**CMPT 310 Spring 2020**

**Final Project**

**John Ordoyo**

**301373568**

1. **The Genetic Algorithm Framework**

**For the Travelling Salesman Problem, I have created what I would like to call “Best Vertices/Best Edges Crossover” mutation method. It draws a similar pattern to the general crossover mutations (i.e. Using two parents and making two offspring from them) and takes influence from the Edge Assembly Crossover [1](i.e. Using edge for the crossover). The following are the steps involved in the algorithm:**

1. ***Get the best 50% edges made from both parent permutations***
2. ***Per parent, distinguish the vertices that are in the best 50% edges and those that aren’t.***

***We are going to call the latter “best vertices”***

1. ***Now that the best vertices are retrieved from the best edges, make a complete permutation by:***
   1. ***Creating a permutation from a parent in which only has “best vertices”***
   2. ***Complete that permutation by grabbing from the other parent and filling in the gaps in such a way that will not cause conflicts***

A close up of text on a black background

Description automatically generated**For the genetic algorithm, it combines the simple swap mutation from TSP.py along with the Best Vertices/Best Edges crossover to make use of efficiency. Per iteration, the Best Vertices/Best Edges algorithm gets to a minimum the fastest compared to other crossover variations implemented in this project. To get an idea of its efficiency, Figure 1 shows a comparison of Partially Mapped, Order Crossover, and Best Edges as to how they improve permutations. Swap mutation is relatively inexpensive thus both crossover compliment each other quite well.**

**Swap mutation will be in use once it hits a certain improvement threshold, else the algorithm will use the Best Vertices/Best Edges crossover mutation. It will also switch to Best Vertices/Best Edges crossover if the improvement becomes 0, since that crossover mutations would make the most impact. The greater the improvement threshold, the more the algorithm will use swap mutation.**

**Figure 1. Comparison of crossover operations to Best Vertices/Best Edges**

1. **Attempted Ideas**

**In this project a few ideas were tried. The Optimal Tree method[2] was attempted to provide a better initial population for the genetic algorithm. This involved developing a minimum spanning tree using either Kruskal’s, Primo’s, or Boruvk’s algorithms and using it to “gauge” initial permutations. For this project I have attempted this using Kruskal’s algorithm, that is adding the least weighted edges that do not create cycles. The issue lies within performance, as creating a minimum spanning tree for 1000 cities demanded a lot from the python program. Furthermore, as it is adding edges it must check each time whether a cycle is present each time. Such method is optimal for smaller TSP problems, but for larger TSP’s it is deemed rather expensive.**

**Edge Assembly Crossover[1] was also an idea of interest when working on this project. Although it has not been directly implemented in this project, EAX involves getting tours from two parents, developing cycles, and creating new tours from those cycles. The complexity is the main reason as to why it was not attempted in this project since It was quite difficult to implement.**

**The last attempted methods were Order Crossover and Partially Mapped Crossover mutations as provided in Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator[3]. These mutations were implemented out of interest and for comparison with other genetic algorithms in TSP.py. They performed better slightly better compared to the provided PMX but fail to beat mutate search. This could be due to the drastic changes it performs on the parent permutations without any guidance(Best Vertices/Edges maintains to find the best edges possible from both parents and passes it on to the offspring).**

1. **Challenges Problem**

**Finding the solution took about a few iterations. It finds an immediate solution at around 50 – 100 iterations, anything past would see very little improvements. Again, the algorithm mixes both Swap mutation and Best Vertices/Edges to balance out speed and efficiency. The best solution was found after 500 iterations**

**Best score: 497490.467362799**

**Time: 13:38:33**

**The Best Vertices/Edges can be ran in the project.py file by calling bestEdges.optimizedBestSearch(). It can be given number of iterations, population size, and the improvement threshold. It also requires the cities and their locations to run. Other crossovers can be ran by calling crossovers.[crossover name].**

**References:**

**[1] A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem**

**Link:** <https://web-a-ebscohost-com.proxy.lib.sfu.ca/ehost/detail/detail?vid=0&sid=7fbd58a3-1fa7-4534-89c3-4529706c3560%40sessionmgr4006&bdata=JnNpdGU9ZWhvc3QtbGl2ZQ%3d%3d#AN=87120578&db=aph>

**[2] Optimal Tree for Genetic Algorithms in the Traveling Salesman Problem(TSP):**

**Link:** <https://arxiv.org/abs/1204.2352>

**[3] Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator:**

**Link:** <https://www.hindawi.com/journals/cin/2017/7430125/>

**tsp.py is provided in the final project of CMPT 310 Spring 2020**