AI Project 4

Eric Dockery
Computer Engineering and Computer Science
Speed School of Engineering
University of Louisville, USA
eadock01@louisville.edu

Introduction:

For this assignment, we are asked to solve the traveling salesman problem using genetic algorithm models, and changing the mutation rate and crossover methods. This is the first time I have worked with genetic algorithms.

Approach:

For this problem we are learning how to implement a genetic algorithm. My program is split into many different functions so I will describe each individually.

My main program takes the filename that the user gives and strips the valuable date from the file. It then creates double the value of the points in Parents by calling the CreateParent function. After each parent is made the program makes the graphs for that generation calling the plotPoints function. After that the main program takes the array of parents from that generation and makes children by calling the Child function. It then displays each Child generations graph and prints the final solution to the main program.

The CreateParent function takes in the array from main and selects nodes at random to create a traveling salesman path. It then calculates the distance that it travels by calling distanceFormula function. It returns this path and distance as a value in an array to the main.

The distance formula calculates the distance of the array and returns it to the calling function. There are two calling functions to the distance formula the CreateParent and the crossover.

The plotPoints function draws the graph for each call showing the traveling salesman path and the total distance.

The Child function takes the Array of Parents and randomly selects a fitness value as a distance that is given from one Parent selected at random. It then removes all parents that distance is greater than the randomly selected distance and calls the crossover function.

The crossover function takes the reduced parents and depending on the values you change in the program during the runs it can create certain children that are different mixes of the Parents, it evaluates the distance between two cities and if whichever parent has a shorter value for a randomly selected city it will switch those city paths in the child. It then adds the rest of the shorter parent in the child making sure to avoid duplicates. The program calls the distanceFormula function to get the new children's distances. The function also has a probability chance of selecting the mutate function that you can change as needed.

The mutate function inverts any array sent to it and returns it to the crossover function.

3. Results:

3.1 Data:

The data that was used for this assignment was generated using Concorde. The format for the data was:

NAME: concorde100

TYPE: TSP

COMMENT: Generated by CCutil_writetsplib

COMMENT: Write called for by Concorde GUI

DIMENSION: 100

EDGE_WEIGHT_TYPE: EUC_2D

NODE_COORD_SECTION

1 87.951292 2.658162

2 33.466597 66.682943

3 91.778314 53.807184

4 20.526749 47.633290

5 9.006012 81.185339

6 20.032350 2.761925

7 77.181310 31.922361

8 41.059603 32.578509

9 18.692587 97.015290

10 51.658681 33.808405

11 44.563128 47.541734

12 37.806330 50.599689

13 9.961241 20.337535

14 28.186895 70.415357

15 62.129582 6.183050

16 50.376904 42.796106

17 71.285134 43.671987

18 34.156316 49.113437

19 85.201575 71.837519

20 27.466659 1.394696

21 97.985778 44.746239

22 40.730003 98.400830

23 73.799860 61.076693

24 85.076449 17.029328

25 16.052736 11.899167

26 20.160527 67.238380

27 22.730186 99.725333

28 77.196570 88.503677

29 18.494217 31.971191

30 72.743919 16.071047

31 4.153569 41.981262

32 79.027680 95.034639

33 14.145329 40.690329

34 66.258736 70.360424

35 22.656941 52.076785

36 82.680746 31.058687

37 88.995025 35.560167

38 87.939085 36.567278

39 82.845546 48.393200

40 5.371258 3.466903

41 80.028687 51.258889

42 8.908353 80.703146

43 69.411298 10.122990

44 10.129093 91.378521

45 61.546678 97.531053

46 61.156041 69.313639

47 39.719840 46.403394

48 38.999603 68.407239

49 43.992431 59.556871

50 26.963103 73.021638

51 28.879666 27.948851

52 58.751183 87.429426

53 85.290078 60.875271

54 40.879543 32.523576

55 67.326884 81.203650

56 19.064913 27.845088

57 14.648885 88.753929

58 4.153569 87.118137

59 10.895108 44.978179

60 23.258156 5.346843

61 68.926054 82.073428

62 11.713004 65.706351

63 83.404035 89.590136

64 11.471908 44.187750

65 41.422773 81.743828

66 91.595202 40.324107

67 31.730094 98.501541

68 56.382946 11.935789

69 43.232521 43.571276

70 56.904813 42.152165

71 93.386639 12.457656

72 71.395001 16.754662

73 77.065340 13.657033

74 70.278024 40.021973

75 76.604511 36.146123

76 31.351665 67.159032

77 23.563341 66.295358

78 20.822779 81.447798

79 52.903836 7.309183

80 5.746635 94.280831

81 40.147099 4.345836

82 13.583789 96.127201

83 62.181463 28.403577

84 4.409925 36.637471

85 72.331919 22.144230

86 71.483505 61.964782

87 30.283517 60.740989

88 35.721915 87.408063

89 77.556688 30.884732

90 49.781793 33.585620

91 99.078341 81.115146

92 77.309488 4.168828

93 61.522263 46.504105

94 63.026826 33.359783

95 69.045076 0.952177

96 59.254738 81.203650

97 27.005829 40.083010

98 24.509415 4.898221

99 54.347362 47.959838

100 59.797967 84.2158275 9.006012 81.185339

3.2 Results:

Excel Sheet – changes in population count, crossover rate, and mutation rate.

