Placing Text Boxes on Graphs*

A Fast Approximation Algorithm for Maximizing Overlap of a Square and a Simple Polygon

Sjoerd van Hagen and Marc van Kreveld

Department of Information and Computing Sciences
Utrecht University, The Netherlands

Abstract. In this paper we consider the problem of placing a unit square on a face of a drawn graph bounded by n vertices such that the area of overlap is maximized. Exact algorithms are known that solve this problem in $O(n^2)$ time. We present an approximation algorithm that—for any given $\epsilon > 0$—places a $(1+\epsilon)$-square on the face such that the area of overlap is at least the area of overlap of a unit square in an optimal placement. The algorithm runs in $O(\frac{1}{\epsilon}\, n \log^2 n)$ time. Extensions of the algorithm solve the problem for unit discs, using $O(\frac{\log(1/\epsilon)}{\epsilon\sqrt{\epsilon}}\, n \log^2 n)$ time, and for bounded aspect ratio rectangles of unit area, using $O(\frac{1}{\epsilon^2}\, n \log^2 n)$ time.

1 Introduction

The annotation of drawn graphs comes in different forms. Vertices can be labeled with their name or index, edges may be labeled with extra information, or faces of the embedded graph may receive a label. The analogy with cartographic label placement is clear: Here we have point feature labels, linear feature labels, and areal feature labels. Areal features are for instance lakes, national parks, provinces, and countries.

A related cartographic question is that of annotating regions of a map with extra information instead of names. These can be text boxes, pie charts, histograms, or other diagrams that show statistics about that region. If the annotation does not fit inside the region, it must obviously overlap parts of other regions. To achieve the best possible association of the correct region and the annotation, it is desirable to have the largest possible overlap in area of the annotation and that region. A possible positive side effect is that not too much of the region boundaries is covered by the annotation, and if more regions are annotated, that their annotations usually do not overlap.

One can abstract an annotation by a rectangle, square, or circle of some given size, which represents the bounding shape of the annotation. A region on a map is typically a simple polygon (although sometimes it has holes). The algorithmic

* This research is partially funded by the Netherlands Organisation for Scientific Research (NWO) under BRICKS/FOCUS grant number 642.065.503.

I.G. Tollis and M. Patrignani (Eds.): GD 2008, LNCS 5417, pp. 284–295, 2009.

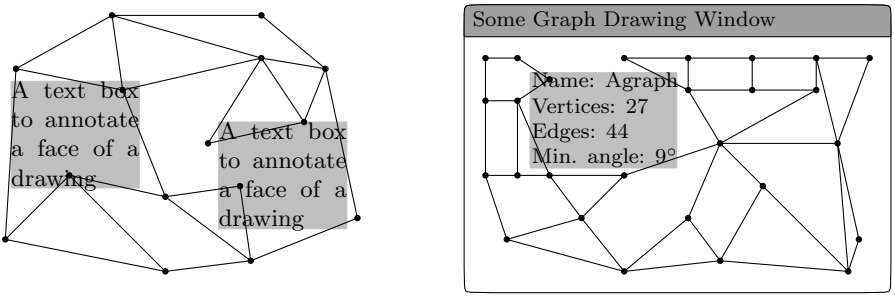

Fig. 1. Left, annotation of two faces by text boxes. Right, annotation in the outer face of a drawn graph.

question that arises is: How do we compute the placement of a simple shape with a simple polygon to maximize area of overlap efficiently? Van Kreveld *et al.* [16] studied this problem (along with some related problems) in the context of placing diagrams on maps. It was shown that the maximum overlap placement of a unit square on a simple polygon P with n vertices can be computed in $O(n^2)$ time. It was also shown that the placement problem with one degree of freedom—for example, the y-coordinate of the top of the unit square is fixed—can be solved in $O(n \log n)$ time.

The faces of a drawn graph are also simple polygons, and the annotation of a face is the same problem as the annotation of a region on a map. Hence, the problem we address in this paper is motivated by both automated cartography and graph drawing. Figure 1 gives two examples where faces are annotated, and maximizing overlap with a face appears reasonable for the best text box positioning. Annotation—or label placement—has been studied in the context of graph drawing various times, see for instance [6,7,14,19].

If one considers a quadratic time solution to the area of overlap maximization problem to be too slow, there are several approaches to deal with this. Firstly, one can argue that faces in typical graphs do not have large complexity, so an algorithm that takes time quadratic in the number of vertices of the face is no problem. In some cases this is obviously true, like drawings of triangulated graphs. In other cases it is not true, like drawings of trees with a few additional edges or other sparse planar graphs.

