized can be done in $O(tn + W \log W + k)$ time.*

Consider the problem of placing a guard somewhere inside a treasury in order to guard treasures placed at the corners. We can use this restriction to obtain a marginal speed-up. By Lemma 3.1 we know that a treasury with one treasure at each vertex contains $O(nr)$ windows which can be computed in $O(m)$ time (Lemma 3.4). If the number of treasures is much less than n we use the main algorithm of Section 3.3 which runs in $O(tn+tr \log tr+k)$ time, where k is the number of visibility regions induced by the windows of the t treasures. Using these facts we can tune Theorem 3.7 into the following:

Theorem 3.8 *In a treasury where treasures are placed at the vertices, the vertex with maximum value of the treasures visible can be found in time $O(\min(m + nr \log n + k, tn + W \log W + k))$.*

4 Conclusions and Discussion

The intention of this paper was to investigate the principles of visibility within a simple polygon. In particular, we study the problem of guarding a finite set of sites, called treasures, whose placement may or may not be restricted by certain rules. We have presented algorithms for computing the placement of a stationary guard in a simple polygon containing treasures so that the value of the treasures visible to the guard is maximized. This is a new type of problem and we can see several extensions of it, for example using multiple guards or watchmen within the treasury. We would also like to obtain lower bounds for the problems studied here.

References

[1] P.K. Bose. Visibility in simple polygons. Master's thesis, University of Waterloo, 1991.

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

[3] V. Chvàtal. A combinatorial theorem in plane geometry. *J. Combinatorial Theory Ser. B*, 13(2):395–398, 1975.

[4] L. L. Deneen and S. Joshi. Treasures in an art gallery. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 17–22, 1992.

[5] H. ElGindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *J. Algorithms*, 2:186–197, 1981.

[6] J. L. Guibas, R. Motwani, and P. Raghavan. The robot location problem in two dimensions. In *Proceedings of the 3rd Symposium on Discrete Algorithms*, pages 259–268, 1992.

[7] J. Hersberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4:141–155, 1989.

[8] D.T. Lee and A.K. Lin. Computing the visibility polygon from an edge. *Computer Vision, Graphics and Image Processing*, 34:1–19, 1986.

[9] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford univ. press, 1987. ISBN 0-19-503965-3.

[10] F. P. Preparata and M. I. Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1985. ISBN 3-540-96131-3.

[11] T. Shermer. Recent results in art galleries. In *Proceedings of the IEEE*, pages 1384–1399, September 1992.