Test Cases:

	Mutation Rate	Crossover Rate	Start Value	End Value	Generations	Starting population
First Runs	0.01	0.1				200
Run1			4966.49	3704	150	
Run2			6857.46	3726.64	150	
Run3			6083.32	3703.45	282	
Run4			5114.61	3726.64	21	
Run5			4446.54	3703.45	32	

	Mutation Rate	Crossover Rate	Start Value	End Value	Generations	Starting population
First Runs	0.01	0.25				200
Run1			4754.017	3703.45	675	
Run2			5867.78	3728.41	257	
Run3			4417.17	3728.4	520	
Run4			4689.33	3726.64	149	
Run5			5867.78	3703.45	100	

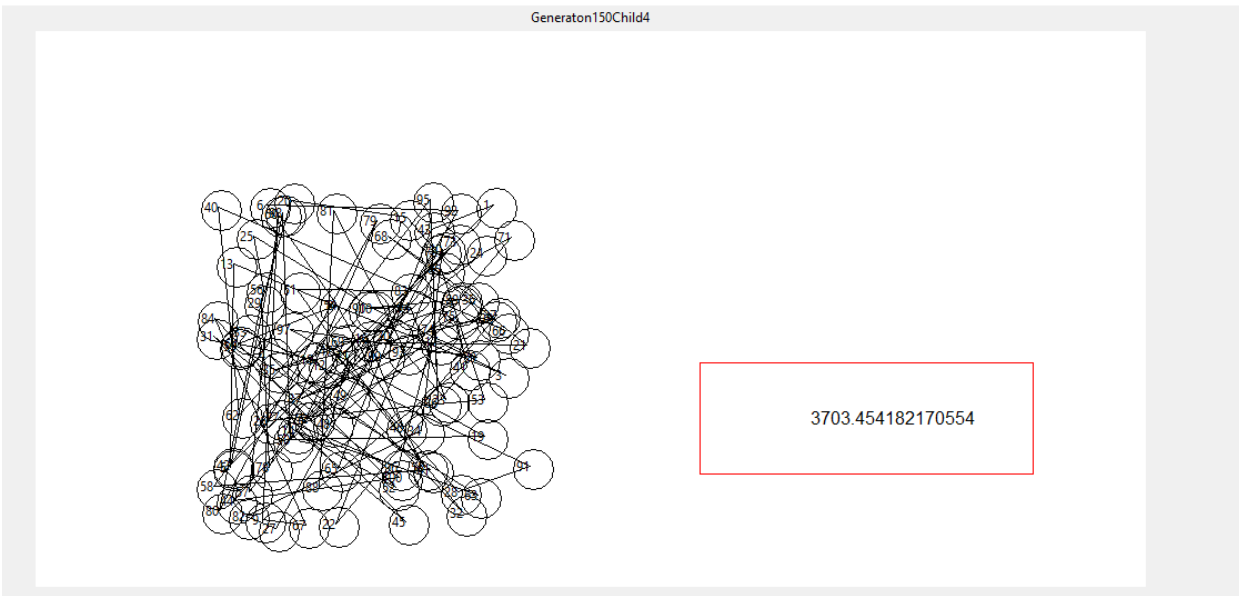
	Mutation Rate	Crossover Rate	Start Value	End Value	Generations	Starting population
First Runs	0.01	0.5				200
Run1			4965.18	3703.45	103	
Run2			4213.53	3726.64	184	
Run3			4308.18	3703.45	163	