Secondly, one can make realistic input assumptions that allows one to show that for inputs satisfying those assumptions, a provably more efficient solution exists. This idea has led to a large body of research in computational geometry. For our problem, this idea does not seem to work. For standard definitions of realistic input polygons [8,20,15], the so-called *placement space* of a unit square remains combinatorially quadratic in size.

Thirdly, one can use approximation. For example, one could try to find the unit square placement that has area overlap with P of at least $c \cdot A$, for some fixed $c \leq 1$, where A is the area of overlap of the optimal placement. Then we have a c-approximation algorithm (which is the optimal algorithm if $c = 1$).

An approximation scheme is an algorithm that, for any $\epsilon > 0$, computes a placement with area overlap of at least $(1-\epsilon)\cdot A$. Approximation algorithms and approximation schemes only make sense if they are significantly more efficient than the corresponding exact algorithm.

It appears hard to develop a subquadratic time approximation algorithm for our problem, due to the fact that the optimal overlap can be very close to 0. However, we can show that if the overlap is at least some constant $\hat{A} > 0$, then we can compute a placement that guarantees an overlap of $(1-\epsilon)\cdot\hat{A}$, for any fixed $\epsilon > 0$. The algorithm runs in $O(\frac{1}{\epsilon}\, n \log^2 n)$ time. Note that assuming that the area of overlap is at least a constant is in a sense a realistic assumption: For instances where the optimal overlap is very small, the solution to the problem is not suitable for a good annotation anyway.

We solve our problem via a detour. We show that for any fixed $\delta > 0$, we can compute a placement of a square of size $1+\delta$ whose area of overlap with the simple polygon P is at least A_{opt}, where A_{opt} is the maximum area of overlap that can be achieved for the placement of a unit square. When we shrink the $(1+\delta)$-square to a unit square, we can lose an area of overlap of at most $2\delta+\delta^2$. Hence, given $\epsilon > 0$ and $\hat{A} > 0$, we choose $\delta = \hat{A}\cdot\epsilon/3$ and compute a placement of a $(1+\delta)$-square with the algorithm we present in this paper. Any unit square inside the $(1+\delta)$-square we found will have area of overlap at least $(1-\epsilon)\cdot A_{\text{opt}}$, so this implies a $(1-\epsilon)$-approximation algorithm.

Our algorithm can be extended to compute the placement of a unit disc with maximum area of overlap approximately in $O(\frac{\log(1/\epsilon)}{\epsilon\sqrt{\epsilon}}\, n \log^2 n)$ time, again assuming that the area of overlap is at least some constant. We note that for this case, no exact algorithm exists at all, due to the algebraic complexity of maximizing the analytic form of the area-of-overlap function. The algorithm can also be extended to place a unit area rectangle with bounded aspect ratio in $O(\frac{1}{\epsilon^2}\, n \log^2 n)$ time (for rectangles with *fixed* aspect ratio one can use the algorithm for squares after scaling). This can be useful for *elastic labels*, an abstraction for text boxes of a fixed length text where the width of the text box is also free proposed by Iturriaga and Lubiw [12,13].

In computational geometry, there is a large body of research on optimal matching of two shapes [21]. One measure for similarity is the area of overlap, and hence, research has been done on maximizing this measure under various transformations. For translations only, Mount *et al.* [17] gave a $O((nm)^2)$ time algorithm for the maximum overlap of a simple n-gon and a simple m-gon. For two convex polygons, an $((n+m)\log(n+m))$ time algorithm exists [2].

There are also several papers that use approximation to find a unit square or disc that covers the maximum number of points of a given point set [9,10]. A main difference with our problem is that we optimize a (continuous) area measure instead of a (discrete) point count measure. Other related research is on finding a largest area rectangle inside a simple polygon, for which Daniels *et al.* [5] give an $O(n \log^2 n)$ time algorithm, and finding the largest similar copy of a convex polygon inside a simple polygon [1]. This would correspond to scaling such that the annotation just fits inside the face. For text boxes this implies changing the

