1984

On Symmetry Detection

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Report Number:
84-476

ON SYMMETRY DETECTION

Mikhail J. Atallah

Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907.

TR-476 3-26-1984

Abstract

A straight line is an *axis of symmetry* of a planar figure if the figure is invariant to reflection with respect to that line. The purpose of this note is to describe an $O(n \log n)$ time algorithm for enumerating all the axes of symmetry of a planar figure which is made up of (possibly intersecting) segments, circles, points, ...etc.

Index Terms: Analysis of algorithms, computational geometry, axis of symmetry, centroid, string pattern matching.

1. Introduction

A straight line is an *axis of symmetry* of a planar figure if the figure is invariant to reflection with respect to that line. For example, in Figure 1.1, line L is an axis of symmetry. The problem we consider is relevant to both computational geometry and pattern recognition: Given a planar figure which consists of a collection of n points, segments, circles, ellipses, ...etc, enumerate all the axes of symmetry of that figure. We give an $O(n \log n)$ time algorithm for this problem.

The input to the algorithm consists of a description of the planar figure under consideration. This description is a collection of linked lists, each of which contains the occurrences in the figure of one type of geometric pattern (i.e. one list contains the points, another list contains the segments, ...etc). The contents of each list are not given in any particular order. The output produced by the algorithm is a (possibly empty) collection of straight lines each of which is an axis of symmetry of the input figure. Note that there cannot be more than n axes of symmetry, so that the size of the output is $O(n)$.

Let S_p be the set of axes of symmetry of the subfigure which consists of only the points of the input figure. The set S_s is analogously defined for segments (we assume for the time being that the figure is made up only of points and segments). Since any reflection maps points into points and segments into segments, it follows that the output of the algorithm is the intersection of the sets S_p and S_s. This observation implies that it is sufficient to show that each of S_p and S_s can be found in time $O(n \log n)$.

The following is an outline of the paper. Section 2 shows that S_p can be computed in time $O(n \log n)$, Section 3 gives an analogous result for S_s, Section 4 briefly sketches how the results can be generalized for figures which include other geometric patterns (such as circles, ellipses, and others), and Section 5 concludes.

We now briefly introduce some conventions and terminology.

Recall the definition of *centroid*: Given a set of points $p_1, \ldots, p_m$ such that with every p_i is associated a weight $w_i > 0$, the centroid of this weighted system of points is the unique point C such that $\sum_{i=1}^{m} w_i \vec{Cp_i} = \vec{0}$.

Finally, we should point out that we deliberately refrained from including in our algorithm various improvements which would not improve its worst-case complexity, but may improve its performance in practice. We did this to avoid unnecessarily cluttering the exposition. In the conclusion (Section 5) we briefly sketch some of these possible practical improvements.

2. Computing S_p

Let C be the centroid of the subfigure which consists of only the points of the input figure, when every such point is given a weight equal to unity. Observe that any line in S_p must pass through C. Note also that a reflection about any line in S_p must map a point whose distance from C is d into a point whose distance from C is also d. This suggests the following approach for computing S_p:

(i) Partition the points into equivalence classes $E_1, \ldots, E_k$ such that all the points in a class E_i are at the same distance (call it d_i) from C,

(ii) For each class E_i, find the set $S_{p,i}$ of lines passing through C and leaving E_i invariant to reflection about them,

(iii) Set S_p equal to the intersection of the $S_{p,i}$'s.

It is clear that Steps (i) and (iii) can be done in time $O(n\log n)$. To show that Step (ii) can be done in time $O(n\log n)$, we need only show that every $S_{p,i}$ can be computed in time $O(n_i\log n_i)$, where $n_i = |E_i|$. The algorithm for doing this follows:

Algorithm for computing $S_{p,i}$

Step 1: Obtain from E_i a string σ over the (infinite) alphabet consisting of the real numbers and the special symbol #, as follows. Start with an empty σ and imagine an axis W revolving counterclockwise about C, starting from a position where it also passes through some point in E_i. As W revolves by 360 degrees about C, we create σ as follows. Whenever W encounters a point in E_i we do $\sigma \leftarrow \sigma\#\#$, and whenever W sweeps the angle θ separating two such encounters we do $\sigma \leftarrow \sigma\theta\theta$. For example, the set E_i shown on Figure 2.1 would result in

$$\sigma = \#\#\theta\theta\#\#\alpha\alpha\#\#\gamma\gamma\#\#\alpha\alpha\ .$$

Note that σ always has even length.

