Genetic Algorithms for Designing Powerline Layouts

2020/2021 Research Placement (Kane Alexander)

1. Research Overview

1.1 Research topic

The area of research for this project was on algorithmic designs of powerline layouts that minimize the total length of connecting powerlines. This was studied through the design and implementation of one such algorithm; a genetic algorithm coded in python from the ground up. The secondary stage of this research was to compare and analyse different implementations of obstacles penalties and how they affect the finalized powerline layouts.

1.2 Specific Research

The differing types of obstacle implementation fall into two broad categories: standard 'hard obstacle' penalties and adaptive obstacle penalties. The standard obstacle function penalises the algorithm for any powerlines that cross an obstacle, essentially adding an extreme cost to any powerlines that pass through an obstacle zone. This forces the genomes to avoid obstacles from the very start and, if the penalty is extreme enough, avoid the obstacles at all costs.

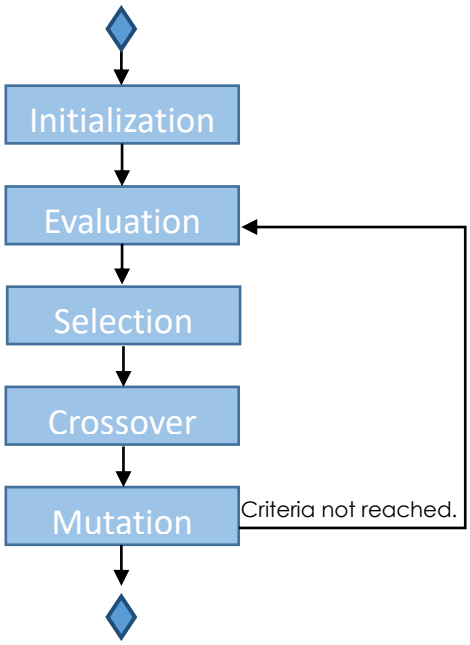
Dynamic obstacle penalties work on the same underlying concept with a different technical execution. This approach does not initially penalize the algorithm for crossing an obstacle; instead, it linearly adds an increasing penalty generation by generation. This application is a 'softer' approach as it intends to guide the algorithm away from the obstacles as time progresses.

The effectiveness of both algorithms can be compared and analysed as the final generation of both algorithms will have identical obstacle penalty functions.

2. Research Activities

2.1 Basic Design of the Genetic Algorithm

A genetic algorithm relies on the same underlying concepts as natural genetic evolution. A large population of 'genomes' all with individual traits are evaluated in respect to a certain goal. Those who perform well 'survive' until the next generation and those less advantageous are removed. The surviving genomes are then mutated/bred, and the process continues.

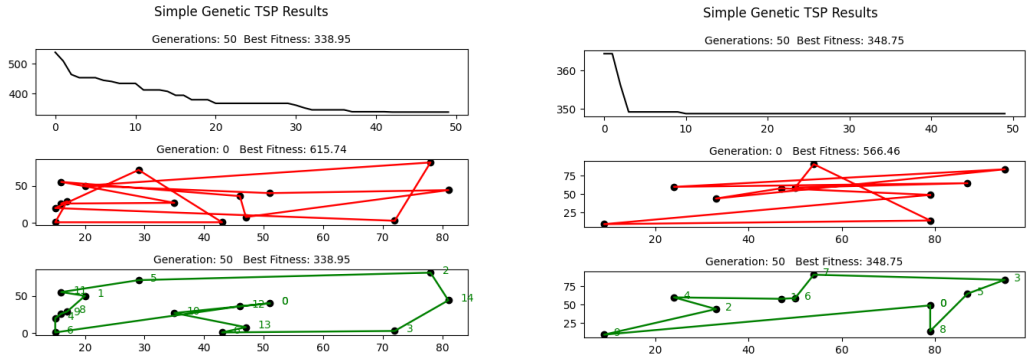


The specific genetic algorithm employed in this project connects all the houses in the best way possible (using an algorithm such as prims algorithm) then places random points and runs the evaluator again. Sometimes these random points can improve the total cost of the power lines. If they do those genomes are favoured and unfavoured genomes are deleted from the gene pool. The remaining genomes are then mutated, and the process is repeated multiple times.

2.2 Basic Implementation of a Genetic Algorithm

The goal of the first week of the project was to explore simple genetic algorithms and implement a very basic one. The traveling salesman problem (TSP) is very intuitively transposed into a genetic algorithm as the genomes of each generation simply consist of what order to go to the cities in. Coding this starting algorithm required setting up a consistent foundation that would be relied upon for the rest of the project, notably the visualization class and basic mutation/selection functions.