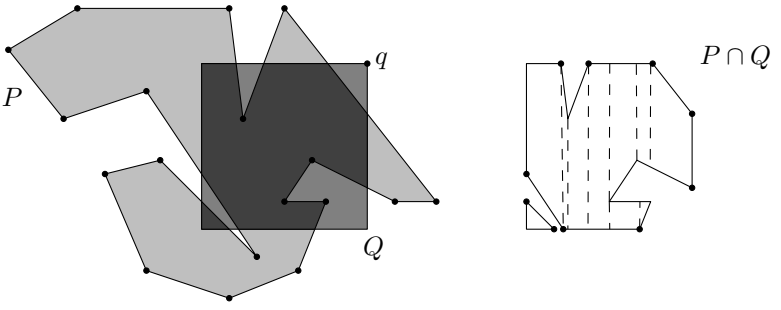

Fig. 2. Example of a simple polygon P intersecting a square Q; the area of intersection can be decomposed into trapezoids

font size, which may not be desirable. Finally, there are many papers on the topic of label placement in the algorithms and automated cartography research fields, but it is beyond the scope of this paper to review them.

We start with a brief description of the quadratic time exact algorithm from [17,16] in Sect. 2 since we will need ideas from it. In Sect. 3 we present the approximation algorithm. We first give a version whose running time is $O(\frac{1}{\epsilon}\, n \log^3 n)$. Then we show how to use Jordan sorting to improve the total running time to $O(\frac{1}{\epsilon}\, n \log^2 n)$. We present the extension for circles and bounded aspect ratio rectangles in Sect. 4. Concluding remarks are given in Sect. 5.

2 An Exact Quadratic-Time Algorithm

In this section we sketch the approach from [17,16] to compute an exact solution to the maximum overlap placement of a unit square on a simple polygon. It is based on the fact that there are quadratically many combinatorially distinct placements of a square Q on a simple polygon P.

The combinatorially distinct placements of Q on P are described by the different pairs of edges—one from Q and one from P—that can intersect. We use the top right corner of Q as a reference point q to characterize the possible positions of Q. When the pairs of intersecting edges are fixed, the reference point still has a little freedom to move, see Fig. 2. As long as the intersecting edges of P and Q remain the same, changing the position of q will change the area of $P \cap Q$, but in a prescribed manner. We can express the area of $P \cap Q$ as a quadratic function in the x- and y-coordinates of the reference point q. Specifically, it has the form:

$$ax^2 + bxy + cy^2 + dx + ey + f\ .$$

This is true because the overlap can be decomposed into a set of trapezoids whose vertices change linearly in x and y, see Fig. 2. Therefore the area changes as a quadratic function. If Q were a circle, the area-of-overlap function would have a non-constant description involving square roots, and maximizing the area of overlap would not be possible exactly.

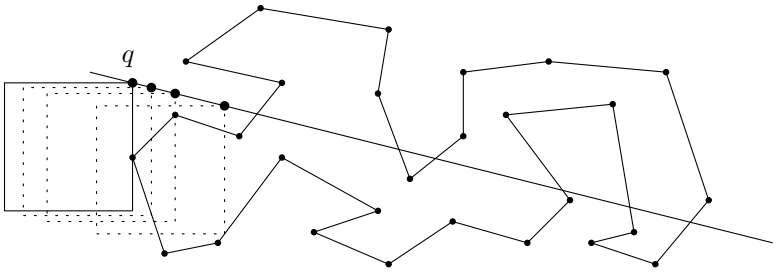

Fig. 3. The 1-dimensional problem of placing a unit square to maximize area of overlap. The first four positions where the area-of-overlap function changes are shown.

Suppose the reference point q and therefore the square Q translates in the plane. The quadratic function giving the area of overlap stops to be valid when the pairs of edges of Q and P that intersect change. Then a different quadratic function will describe the area of overlap, that is, the coefficients a, b, c, d, e, f are different. This happens when:

- an edge of Q passes over a vertex of P, or
- a vertex of Q passes over an edge of P.

Let Π be the subdivision obtained from all positions of q where an edge of Q coincides with a vertex of P, or vice versa. Π is called the placement space of Q with respect to P. In each face of Π, some fixed quadratic function describes the area of overlap of Q and P.

Theorem 1. (Adapted from [17,16]) *Given a simple polygon P with n edges and a square Q, the placement space of Q with respect to P can be constructed in $O(n \log n + N)$ time, where $N = O(n^2)$ is the number of combinatorially distinct placements.*

It can also be shown that the quadratic function that is valid in each cell of Π can be computed in quadratic time by a suitable traversal of the cells of Π. Given Π and the quadratic function for each cell, we can compute the placement of Q that maximizes the area of overlap in $O(N) = O(n^2)$ time.

