2D curve and network reconstruction

A. van den Boogaart W. Brouwer C. Mens M. Muijsers R. Wu

May 16, 2014

Abstract

We present three algorithms that connect all nodes in an unorganised set in the 2D plane in an aestetically pleasing manner. The first algorithm will reconstruct a set of nodes into a single curve, such that no lines intersect. The second algorithm will create multiple curves from the set of nodes, also in such a way that no lines intersect. The third and final algorithm will attempt to create a road network from the nodes, and can add additional nodes when intersections occur.

1 Introduction

Maximum size: 2 pages.

The problem that we attempt to solve is that of reconstructing a set of points back into curves in the 2D plane. This problem is split into three different parts. The first two require the reconstruction of a set of points into curves, in the first case there is only one curve, whereas in the second case there are mutiple. The third part of the problem requires reconstructing to a road network.
This means that in the second case it is also required to distinguish multiple different curves from each other.
It is important to note that the points are not given in order, since the goal is for the algorithm to determine the order, after which we can connect these points to reconstruct the curve.
The multiple curve problem has the additional problem of finding out which points belong to which curve.
Both algorithms have some constraints; Curve segments should not intersect each other and all points in the input must be used in the curve.
The third part of the problem requires reconstruction of a road network. This algorithm could be used to create a road map after data has been collected using the GPS locations of cars.
The goal of the algorithm is to create a reasonable representation of the road network, such that a person would agree with the representation that the algorithm gives. As with the other algorithms, there are some constraints.
All points need to have at least one possible path to every other point and lines cannot intersect one another, however, unlike with the other problems, points can be added to alleviate intersecting lines.
All nodes are points in a 2D plane, given by floating points ranging from 0 to 1. The nodes

are not determined randomly, but are determined beforehand. In practice, these datasets can be created by laser scanners, or in the case of the network, by following roads and giving the location at some interval.

There are multiple known solutions to these problems. One of these is 'The Crust Algorithm'[1], which first creates a Volonoi diagram, then uses Delaunay triangulation between the Volonoi diagram and the Volonoi vertices.
The Crust Algorithm solves the first problem that is described above, but needs additional work to determine the multiple curve problem.
Delaunay triangulation could also be used to create a relatively small set of edges for the third problem, after which a recilinear spanning graph or straight lines can be determined between points.
The Crust algorithm is however fairly complicated, and we do not think it is viable to recreate this within the given time.
Another algorithm we looked at, that is useful for the single curve reconstruction, is creating a convex hull. A convex hull will create segments between all the outermost points, such that no unconnected point is outside of the enclosed area. Like putting a rubber band around all the points. This can be very useful for the outermost edges, but after that it is very difficult to extract meaningful information.
A very useful subgraph that we looked at is the rectilinear spanning tree. This is a minimum spanning tree based on rectilinear distance, which is very much like manhattan distance, and is useful because it already has all the properties that the network reconstruction requires; It connects the points in such a way that there is a path from every node to every other node and there are no intersecting lines.
To solve the single curve reconstrution problem, we have chosen for a spider-type algorithm, which will go from point to point and then determines the next best point to go to.
The multiple curve reconstruction problem is solved using a weighted spanning tree, and the network reconstruction problem is also solved using a spanning tree, in this case the rectilinear spanning tree.
After reconstructing this tree we try to find straight lines and connect those, since a spanning tree will not create cycles.
The single curve reconstruction solution has produced good results so far, and has no problems with most inputs.
The result of the network reconstruction are not yet complete, we are often missing edges when adding those would add cycles.

2 The algorithms

Maximum size: 8 pages. Use subsections where appropriate..

Single curve reconstruction

The algorithm to reconstruct a single curve randomly starts at a point. Then it will look at all possible points and picks the one that is most likely, and continues doing this until all points are visited.
If the algorithm cannot find a next point that seems logical, it will return to the previous point and pick the next most likely point to go to.
Every cycle the algorithm determines a value for all unvisited points based on the angle from the line segment constructed in the previous cycle and the distance.

Multiple curve reconstruction

Network reconstruction

To reconstruct a network from a set of points, we start by creating a minimum spanning tree between all points using Kruskal's algorithm.
After this we look for straight lines in this new set of points, and check if continuing this line will result in intersections. If they do intersect within a certain distance, then we can connect these lines by adding an additional point to create an intersection.

Project guide The description of the algorithms should be such that a programmer can implement them without much difficulty. It is good practice to first explain the main ideas behind the algorithm at a more intuitive level, and then give a detailed description (for example using pseudo-code). Don't forget to describe which supporting data structures you use: linked lists, arrays, search trees, and so on. For standard data structures from the literature you do not need to explain how they work; a reference to the literature suffices.

> *Example: We store the set P of points in a red-black tree [2], using the x-coordinate of each point as its key.*

Try to theoretically analyze the worst-case running time of your algorithms and the amount of storage they use. Also try to say something about the quality of your algorithm: you might be able to prove that the algorithm is guaranteed to find the correct solution if the input has certain well-defined properties, or you might be able to give examples of inputs for which the algorithm will fail to give a correct solution.

References

[1] S. Albers. On randomized online scheduling. In *Proc. 34th ACM Sympos. Theory Comput.*, pages 134–143, 2002.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein. *Introduction to Algorithms* (2nd edition). MIT Press, 2001.

[3] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM* 30: 852–865 (1983).

- for journals: Authors. Title of paper. *Journal Name (italic)* volume: page numbers (year). See reference [3].
- for conference proceedings: Authors. Title of paper. In *Proc. Conference Name and number (italic)*, pages xxx–yyy, year. See reference [1]
- for books: Authors. *Book title (italic)*. publisher, year. See reference [2]