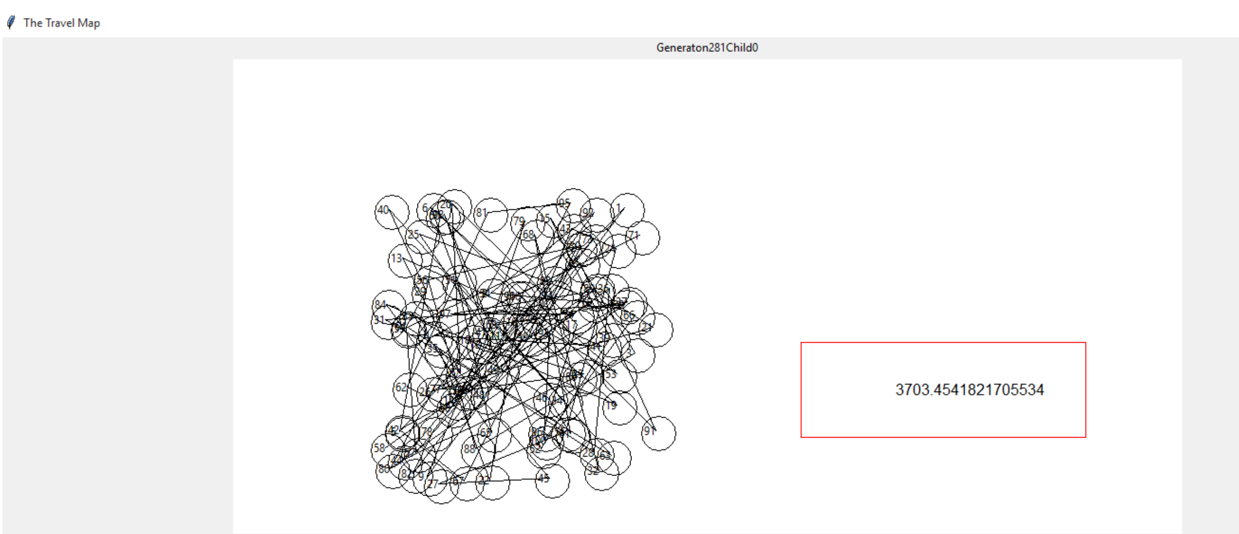
	Mutation Rate	Crossover Rate	Start Value	End Value	Generations	Starting population
First Runs	0.1	0.5				200
Run1			4937.45	3703.45	270	
Run2			5107.64	3726.64	250	
Run3			5837.43	3703.45	11	

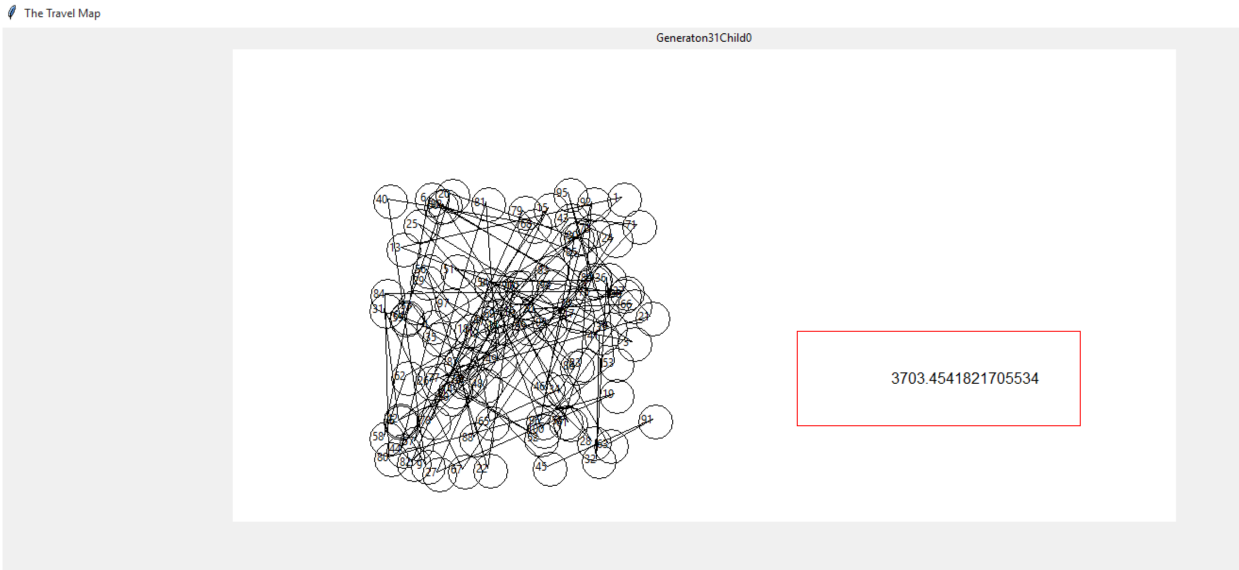
	Mutation Rate	Crossover Rate	Start Value	End Value	Generations	Starting population
First Runs	0.2	0.9				200
Run1			5028.08	3703.45	132	
Run2			5783.18	3726.64	58	
Run3			3821.68	3726.64	51	