In case we are only interested in square placements where the reference point is restricted to lie on a given line, the placement space is 1-dimensional and there are only $O(n)$ combinatorially distinct placements, see Fig. 3. The optimal placement can now be solved by a sweep of the square with its reference point on the line, and updating the quadratic function. Since we must sort the $O(n)$ events where the quadratic function changes, this takes $O(n \log n)$ time.

3 An Approximation Algorithm

In this section we compute a placement of a $(1 + \epsilon)$-square on a simple polygon P with n vertices so that the area of overlap is at least the maximum area of

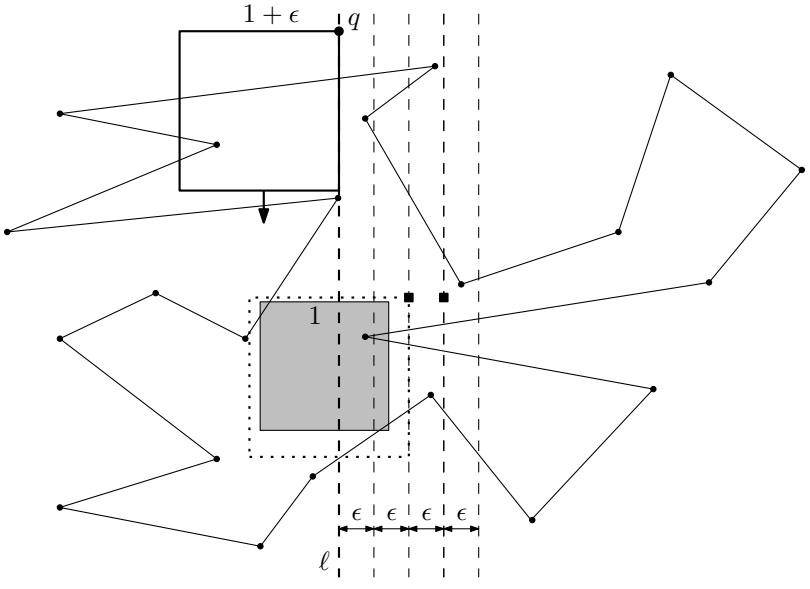

Fig. 4. If the optimal unit square intersects the split line ℓ, then one of the $1/\epsilon$ sweeps will consider $(1+\epsilon)$-squares (like the one shown dotted) that contains it

overlap of a unit square and P. The solution is based on divide-and-conquer, where dividing gives rise to running the 1-dimensional exact algorithm for a $(1+\epsilon)$-square a number of times. The algorithm will at some point consider a $(1+\epsilon)$-square that contains the unit square in its optimal placement. This leads to the desired approximation guarantee.

Divide-and-conquer. The divide-and-conquer algorithm chooses vertical lines to partition the simple polygon into pieces. This will give vertical slabs in the plane. Every split will guarantee that the number of vertices in the interior of the slab is at least halved. So we determine the median of the x-coordinates of the vertices and choose a vertical line ℓ through this vertex.

We would like to create two subpolygons and recurse on them, but it may be that the optimal unit square intersects ℓ, and we must take this possibility into account. This is done by running the 1-dimensional algorithm $1+1/\epsilon$ times.[1] The 1-dimensional algorithm is run with the reference point q of the $(1+\epsilon)$-square on ℓ, and on vertical lines at distances $\epsilon, 2\epsilon, 3\epsilon, \ldots, 1+\epsilon$ to the right of ℓ, see Fig. 4. We observe:

Observation 1. *If the optimal unit square intersects ℓ, then at least one of the $1+1/\epsilon$ sweeps with a $(1+\epsilon)$-square will give a position where a $(1+\epsilon)$-square contains the optimal unit square.*

[1] With slight abuse of notation, we assume that $1+1/\epsilon$ is an integer, but technically we should use rounding. Asymptotically the running time is not affected.

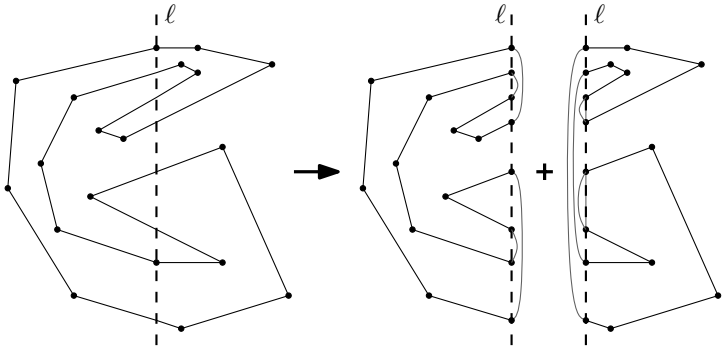

Fig. 5. Splitting a simple polygon by a line and repairing the parts into two simple polygons

Hence, in $O(\frac{1}{\epsilon}\, n \log n)$ time, we find a $(1+\epsilon)$-square whose area of intersection with P is at least as large as the optimal unit square that intersects ℓ.

