Minimal Enclosing Circle

url: http://www.cs.mcgill.ca/~cs507/projects/1998/jacob/problem.html

Problem Statement

The Minimal Enclosing Circle Problem is, simply stated, the problem of finding the smallest circle that completely contains a set of points.

More formally: **Given a set S of n points in the plane, find the circle C of smallest radius with the property that all points in S are contained in C or its boundary.**

Applications

The minimal enclosing circle is useful when we want to plan the location of a shared facility. Consider, for example, a hospital servicing a number of communities. If we think of the communities as points in the plane, finding their minimal enclosing circle gives a good place to put the hospital: the circle's center. Placing the hospital at the point defined by the circle's center minimizes the distance between the hospital and the farthest community from it.

In the military, the minimal enclosing circle is known as the Bomb Problem. If the points in the set are viewed as targets on a map, the center of the minimal circle surrounding them is a good place to drop a bomb to destroy the targets, and the circle's radius can be used to calculate how much explosive is required.

If we want to rotate a set of points into any arbitrary alignment, e.g. rotate the shape to make the line connecting some two chosen points vertical, then the minimal enclosing circle of the points is the amount of space in the plane that must be empty to permit this rotation.

The minimal enclosing circle provides a rough approximation to its points. In other words, any application involving a test of proximity between sets of points and a new point could benefit from having computed the minimal enclosing circles of the point sets to quickly determine whether the new point is close to a given set.

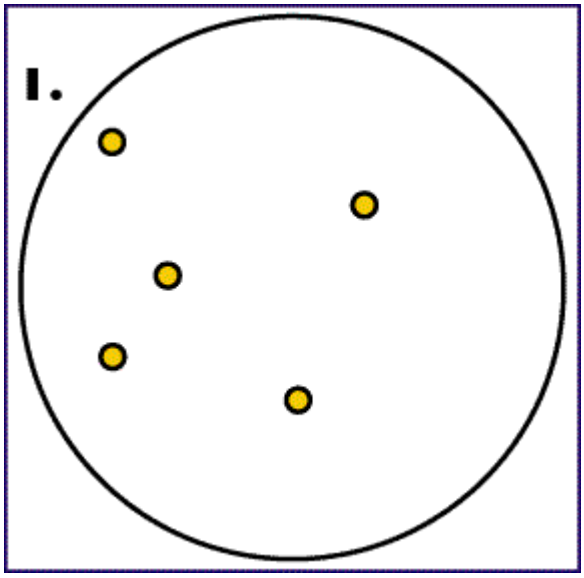
It can also be useful to examine the points which lie on the boundary of the minimal enclosing circle. These points are in a sense the outliers of the set, and in statistics they are sometimes discarded so that estimates are more robust. This discarding process can be repeated to make the data set more concentrated.

The points which lie on the boundary of the minimal enclosing circle are The diameter of the minimal enclosing circle provides an approximation to the span of the point set. It can be shown that it is no more than a factor of `sqrt(3)` larger than the span.
(NB: The span of a point set is the maximum distance between two points in the set.)

---

by Jacob Eliosoff and Richard Unger, Oct 1998

An Intuitive Solution



1. Draw any circle around all the points. This circle can clearly be made smaller.
2. Make your circle smaller by finding the point A farthest from your circle's center, and drawing a new circle with the same center and passing through the point A. This produces a smaller enclosing circle, since it still contains all the points, but now passes through A rather than enclosing it.
3. If the circle passes through 2 or more points, proceed to step 4. Otherwise, make the circle smaller by moving its center towards point A, until the circle makes contact with another point B from the set.
4. At this stage, you will have a circle, C, passing through two or more points from the set. If the circle contains an interval of arc greater than half the circle's circumference on which no points lie, the circle can be made smaller. We will call such an interval a *point-free interval*. Let D and E be the points on the ends of this point-free interval. While keeping D and E on the circle's boundary, reduce the diameter of the circle until
    a. the diameter is the distance DE, or
    b. the circle C touches another point from the set, F.

    In case a), we are finished. In case b), we must check whether there are now no point-free arc intervals of length more than half the circumference of C. If there are none, we are done. If there does exist such a point-free interval, we need to repeat step 4. In this case, three points must lie on an arc less than half the circumference in length. We repeat step 4 on the outer two of the three points on the arc.

This solution is very simple to visualize, but expensive to implement. The solution is presented along with a full proof of correctness in (ref).

Step 1 requires linear time in the number of points in our set, as does step 2. Step 3 is also linear in the number of points. In step 4 above, it takes linear time to find each new point F.
However, finding the point F does not guarantee the termination of the algorithm. Step 4 must be repeated until the circle contains no point-free interval longer than half its circumference.