Why we create σ in this way becomes clear once the following crucial observation is made: There is a one to one correspondence between the lines in $S_{p,i}$ and the circular rotations which turn σ into a palindrome (a palindrome is a string which is equal to its reverse). For example, rotating the string σ corresponding to Figure 2.1 by three positions to the left turns it into the palindrome

$$\theta\#\#\alpha\alpha\#\#\gamma\gamma\#\#\alpha\alpha\#\#\theta$$

and this corresponds to a line L in $S_{p,i}$ (see Figure 2.1). This observation implies that computing $S_{p,i}$ reduces to enumerating all the rotations which turn σ into a palindrome. This is what the next Step does.

Step 2: Compute $S_{p,i}$ by enumerating all rotations which turn σ into a palindrome.

End of algorithm

Correctness of the algorithm follows from the comment following Step 1. Step 1 can clearly be implemented in time $O(n_i \log n_i)$ by sorting the points in E_i according to the order in which W encounters them when revolving about C. We now show that Step 2 can be implemented to run in $O(n_i)$ time. In what follows, $|x|$ denotes the length of a string x, and x^R denotes its reverse.

Given a string $x = a_1 \ldots a_{2m}$, rotating x by i positions to the left turns it into a palindrome if and only if $x = uu^R ww^R$ where $|u| = i$. Therefore, enumerating all the rotations which turn x into a palindrome is equivalent to enumerating all j such that each of the two strings $a_1 \ldots a_{2j}$ and $a_{2j+1} \ldots a_{2m}$ is a palindrome. This last problem is equivalent to finding all occurrences of x at even positions in $y = x^R x^R$, which can be done in time $O(|x|)$ using well-known techniques [AHU]. This shows that Step 2 can be done in time $O(n_i)$, and therefore computing $S_{p,i}$ can be done in time $O(n_i \log n_i)$. As previously mentioned, this implies that S_p can be computed in time $O(n \log n)$.

The next Section considers the somewhat trickier problem of computing R_s in time $O(n \log n)$.

3. Computing S_s

Let $\hat{C}$ be the centroid of the set of midpoints of the segments when every such midpoint is given a weight equal to the length of the corresponding segment. Observe that any line in S_s must pass through $\hat{C}$.

If $\hat{C} \neq C$, where C is as defined in Section 2, then there is no need to compute S_s because in this case the problem is practically solved: The only possible axis of symmetry of the planar figure is the line joining C to $\hat{C}$ and therefore we need only check whether that line is indeed an axis of symmetry of the input figure. However, in the worst case, $\hat{C}$ will coincide with C. Therefore, for the rest of this section, we assume that $\hat{C}$ coincides with C and therefore there is a need to compute S_s.

Let the *triple* of a segment of the figure be (l, d_1, d_2), $d_1 \le d_2$, where l is the length of the segment and where d_1 and d_2 are the distances between C and the segment's two endpoints. Observe that a reflection about any line in S_s maps a segment whose triple is (l, d_1, d_2) into a segment whose triple is also (l, d_1, d_2). This suggests the following approach for computing S_s:

(i) Partition the segments into equivalence classes $F_1, \ldots, F_k$ such that all the segments in a class F_i have the same triple,

(ii) For each class F_i, find the set $S_{s,i}$ of lines passing through C and leaving F_i invariant to reflection about them,

(iii) Set S_s equal to the intersection of the $S_{s,i}$'s.

It is clear that Steps (i) and (iii) can be done in time $O(n \log n)$. To show that Step (ii) can also be done in time $O(n \log n)$, we need only show that every $S_{s,i}$ can be computed in time $O(n_i \log n_i)$, where $n_i = |F_i|$. The algorithm for doing this follows:

Algorithm for computing $S_{s,i}$

Let the triple of the segments in F_i be (l, d_1, d_2). We distinguish three cases:

Case I: $d_1 \neq d_2$ and $l > d_2 - d_1$ (see Figure 3.1).

If $|F_i|$ is an odd number then set $S_{s,i} \leftarrow \varnothing$ and Stop, otherwise compute $S_{s,i}$ in the following way.

First, obtain from F_i a string σ over the (infinite) alphabet consisting of the real numbers and the special symbols # and &. Before describing how σ is created, we introduce some terminology. Let the *representative* of a segment in F_i be its endpoint whose distance to C is d_1. In Figure 3.1, the representatives are the six endpoints on the smaller circle. If x is a string, we use $\bar{x}$ to denote the string obtained by replacing in x every & symbol by #, and every # symbol by & (for example, $\overline{\#\&\#} = \&\#\&$). A string x is a *pseudo-palindrome* if $x = \bar{x}^R$. For example, &#&# is a pseudo-palindrome.

