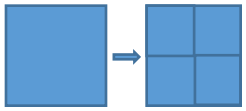
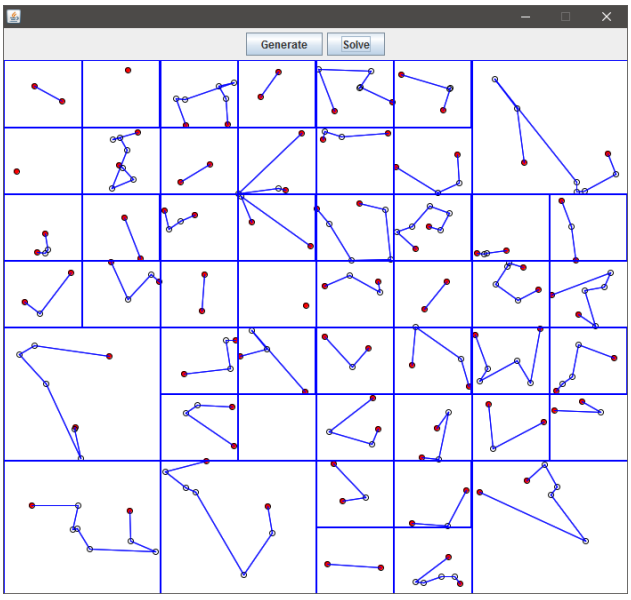
NP Solver for The Traveling Salesman Problem

I decided to create an NP Solver on the Traveling Salesman Problem. I think my approach to the problem is unique, but not incredibly accurate. The basic idea is to start with a grid of points. Then I split that grid into 4 parts, each square ½ the width and height of the original.

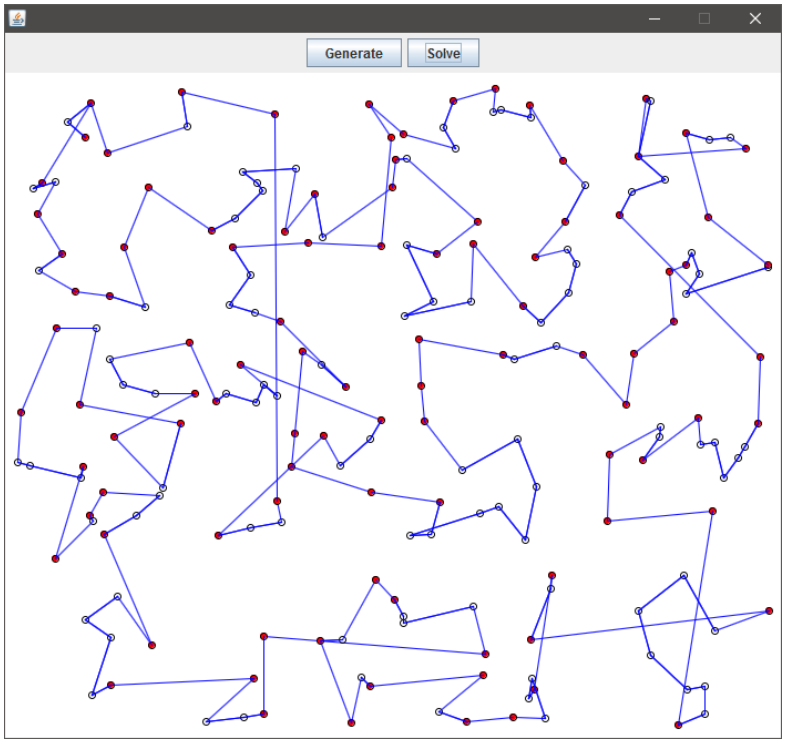


I keep doing this recursively until each grid contains 5 or less points. This allows me to solve each grid independently (map reduce), only taking $O(n!)\ for\ n = 5$. In an example with 300 points, I will be able to reduce that to $300/5 = 6$ separate grids each with 5 or less points that can each independently solved and recombined at the end. Since $300 \gg 60 * 5!$, this is an effective map reduce method. Combining of these independent grids is optimized to only combine with a grid next to itself, because there will always exist a point in that grid closer to any other free point. However, my code isn't clearly made to use the optimal fan in algorithm, it just chooses to closest point to one if its endpoints.

Here is a screenshot of my algorithm running before combining the grid shortest paths. I drew the grids to show more clearly what I mean by it. The red dots are labled "end-points" meaning those are the only points that have to be connected, and only one link is available to link them. A big difficulty I had implementing the algorithm for combining (fan in) was that I wasn't sure when a combination would produce a closed loop without including all end-points in the graph. My approach to this was to save the first grid I visit, and for every other grid visited, I make sure to connect that to a non-visited grid. So S->A, I make sure A -> B where B is a non-visited grid, and B -> C and so



on. Lastly when all grids have been visited, I connect the last Z -> S to close the loop. The only problem with this approach is that the last grid to be visited could be in the opposite corner of the starting grid, making that edge very expensive (length wise). That is definitely something to work on if this algorithm is to be improved.



This is what my algorithm can produce. As it can be seen, there is a long line going through many grids for it to connect to the first grid as mentioned. Overall, the method can find a path very quickly, but there is certainly room for improvement for decreasing the path length. Ideally, no paths should cross each other to achieve a shortest path (Sales and Chips), and there are clearly a lot of crossing paths here, but it's a start.

This assignment has been a good practice for thinking about and figuring out algorithms that can solve difficult problems. This program has much room for improvement in overall path calculations to minimize the tour. The next step would be to find a way to remove crossing edges, and possibly a less greedy algorithm for combining the grids, maybe even TSP on the grids as if they're separate points.

Reference

Sales and Chips. (2016). Retrieved December 8, 2016, from http://www.ams.org/samplings/feature-column/fcarc-tsp