---

by Jacob Eliosoff and Richard Unger, Oct 1998

In the worst case, this requires n-2 iterations of step 4, implying that the total time spent in step 4 could be on the order of the square of the size of the point set.

Hence this algorithm is $O(n^2)$.

Modern Solutions

This section is divided into two parts. In the first part, we describe the algorithm we actually used in our applet to compute minimal enclosing circles. In the second part we present the best known algorithm for finding MECs. This is the algorithm we would have used were we dealing with huge data sets, or had we unlimited time to complete this project

Our Algorithm

The algorithm we decided to use dates back as far as 1885 and was proposed by Pr. Chrystal in the proceedings of the third meeting of the Edinburgh Mathematical Society.

The algorithm proceeds as follows:

First we observe that the MEC is entirely determined by the **Convex Hull** of our point set. This is because the points of the set touched by the MEC are always on the convex hull of the set. Hence we first compute the convex hull of the points. This is done in linear time, as the points are kept ordered by x-coordinate. Call the convex hull **H** and the number of convex hull vertices **h**.
Our next step is to pick any side of the convex hull. Call this side **S**. We are now ready to describe the main body of the algorithm:

Main Loop:

1. For each vertex of H other than those of S, we compute the angle subtended by S. The minimum such angle occurs at vertex **v**, and the value of the minimum angle is **a**.
    - If a is larger than or equal to 90 degrees, then we are finished, and the MEC is determined by the diametric circle of S.
    - If a is less than 90 degrees, then we are not yet done, and must proceed to step 2 below.

2. Since a was less than 90 degrees, check the remaining vertices of the triangle formed by S and v. If none of the vertices are obtuse, then we are finished, and the MEC is determined by the vertices of S and the vertex v.
3. If one of the other angles of the triangle formed by S and v is obtuse, then we set S to be the side opposite the obtuse angle and we repeat the main loop of the algorithm (step 1 above).

---

by Jacob Eliosoff and Richard Unger, Oct 1998

Analysis

This algorithm has an initialization time that is linear in the number of points in the set, assuming the points are given in sorted order. The main loop of the algorithm requires linear time in terms of the number of convex hull points, and this main loop could be repeated as often as once for each diagonal of the convex hull. The number of diagonals is proportional to the square of the number of convex hull points.
For that reason the total (worst case) runtime of the algorithm is proportional to the number of points in the set, **plus** the cube of the number of convex hull points.
In practice, however, the running time depends on the side initially chosen to start the algorithm, and the algorithm can be expected to perform quite well in *normal* situations.

It remains to prove that the algorithm will converge towards an answer, rather than looping forever. The proof follows from the fact that, at each iteration of the algorithm, we are reducing the radius of the circle being considered, while ensuring that all the points of the set are still within that circle. Thus the circle will converge towards the MEC.

Why did we not use the linear time algorithm? Look at it, it is detailed below. This algorithm requires the implementation of linear time median finding algorithms, and is generally far more complicated than the approach we took. In our approach, we greatly reduce the data we work on by first finding the convex hull, and then use a polynomial time solution on the reduced data set. For the specific implementation in an applet, we did not expect our data volume to be large enough to warrant the overhead involved in the linear time approach. When all is said and done n^3 for 10 points or so is going to be faster than 50n for 100 points.
Furthermore, what we are more interested in is the performance of the algorithm in terms of incremental changes to the data. We would like a fast way to add one point to the data, and patch up the MEC without having to recompute everything. To do this, we run the main loop of the algorithm with S initialized to be the last side considered when the MEC was last computed. For the addition of a single point, we expect this to have a significantly better expected running time than starting with an arbitrary side.

Linear Time Solution

The State of the Art

The Minimal Enclosing Circle problem has a long history. The simplest algorithm considers every circle defined by two or three of the n points, and finds the smallest of these which contains every point. There are $O(n^3)$ such circles, and each takes O(n) time to check, for a total running time of $O(n^4)$. Improvements on this date back as far as 1860. Elzinga and Hearn gave an $O(n^2)$ algorithm in 1972, and the first O(n logn) algorithms were discovered by Shamos and Hoey (1975), Preparata (1977), and Shamos (1978).

---

by Jacob Eliosoff and Richard Unger, Oct 1998

Finally, and to everyone's surprise, Nimrod Megiddo showed in 1983 that his ingenious prune-and-search techniques for linear programming could be adapted to find the minimal enclosing circle in **O(n) time**. Furthermore, the linear algorithm requires no more than high school mathematics to be understood and proved correct, and (as refined by Dyer in 1984) makes no general-position assumptions about the input data. This landmark result is among the most beautiful in the field of computational geometry.