Start with an empty σ and imagine an axis W revolving counterclockwise about C, starting from a position where it also passes through the representative of some segment in F_i. As W revolves by 360 degrees about C, we create σ as follows. Whenever W encounters a representative of a segment in F_i, we do

(i) $\sigma \leftarrow \sigma\#$ if the segment is to the left of W at the time of the encounter,

(ii) $\sigma \leftarrow \sigma\&$ if the segment is to the right of W at the time of the encounter.

Whenever W sweeps the angle θ separating two such encounters we do $\sigma \leftarrow \sigma\theta\theta$. For example, the situation depicted in Figure 3.1 would result in

$$\sigma = \&\alpha\alpha\#\beta\beta\&\gamma\gamma\&\theta\theta\#\gamma\gamma\#\beta\beta\,.$$

Note that the resulting σ always has even length, since $|F_i|$ is even.

The crucial observation to be made here is that there is a one to one correspondence between the lines in $S_{s,i}$ and the circular rotations which turn σ into a pseudo-palindrome. For example, rotating the string σ corresponding to Figure 3.1 by two positions to the left turns it into the pseudo-palindrome

$$\alpha\#\beta\beta\&\gamma\gamma\&\theta\theta\#\gamma\gamma\#\beta\beta\&\alpha,$$

and this corresponds to a line L in $S_{s,i}$ (see Figure 3.1). This observation implies that computing $S_{s,i}$ reduces to enumerating all the rotations which turn σ into a pseudo-palindrome, which is equivalent to finding all occurences of σ at even positions in $\bar{\sigma}^R\bar{\sigma}^R$.

Case II: $d_1 = d_2$ (see Figure 3.2).

Let FP_i denote the set of midpoints of the segments in F_i. Now, observe that in this case $S_{s,i}$ is precisely the set of lines passing through C and leaving FP_i invariant to reflection about them. Therefore the techniques of Section 2 can be used for computing $S_{s,i}$.

Case III: $d_1 \neq d_2$ and $l = d_2 - d_1$ (see Figure 3.3).

The same remarks as in Case II hold.

End of algorithm

Correctness of Cases II and III is obvious, while correctness of Case I follows from the observation following it. That Cases II and III can be done in time $O(n_i \log n_i)$ was shown in Section 2. Therefore we need only show that Case I can also be done in time $O(n_i \log n_i)$. Creating σ in Case I can clearly be done in time $O(n_i \log n_i)$. Once we have σ we can find $S_{s,i}$ in time $O(n_i)$, because finding all occurrences of σ at even positions in $\bar{\sigma}^R \bar{\sigma}^R$ can be done in time $O(n_i)$.

This shows that every $S_{s,i}$ can be computed in time $O(n_i \log n_i)$. As previously mentioned, this implies that S_s can be computed in time $O(n \log n)$.

4. Other Geometric Patterns

In this section we briefly sketch how the techniques of the last two sections can be generalized to figures that include other geometric patterns in addition to points and segments.

If the input figure includes circles, then we replace every circle by its center and give that center a weight equal to the radius of the circle. If the centroid of the weighted set of centers does not coincide with C then we need only check whether the line joining it to C is an axis of symmetry of the figure. Otherwise (i.e. if the centroid coincides with C) we must compute the set S_c of lines which pass through C and which leave the set of weighted centers invariant to reflection about them. The algorithm of Section 2 can be modified to work for weighted sets of points, and therefore it can be used on the weighted set of centers in order to compute S_c in time $O(n \log n)$ (the details of these modifications are left to the reader).

If the input figure includes (non-degenerate) ellipses, then, as in the case of circles, the problem can be shown to boil down to computing the set S_e of lines passing through C and which are axes of symmetry of the subfigure which consists of only the ellipses. S_e is computed as follows. First, the ellipses are partitioned into sets H_1, ... ,H_k where all the ellipses in a set H_i have same major axis length and same minor axis length. Let $n_i = |H_i|$. As in Section 3, the problem becomes that of computing, in time $O(n_i \log n_i)$, the set $S_{e,i}$ of lines passing through C and leaving H_i invariant to reflection about them. If we replace every ellipse in H_i by a segment coinciding with its major axis, then we can use the algorithm of Section 3 on these segments to compute $S_{e,i}$ in time $O(n_i \log n_i)$.

Similar techniques can be used when the input figure includes other geometric patterns as well (e.g. portions of circles, or of ellipses, or of parabolas ...etc). We omit the details of these since they involve no new ideas.

5. Conclusion

We gave an $O(n \log n)$ time algorithm for computing the (possibly empty) set of axes of symmetry of a planar figure made up of a n (possibly intersecting) segments, circles, ellipses, points ...etc.