Splitting the polygon. Now we can divide the problem into two subproblems by splitting the simple polygon. We may split the polygon into more than two parts, but we can repair the situation while using only vertices on the splitting line, see Fig. 5. The resulting polygons may have edges that coincide on the split lines, but this degeneracy does not influence the algorithm. It is standard to perform such a split and repair in $O(n \log n)$ time. Observe that the number of vertices interior to each of the resulting slabs is at least halved. The divide-and-conquer algorithm will find a $(1+\epsilon)$-square strictly left of ℓ recursively, a $(1+\epsilon)$-square strictly right of ℓ recursively, and a $(1+\epsilon)$-square that intersects ℓ. The one with largest area of overlap with P is returned.

Recall that a 1-dimensional sweep has two types of events: An edge of Q passes over a vertex of P, or a vertex of Q passes over an edge of P. In a slab, a subpolygon of P has interior vertices and boundary vertices. We call an edge of P that intersects a slab *short* if it has an endpoint that is an interior vertex, and we call it *long* if both endpoints are boundary vertices. Long edges cross the slab completely. The divide-and-conquer algorithm takes care of halving the number of interior vertices, and therefore the number of short edges is bounded as well. But the number of long edges can become large, and these also give rise to events in the 1-dimensional sweeps. Ultimately, the divide-and-conquer algorithm bottoms out when a slab has no more interior vertices, or when its width is at most unit. The final $O(n)$ slabs may all be crossed by a linear number of edges, leading to an algorithm that takes at least quadratic time in the worst case.

Free splits. To control the number of long edges in recursive subproblems, we use the concept of *free splits*, introduced by Patterson and Yao to prove bounds on the size of binary space partitions [18]. We will take measures to eliminate

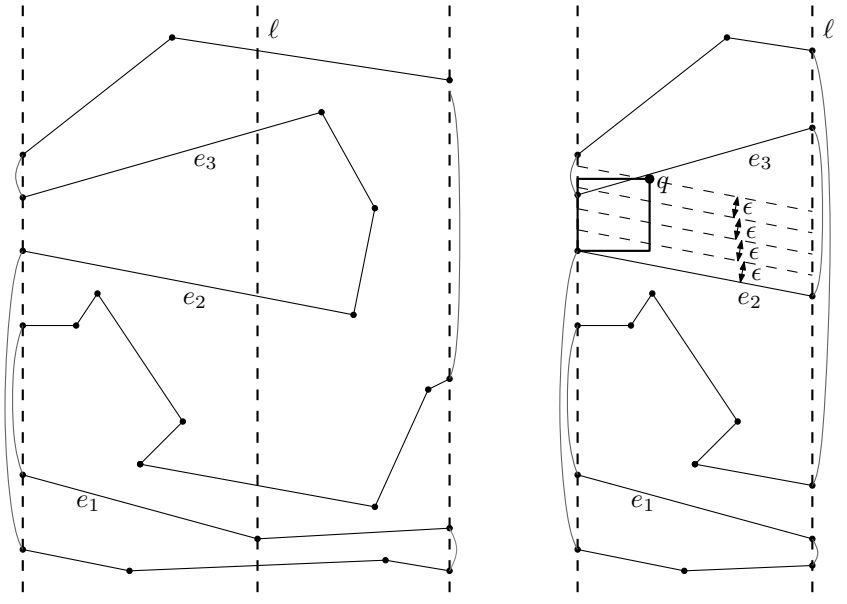

Fig. 6. After splitting along ℓ, the free split along e_2 is done in the left part, and then free splits along e_1 in the bottom left polygon and along e_3 in the top left polygon. Two free splits are performed in the right part as well. In the right figure, the sweeps are shown by dashed segments parallel to e_2.

long edges when they appear after a split with a line ℓ, so that there are no long edges when we choose a next split line.