Prune-and-Search

The essence of Megiddo's algorithm is **prune-and-search**. In a prune-and-search algorithm, a linear amount of work is done at each step to reduce the input size by a constant fraction f. If this can be achieved, then the total amount of work done will reduce to $O(n)*(1 + (1-f) + (1-f)^2 + ...)$. In this formula, the infinite series is geometric and sums to a constant value, and so the total running time is O(n).

For example, suppose by inspecting our set of n input elements we can discard 1/4 of them as irrelevant to the solution. By repeatedly applying this inspection to the remaining elements, we can reduce the input to a size which is trivial to solve, say n<=3. The total time taken to achieve this reduction will be proportional to (n + 3n/4 + 9n/16 + ...). It is easy to show that this series approaches, and never exceeds, a limit of 4n. So the total running time is proportional to n, as required.

The idea of using the geometric series to reduce an algorithm to linear time predates Megiddo's work; in particular, it had previously been used to develop O(n) median-finding algorithms. However, he was the first to apply it to a number of fundamental problems in computational geometry.

Megiddo's Linear-Time Algorithm

To find the minimal enclosing circle (MEC) of a set of points, Megiddo's algorithm discards at least n/16 points at each (linear-time) iteration. That is, given a set S of n points, the algorithm identifies n/16 points which can be removed from S without affecting the MEC of S. This procedure can be repeatedly applied until some trivial base case is reached (such as n=3), with the total running time proportional to (n + 15n/16 + 225n/256 + ...) = 16n.

In order to find n/16 points to discard, a great deal of cleverness is required. The algorithm makes heavy use of two subroutines:

`median(S, >)`
: takes a set S of elements and a metric for comparing them pairwise, and returns the median of the elements.

`MEC-center(S, L)`
: takes a set S of points and a line L, and determines which side of L the center of the MEC of S lies on.

---

by Jacob Eliosoff and Richard Unger, Oct 1998

As mentioned above, `median()` predates Megiddo's work, whereas the algorithm described here as `MEC-center()` was presented as part of his 1983 paper. To explore these procedures in detail would go beyond the scope of this outline, but each uses prune-and-search to run in linear time. The algorithm used by `MEC-center()` amounts to a simplified version of the algorithm as a whole.

Given these primitives, the algorithm for discarding n/16 input points runs as follows:

1. Arbitrarily pair up the n points in S to get n/2 pairs.
2. Construct a bisecting line for each pair of points, to get n/2 bisectors in total.
3. Call `median()` to find the bisector with median slope. Call this slope m_{mid}.
4. Pair up each bisector of slope >= m_{mid} with another of slope < m_{mid}, to get n/4 intersection points. Call the set of these points I.
5. Call `median()` to find the point in I with median y-value. Call this y-value y_{mid}.
6. Call `MEC-center()` to find which side of the line $y=y_{mid}$ the MEC-center C lies on. (Without loss of generality, suppose it lies above.)
7. Let I' be the subset of points of I whose y-values are less than y_{mid}. (I' contains n/8 points.)
8. Find a line L with slope m_{mid} such that half the points in I' lie to L's left, half to its right.
9. Call `MEC-center()` on L. Without loss of generality, suppose C lies on L's right.
10. Let I" be the subset of I' whose points lie to the left of L. (I" contains n/16 points.)

We can now discard one point in S for each of the n/16 intersection points in I". The reasoning runs as follows. After our two calls to `MEC-center()`, we have found that the MEC-center C must lie above y_{mid} and to the right of L, whereas any point in I" is below y_{mid} and to the left of L.

Each point in I" is at the meeting point of two bisector lines. One of these bisectors must have slope >= m_{mid}, and therefore must never pass through the quadrant where we know C to lie. Call this bisector B. Now, we know which side of B C lies on, so of the two points whose bisector is B, let P_C be the one which lies on the same side as C, and let the other be P_X.

It is easy to show that P_C must be nearer to C than P_X. It follows that P_C cannot lie on the minimal enclosing circle, and thus we can safely discard a point P_C for each of the n/16 intersection points in I".

We have not discussed here how this algorithm can be made to deal with degenerate input (parallel bisectors, colinear points, etc), but it turns out that we get the same performance guarantees for such cases - in fact, for degenerate input the algorithm is able to discard more than n/16 points. In short, no matter what the input, we are guaranteed to prune at least n/16 points at each iteration. So, by the fundamental argument from the geometric series, Megiddo's algorithm computes the minimal enclosing circle in linear time.

---

by Jacob Eliosoff and Richard Unger, Oct 1998