Found at: https://github.com/Kanealex/GCRL2000-Research-Project/tree/main/solving-TSP-genetic-algorithm



2.3 Minimum Spanning Tree and Steiner Point implementation

After the initial execution of the basic TSP genetic algorithm, the concepts were expanded to the core focus of the genetic power grid algorithm. This inevitable step required a complete advancement and remake of many functions. A working prims algorithm to find the minimum spanning tree (MST) of a set of point was employed. The algorithm implemented found the MST in O(v^2) time; whereas if a priority queue were employed it could have been be improved to O(vlog(v)) time (where v is the number of nodes). A few more possible improvements in the implementation were discussed at the supervisor meeting, namely, the algorithm was sometimes losing its 'best' genomes and the visualization axis was not standard but instead adaptive (warping the look of the MST).

This step in the development process was a relative success, despite its many technical downfalls. The algorithm did its intended purpose and showed great signs of a promising project as well as leaving many possible avenues for improvement.

2.4 Substantial Improvements

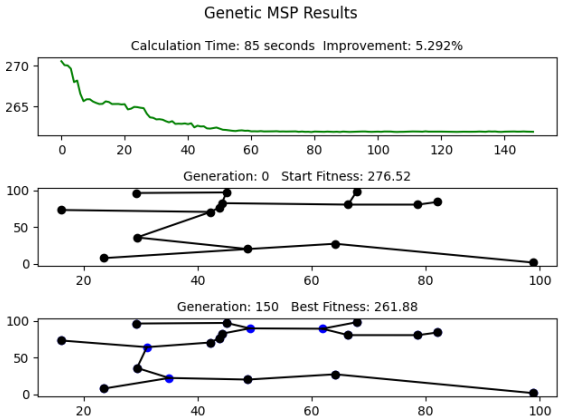
With larger sample sizes the time for the algorithm to complete a full evolution to a satisfactory standard was astronomical and clearly needed improvement. The algorithm underwent a complete technical re-evaluation, changing the basic evolution function to more robust and faster one. In simplistic terms, the old algorithm checked every genome every generation if the added points made an improvement whilst the new algorithm only checked the leading genomes but more thoroughly. This ultimately led to a very similar overall performance but an almost 100% faster algorithm; a very substantial improvement considering the multitude of trials the algorithm will have to undergo to evaluate the different obstacle functions. As pictured below, for the exact same data set the old algorithm took 85 seconds (300 genomes/150 generations) for a 5.292% improvement whilst the new algorithm took only 44 seconds for a roughly equivalent improvement (5.276%).

The new algorithm also fixed the problem of losing its 'best' solutions on some generations. As pictured below in the left graph, the old algorithms path length sometime slightly increased as the generations progressed (small spikes upward despite the overall trend decreasing). This was

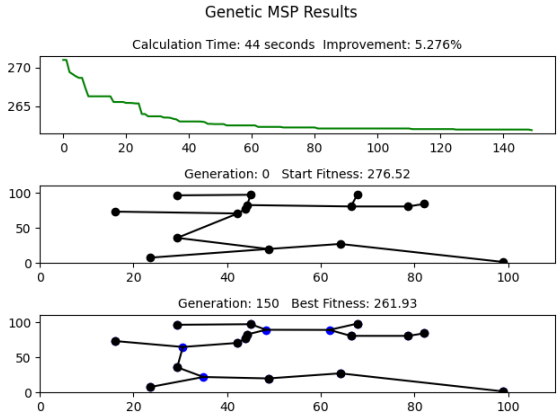
because of the random mutations sometimes negatively effecting the leading genomes. A simple counter measure of 'preserving' (not mutating or breeding) the leading genomes was employed. As seen on the left, this ensured the population could only improve/flatline as time progressed.

Improved algorithm available at: https://github.com/Kanealex/GCRL2000-Research-Project/tree/main/solving-MST-obstacles-genetic-algorithm