Let S be a slab with no long edges. We determine the median x-coordinate of the interior vertices, which defines a vertical split line ℓ, and perform $1 + 1/\epsilon$ sweeps as described above. Then we split the simple polygon P into two polygons P_{left} and P_{right}, as described above as well. For each resulting polygon, say, P_{left}, we determine the long edges $e_1, \ldots, e_k$ from bottom to top, see Fig. 6. We use these edges to partition P_{left} further, also in a divide-and-conquer fashion. So we select $e_{\lceil k/2 \rceil}$, perform a number of sweeps parallel to this edge and then split P_{left} at $e_{\lceil k/2 \rceil}$ into two subpolygons that are handled recursively. Since the diameter of a unit square is $\sqrt{2}$, we must now perform up to $\sqrt{2}/\epsilon$ sweeps with q on lines that are a distance ϵ apart, see Fig. 6. Free splits are performed in the same way as splits along vertical lines; no additional cases occur.

Running time. To prove an upper bound on the running time of the algorithm, we first observe that the number of free splits is $O(n \log n)$ throughout the whole algorithm. This is standard; see for instance Chapter 10 in [3], or [4,18].

Lemma 1. *Assume that a slab contains m interior vertices and no long edges. The time needed to perform the vertical split and all necessary free splits is $O(\frac{1}{\epsilon}\, m \log^2 m)$, including the time for the sweeps.*

Proof. We already argued that the vertical split takes $O(\frac{1}{\epsilon} m \log m)$ time. We may have created $O(m)$ long edges while doing this. For each free split, each of the $O(m)$ interior and boundary vertices appears in only one of the two new polygons that is created. Each vertex creates an event in $O(\frac{1}{\epsilon} \log m)$ 1-dimensional sweeps, because the recursion depth of the free splits is $O(\log m)$. Hence, the $O(\frac{1}{\epsilon} m)$ sweeps due to free splits encounter $O(\frac{1}{\epsilon} m \log m)$ events together, leading to $O(\frac{1}{\epsilon} m \log^2 m)$ time for all free splits. □

The lemma can be used to prove an $O(\frac{1}{\epsilon} n \log^3 n)$ time bound for the approximation problem: The recurrence that describes the efficiency of the algorithm is given by $T(n) = 2 \cdot T(n/2) + O(\frac{1}{\epsilon} n \log^2 n)$ for $n > 1$ (and $T(1) = O(1)$), which solves to $O(\frac{1}{\epsilon} n \log^3 n)$ time. However, we can remove a logarithmic factor.

Jordan sorting. We next improve the overall running time to $O(\frac{1}{\epsilon} n \log^2 n)$ by applying Jordan sorting to the 1-dimensional problems. Jordan sorting is a linear time algorithm which, given a simple polygon and a line, sorts their intersection points along the line [11].

The 1-dimensional sweep algorithm to find the $(1 + \epsilon)$-square that has the largest area of overlap with P takes $O(n \log n)$ time due to the sorting of the events. If the events were sorted, we could update the quadratic function in constant time because at most three trapezoids can appear or disappear during an event. We only have to perform some simple additions to the coefficients a, b, c, d, e, and f based on these changed trapezoids to get the new quadratic function that is valid. This fact was already used in [16] to generate the placement space with all quadratic functions.

To obtain a sorted list of events, recall that there are two types of events: an edge of Q crosses an vertex of P, and a vertex of Q passes a edge of P. The former type will be obtained in sorted order by pre-sorting and maintaining two sorted lists, the second type by Jordan sorting.

For the first type, we perform preprocessing for the algorithm by sorting all vertices of P by x-coordinate into a list L_x, and also by y-coordinate into a list L_y. Whenever we perform a split, by a vertical line or a free split, we traverse each list and generate two new sorted lists for the two subproblems that appear. This will take time linear in the length of the list, which is linear in the number vertices in the slab or trapezoid that is split.

For the second type, we compute the event just before performing the 1-dimensional sweep. We perform Jordan sorting four times, once for each path of a vertex of Q; this path is a line segment. We merge these sorted lists into one, and also merge them with the events of the first type. In total, we need six list merges to obtain all events in sorted order. Hence, a 1-dimensional sweep can be performed in linear time.

Summarizing the results of this section, we conclude:

Theorem 2. *Given a simple polygon P with n vertices and a constant $\epsilon > 0$, an $O(\frac{1}{\epsilon} n \log^2 n)$ time algorithm exists that computes a placement of an axis-aligned square with side length $1 + \epsilon$ of which the area of overlap with P is at least the area of overlap of any axis-aligned unit square with P.*

4 Extensions to Circles and Rectangles

Suppose we wish to place a circular annotation on a region of a map or a face drawn graph, like a pie chart. We can adapt the approximation algorithm given in previous section to this case. The idea is to choose a suitable integer k, and adapt the algorithm for squares to work for regular k-gons.