As previously stated, there are improvements to the algorithm which, although they do not change its worst-case time complexity, may in practice improve its speed. All these improvements are attempts to find a point *distinct from* C through which any axis of symmetry of the input figure must pass. If we succeed in finding such a point $\hat{C}$ then we need only check whether the unique line through C and $\hat{C}$ is an axis of symmetry. Such verification still takes time $O(n \log n)$, but the constant factor hiding behind the 'O' would then be smaller than that of the algorithms of Sections 2, 3 and 4. We give below a few examples ((i) to (iv)) of possible attempts at finding such a point $\hat{C}$. The reader should convince himself that, for each of (i)-(iv) below, any axis of symmetry of the input figure must pass through $\hat{C}$:

(i) $\hat{C}$ is the centroid of the system of points consisting of the midpoints of the segments of the input figure when every such midpoint has a weight equal to unity.

(ii) $\hat{C}$ is the centroid of the weighted system of points consisting of the centers of the circles of the input figure when every such center has a weight equal to unity.

(iii) Same as (i) with the weight equal to the square of the segment length.

(iv) Same as (ii) with the weight equal to the square of the radius.

The reader can probably think of many more additional examples of such points $\hat{C}$ through which any axis of symmetry of the figure must pass.

Of course, in the worst case, it is possible that none of examples (i) to (iv) above succeeds in producing a point $\hat{C}$ which is distinct from C, in which case we would have to use the algorithms of Sections 2,3 and 4. One such "worst case" is when the input figure is made up of only one type of geometric pattern, e.g. only points. However, when the figure is made up of more than one geometric pattern, then it is quite likely that one of the points $\hat{C}$ mentioned above will be distinct from C.

Acknowledgement: The author sincerely thanks S.R. Kosaraju for many helpful comments.

References

[AHU] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

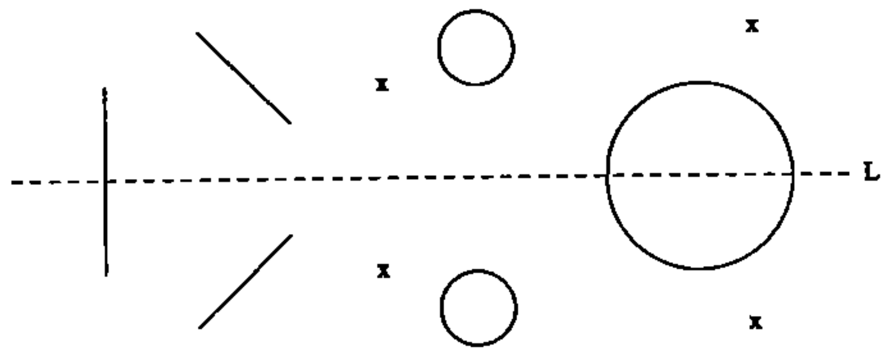

Figure 1.1: Line L is an axis of symmetry.