Old Algorithm

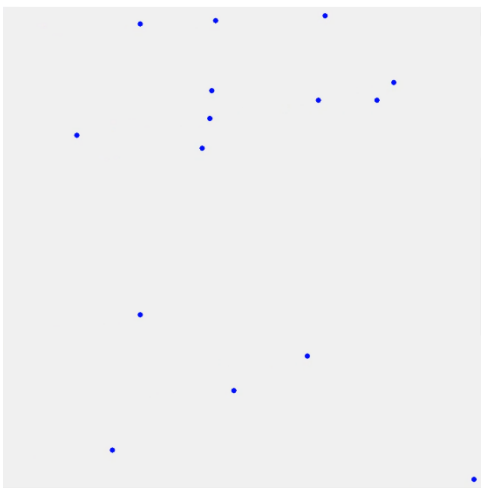


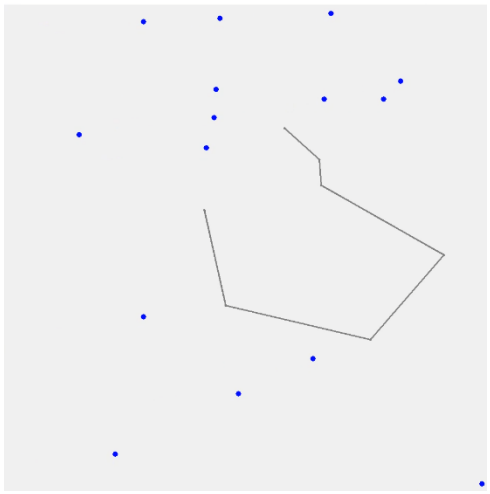
Improved Algorithm

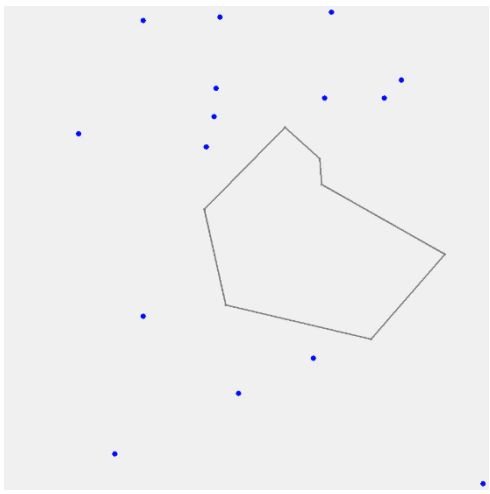


2.5 Simple Obstacles Implementation

The next stage in the project was to add obstacles that the algorithm could recognize to later develop and compare its penalty functions. Obstacles in the real world can be simplified to simple polygons, which can be simply represented to a computer as an array of co-ordinates. Although this representation in code would be relatively straight forward; it would not be intuitive to test and experiment with. Instead, I imported the tkinter library and used their functions to develop an interactive screen so the user could 'draw' their own obstacles in. The program imports the random locations of the houses and draws them on screen to ensure the user does not accidentally draw an obstacle on them. The user can add obstacle points by left clicking which will instantaneously connect and show a line leading to the last click point. When the user has added enough points to their liking they can right click, and the program completes the polygon automatically. The coordinates of the user drawn obstacle are then exported to the MST file and the genetic algorithm is run. If the user does not want to add any obstacles and instead wants to run the algorithm on just the points alone, they can just right click once on the canvas.





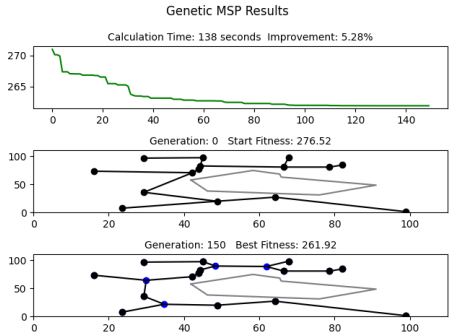


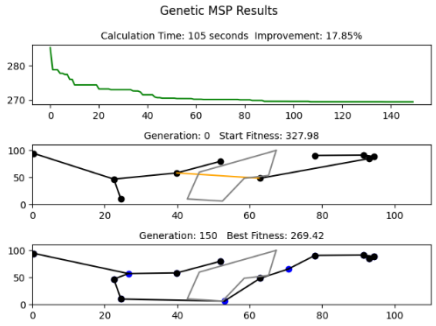
2.6 Standard Obstacle Penalty Implementation

The next stage of the project was to utilize the coordinates of the imported obstacle to employ a standard 'hard obstacle' penalty system. Whenever the algorithm worked out the path length between any 2 points in the MST a simple check was undergone; did it collide with the object. This check was designed using the fact that the polygon was simply a finite list of lines. When the check was undergone, a method returned if the line connected with the first obstacle line, if it did not then it checked the next line… ect. If there was never a collision, then it is true that the line in question did not collide with the object and its simple path length without a penalty was returned.

If the method returned that there was a collision, a penalty was associated with that line. This penalty was set to the distance between the furthest 2 points in the canvas to ensure the algorithm avoided the obstacle at all costs. This was a successful implementation of the standard penalty obstacle function.

A few quality-of-life improvements were also added to the final visualization of an algorithm run through. The obstacles were included in the visual representation, as well as changing the colour to orange of any intersecting lines.