We will use a regular k-gon that is inside a diameter $(1+\epsilon)$-disc but outside a $(1+\frac{\epsilon}{2})$-disc. Instead of performing 1-dimensional sweeps with a distance ϵ in between, we must use a distance of $\epsilon/2$ in between. Finally, we need to maintain k sorted lists that give the order in which the edges of the regular k-gon will cross the vertices of subpolygons of P. The extensions are straightforward.

It is well known that the choice $k = \Theta(1/\sqrt{\epsilon})$ satisfies our requirement of approximating a disc well enough. Following the analysis for the square case, we notice that merging k sorted lists with $O(nk)$ events in total takes $O(nk \log k)$ time. Hence, we conclude:

Theorem 3. *Given a simple polygon P with n vertices and a constant $\epsilon > 0$, an $O(\frac{\log(1/\epsilon)}{\epsilon\sqrt{\epsilon}}\, n \log^2 n)$ time algorithm exists that computes a placement of a disc with diameter $1+\epsilon$ of which the area of overlap with P is at least the area of overlap of any unit disc with P.*

Next we discuss the extension to placing an axis-aligned rectangle with aspect ratio $r : 1$ or less and unit area (assuming $r \geq 1$). If the aspect ratio were fixed, we could simply scale the input so that the problem reduces to placing an axis-aligned square. Our algorithm will test a fixed number of aspect ratios (depending on ϵ and r), scale the input appropriately, and run the square placement algorithm. We will asume that $r = O(1)$ since this will be true in any practical context.

Suppose that the optimal unit area rectangle is R_{opt}. Then we wish to find a rectangle with area at most $1+\epsilon$, aspect ratio at most $r : 1$, and that has area of overlap with P that is at least as much as the area of overlap of R_{opt}. We must make sure to that our algorithm tries some rectangle during a sweep that contains the optimal rectangle R_{opt}.

We will try the following rectangle widths (or heights): $(1+\frac{3\epsilon}{5r}), (1+\frac{4\epsilon}{5r}), (1+\frac{\epsilon}{r}), (1+\frac{6\epsilon}{5r}), \ldots$ and the corresponding heights (resp. widths) to get an area of $1+\epsilon$; these corresponding heights (resp. widths) increase by less than $\epsilon/(5r)$. We continue until the first value greater than $\sqrt{(1+\epsilon)r}$.

One of the rectangles we try will be larger by $\epsilon/(5r)$ in height and width than R_{opt} but at most larger by $2\epsilon/(5r)$. If the optimal rectangle R_{opt} has size $h \times (1/h)$, then $1 \leq h \leq \sqrt{r}$, and a rectangle of size $(h+\frac{2\epsilon}{5r}) \times ((1/h)+\frac{2\epsilon}{5r})$ has area less than $1+\epsilon$.

If we run the 1-dimensional sweeps with lines at distance $\epsilon/(5r)$ in between, then we will encounter a rectangle with the desired properties that contains R_{opt}. Since r is assumed to be constant, we run the algorithm for squares $O(1/\epsilon)$ times. We conclude:

Theorem 4. *Given a simple polygon P with n vertices, a constant $\epsilon > 0$, and a value $r \geq 1$, an $O(\frac{1}{\epsilon^2}\, n \log^2 n)$ time algorithm exists that computes a placement*

of an axis-aligned rectangle with area $1+\epsilon$ *and aspect ratio bounded by* $r:1$ *of which the area of overlap with* P *is at least the area of overlap of any axis-aligned unit area rectangle and aspect ratio bounded by* $r:1$ *(assuming* $r = O(1)$*).*

5 Concluding Remarks

We have studied the problem of annotating a region on a map or a face of a drawn graph by a square, circular, or rectangular shape while maximizing the area of overlap. This will give a good association between the face and the shape, may avoid unnecessary covering of edges by the annotation, and if more faces are annotated, may help to avoid overlap of different annotations. It was known that the problem can be solved in $O(n^2)$ time if the face has n boundary vertices. We showed that a placement of a shape that is larger by a factor $1+\epsilon$ can be found in $O(n \log^2 n)$ time that has at least the area of overlap of the optimal placement of the original shape (ignoring factors depending on the constant $\epsilon > 0$).