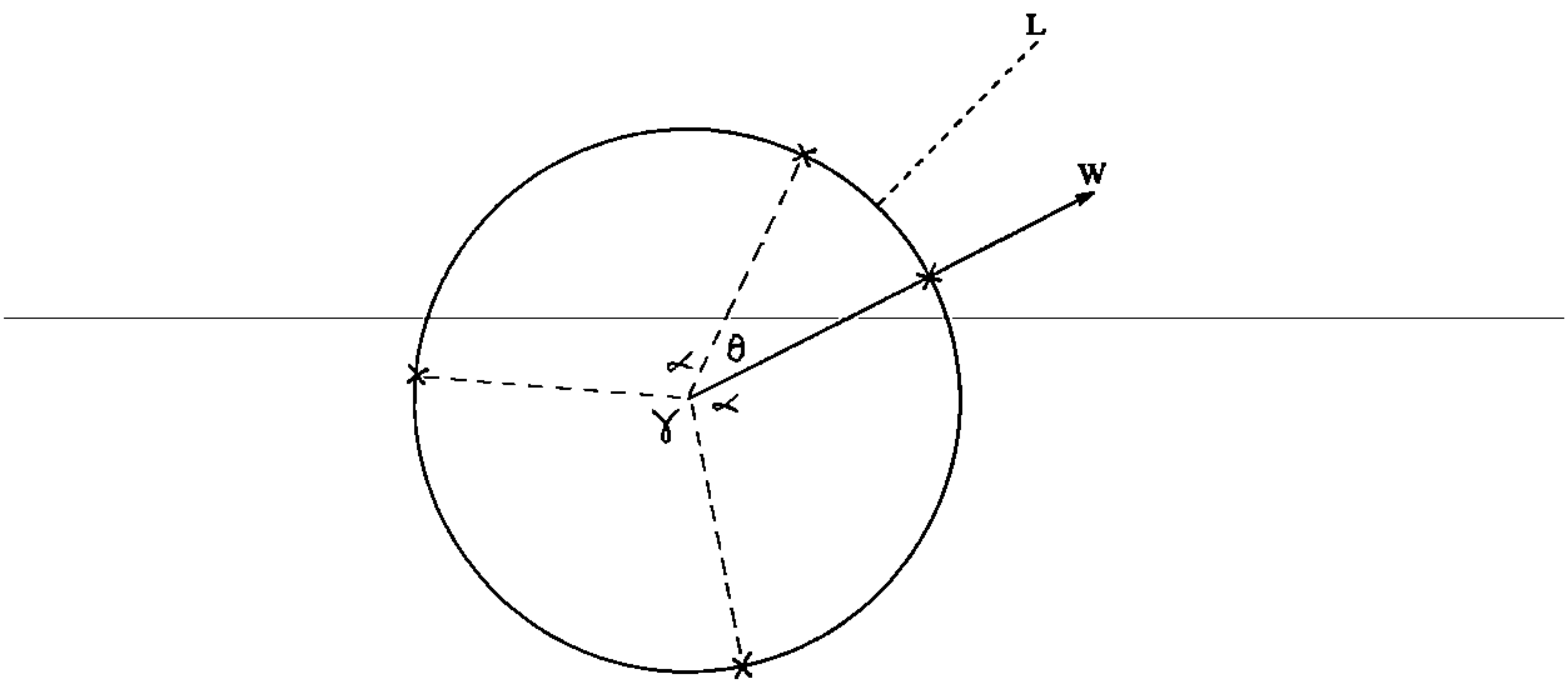

Figure 2.1: The points of E_L are on a circle centered at C. W rotates counterclockwise about C.

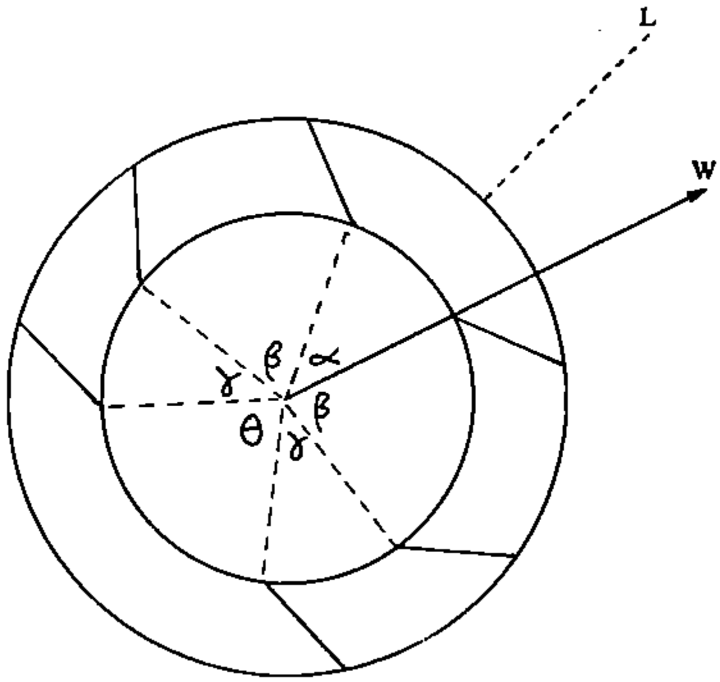

Figure 3.1: Illustrating Case I. The endpoints of the segments in F_L lie on two circles centered at C. W rotates counterclockwise about C.

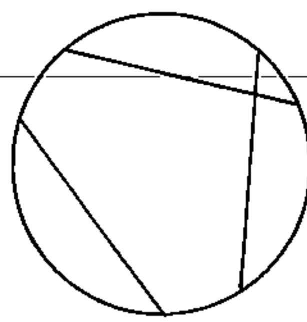

Figure 3.2: Illustrating Case II, when the two endpoints of segments in F_L are equidistant from C.

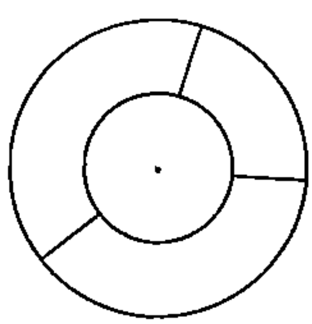

Figure 3.3: Illustrating Case III, when $l=d_2-d_1$.