GIS Cup Solution Plan

Brian Olsen, Matt Lievens

Our approach will be to start out with a simple implementation and mainly get something working first and try to develop optimizations to the working implementation later. We are going to maximize our efforts to remove the greatest amount of points with no regard to time constraints. Modeling off of the simplicity of the Visvalingam-Whyatt Algorithm, we are going to test for point in Polygon with every individual point instead of our test being an effective area.

Rough Draft Algorithm

1. Read in list of Lines and Points
2. Execute point in polygon on every single point with its surrounding points. This will not include the first and last point.
3. Append points that pass the test into a separate array.
4. Set the keep list equal to the current list.

Advantages:

- No need to calculate the polygons.
- No need to map the points to the polygons.
- Good starting spot due to it's simplicity it will be easy to improve upon.
- This code is easily parallelizable

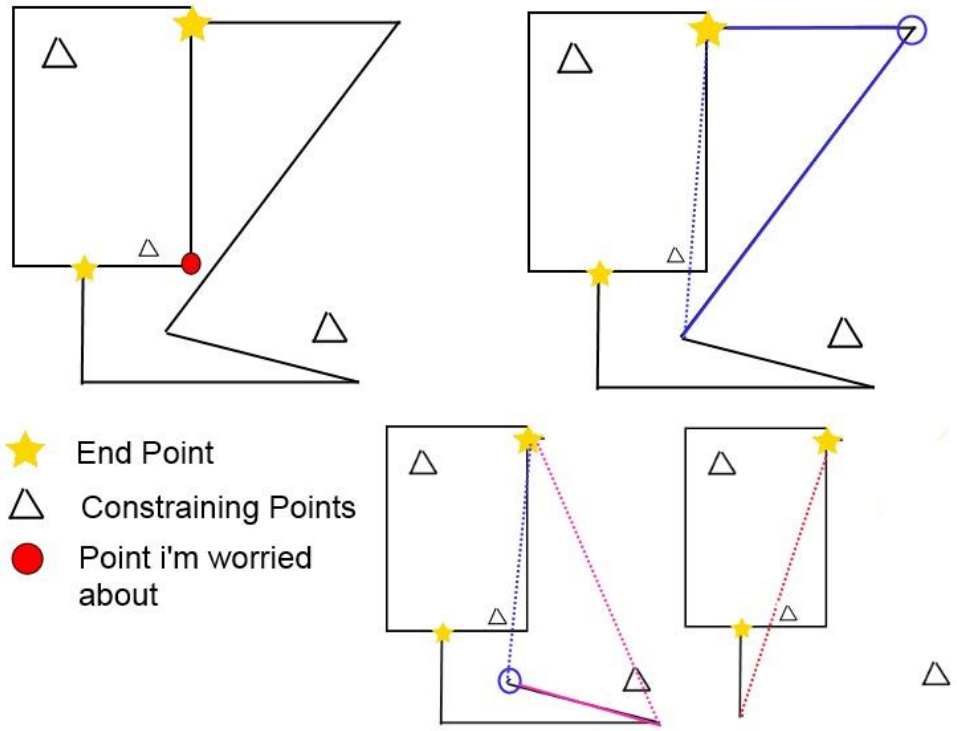
Disadvantages:

- The disadvantage is that we are very weak with time complexity.
- There is an edge case still not covered by our current algorithm that we need to address.

Plans and potential algorithms:

- Since many of the modeled algorithms display a tolerance being used we may be able to extend our simple algorithm to take multiple points into account at a time based off of some tolerance that will be based off of the number of points and lines. Possible algorithms (Perpendicular distance algorithm, or the Douglas-Peucker Algorithm)
    - The more lines and points we get the lower the tolerance will be and we may slowly degrade to our original simple algorithm but it will always remain accurate.
- Although we are trying to take the approach to avoid use of needing to make the polygons there is a possibility that doing some work before hand could give us much needed information for speed and accuracy such as building polygons but our initial intent is to avoid that.

We are aware of some shortcomings of the current algorithm.



To fix this we will have to add in verification of lines sharing the same endpoints. The simple solution is to add all the points of lines sharing the same endpoints to the list of constraining points. While this is not efficient it follows the simplicity of our algorithm and will work for now.

```
class Line:
    def __init__(self):
        #start point
        self.start = ()
        #end point
        self.end = ()
        #all points in between not including stat and end point
        self.points = []
    def getPoints(self):
        return self.points

#lines that define polygons
lines = [Line()]
#points that define constraining points
points = []

def pointInPolygon(self, polygon, point):
    return None

def makePolygon(self,points):
    return None

#get lines from the lines_out.txt
def getLines(self):
    lines = []
    #for readLine in lines_out.txt:
        #line = Line()
        #line.start = the first point
            #for each point after that but before the end
            #line.getPoints.append(point)
        #line.end  = the last point
        #lines.append(line)
    return lines

#get points from the points_out.txt
def getPoints(self):
    points = []
    return points

lines = getLines()
points = getPoints()

for l in lines:
    keepPoints = []

    for i, p in enumerate(l.getPoints()):

        before = l.getPoints()[i-1]
        after = l.getPoints()[i+1]
        polygon = makePolygon((before, p, after))
        for testP in points:
            if(pointInPolygon(polygon, testP)):
                keepPoints.append(p)
                break
```