With the same approach, we can compute a placement whose length of overlap with the boundary of the face is minimized. For this problem to make sense, we must restrict the space of all placements somehow, otherwise the optimum can lie fully outside the face. We can for instance require that the center of the shape lies inside the proper face. For each combinatorially distinct placement, the length of overlap changes linearly in the coordinates of the reference point, but otherwise, the solution approach is the same, and we get the same running time bounds.

The main open problems are improvements in the running time. Firstly, we suspect that it must be possible to remove one log-factor from the running time, but it is even conceivable that both log-factors can be removed. For the disc and rectangle versions, we may be able to improve the dependency on ϵ, or generalize to rectangles of unbounded aspect ratio.

Acknowledgements

The authors thank Sariel Har-Peled for sharing his ideas that lead to the solution.

References

1. Baker, B., Fortune, S., Mahaney, S.: Polygon containment under translation. J. Algorithms 7(4), 532–548 (1986)
2. de Berg, M., Cheong, O., Devillers, O., van Kreveld, M., Teillaud, M.: Computing the maximum overlap of two convex polygons under translations. Theory Comput. Syst. 31(5), 613–628 (1998)
3. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry – Algorithms and Applications, 3rd edn. Springer, Berlin (2008)
4. Chazelle, B., Edelsbrunner, H., Guibas, L., Sharir, M.: Algorithms for bichromatic line-segment problems and polyhedral terrains. Algorithmica 11(2), 116–132 (1994)

5. Daniels, K., Milenkovic, V., Roth, D.: Finding the largest area axis-parallel rectangle in a polygon. Comput. Geom. Theory Appl. 7, 125–148 (1997)
6. Dogrusöz, U., Feng, Q.W., Madden, B., Doorley, M., Frick, A.: Graph visualization toolkits. IEEE Computer Graphics and Appl. 22(1), 30–37 (2002)
7. Dogrusöz, U., Kakoulis, K., Madden, B., Tollis, I.: On labeling in graph visualization. Inf. Sci. 177(12), 2459–2472 (2007)
8. Efrat, A., Sharir, M.: On the complexity of the union of fat convex objects in the plane. Discr. & Comput. Geometry 23(2), 171–189 (2000)
9. Gudmundsson, J., van Kreveld, M., Speckmann, B.: Efficient detection of patterns in 2D trajectories of moving points. GeoInformatica 11, 195–215 (2007)
10. Har-Peled, S., Mazumdar, S.: Fast algorithms for computing the smallest k-enclosing circle. Algorithmica 41(3), 147–157 (2005)
11. Hoffman, K., Mehlhorn, K., Rosenstiehl, P., Tarjan, R.: Sorting jordan sequences in linear time using level-linked search trees. Information and Control 68(1–3), 170–184 (1986)
12. Iturriaga, C., Lubiw, A.: Elastic labels: the two-axis case. In: DiBattista, G. (ed.) GD 1997. LNCS, vol. 1353, pp. 181–192. Springer, Heidelberg (1997)
13. Iturriaga, C., Lubiw, A.: Elastic labels around the perimeter of a map. J. Algorithms 47(1), 14–39 (2003)
14. Kakoulis, K., Tollis, I.: A unified approach to automatic label placement. Int. J. Comput. Geometry Appl. 13(1), 23–60 (2003)
15. van Kreveld, M.: On fat partitioning, fat covering, and the union size of polygons. Comput. Geom. Theory Appl. 9, 197–210 (1998)
16. van Kreveld, M., Schramm, E., Wolff, A.: Algorithms for the placement of diagrams on maps. In: GIS 2004: Proc. 12th annu. ACM Int. Symp. on Advances in Geographic Information Systems, pp. 222–231. ACM Press, New York (2004)
17. Mount, D., Silverman, R., Wu, A.: On the area of overlap of translated polygons. Computer Vision and Image Understanding 64(1), 53–61 (1996)
18. Paterson, M., Yao, F.: Binary partitions with applications to hidden surface removal and solid modelling. In: Proc. 5th annual ACM Symp. on Computational Geometry, pp. 23–32 (1989)
19. Ryall, K., Marks, J., Shieber, S.: An interactive system for drawing graphs. In: North, S.C. (ed.) GD 1996. LNCS, vol. 1190, pp. 387–394. Springer, Heidelberg (1997)
20. van der Stappen, A.: Motion Planning amidst Fat Obstacles. Ph.D. thesis, Department of Computer Science, Utrecht University (1994)
21. Veltkamp, R., Hagedoorn, M.: State of the art in shape matching. In: Lew, M. (ed.) Principles of Visual Information Retrieval, pp. 87–119. Springer, Heidelberg (2000)