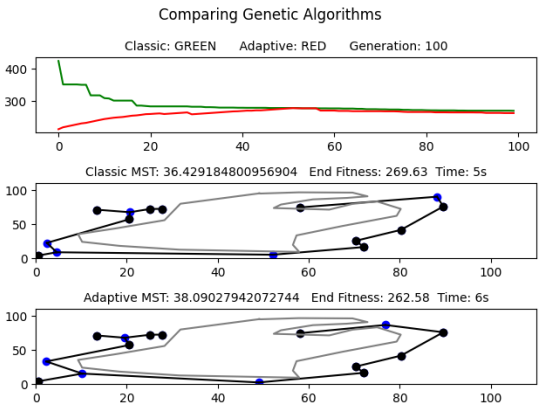
2.7 Adaptive Obstacle function implementation

Following the successful implementation of the standard obstacle penalty function, the next logical step was to design and execute an adaptive obstacle function. The design and execution of this function was very straightforward. The standard penalty function was simply modified to start at a penalty of 0 and increase to the aforementioned maximum penalty (distance between the furthest 2 points in the canvas). It did this at the same rate that the generations were increasing by; penalty = (generation/maxGeneration)*Maximum_Penalty.

Obstacle algorithm available at: https://github.com/Kanealex/GCRL2000-Research-Project/tree/main/solving-MST-obstacles-genetic-algorithm

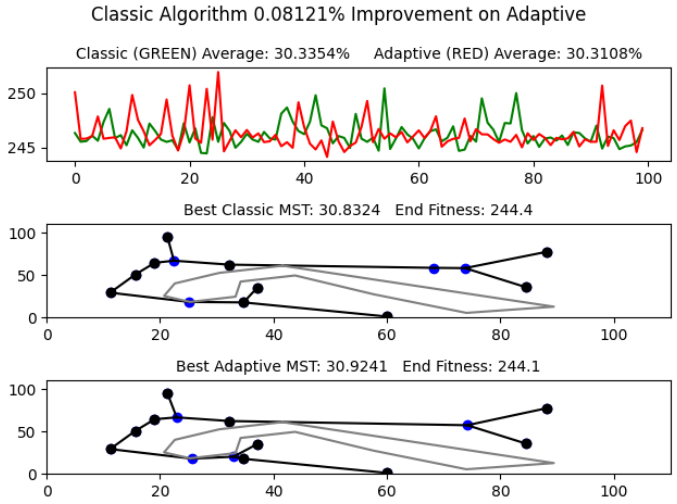
2.8 Analysis of Obstacle functions

After both obstacle functions were implemented, a simple class for testing how each performed for a set location of houses, obstacles, generations, and population size was developed. It graphed the minimum MST length over time for both functions as well as their end best paths. As shown below, the adaptive obstacle function increases over time. This is expected as initially it can pass through obstacles; but the increasing penalties result in it taking longer paths to avoid it as the generations progress. The classic function acts as intended too, starting at a longer path length but then improving over time. Interestingly both different obstacle penalty systems produce similar results, this class only tested one algorithm run through so further testing was very much needed.



2.9 Repeated Analysis of Obstacle functions

Another program was created to gather more accurate data and reduce the possibilities of randomness impacting results. This program ran the above simulation multiple times and graphed the improvement for both the classic and adaptive against each other. It also averaged the improvement for both functions to easily compare results. Again, the best paths were visualized below the data. Similarly, the results showed a very marginal difference between both different obstacle penalty systems. This completed the main goal of the research project, as it discovered that the two different types of obstacle implementations produced similar results.



Analysis algorithm available at https://github.com/Kanealex/GCRL2000-Research-Project/tree/main/genetic-algorithm-obstacle-evaluation

3. Reflection Overview

3.1 Reflection

This project experience was overwhelmingly positive and the skills I have learnt will almost certainly be applicable in my later studies. This project was my introduction to python as well as genetic algorithms as a whole. Areas of computer science that seemed extremely daunting at first. In these past few weeks, I have learnt essential python skills as well as a concrete understanding of how to implement and design genetic algorithms. My research communication skills have also been improved throughout the weekly meetings. The only real difficulties I faced were time management, the recent Perth lockdown turned my part time retail job at a pharmacy into a fulltime position for a couple of the later weeks in the project.

The future applications of this project have extreme potential. There are many aspects that I personally will continue to work on as, although the project has reached its conclusion, there remains many avenues for improvement. Importing real world maps to use real examples of houses and obstacle setups is a possible future aside, as well exploring different types of obstacle penalties than just the two explored in this project. There are these and many other future possibilities that I endeavour to work on.

Kane Alexander