	Mutation Rate	Crossover Rate	Start Value	End Value	Generations	Starting population
First Runs	0.2	0.9				500
Run1			3703.45	3703.45	194	
Run2			5001.95	3703.45		

Screenshots of final value

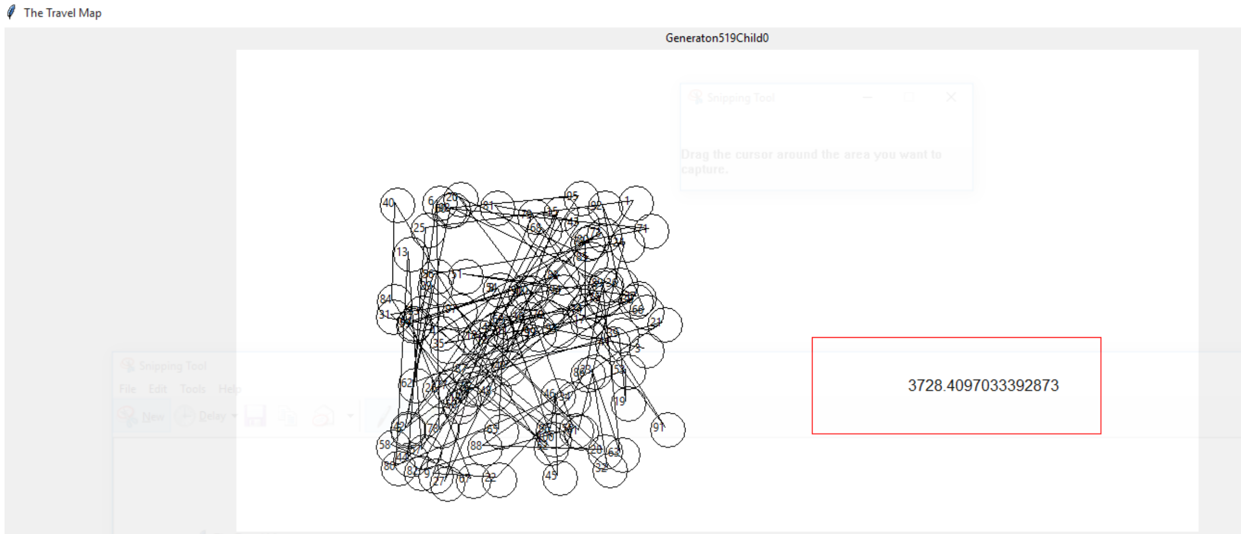


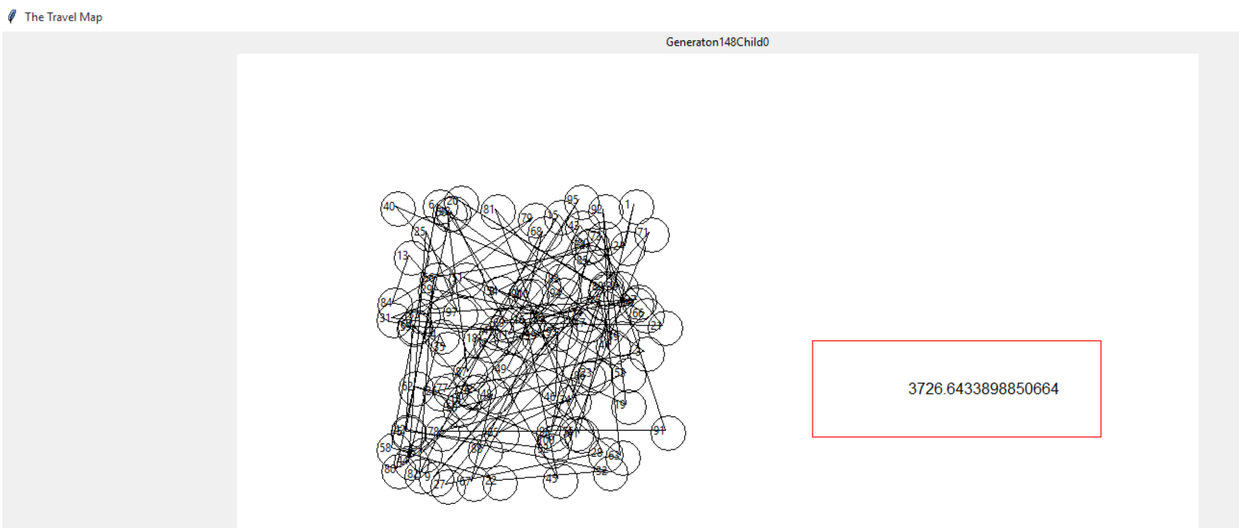



The Travel Map
Generaton31Child0
3703.4541821705534

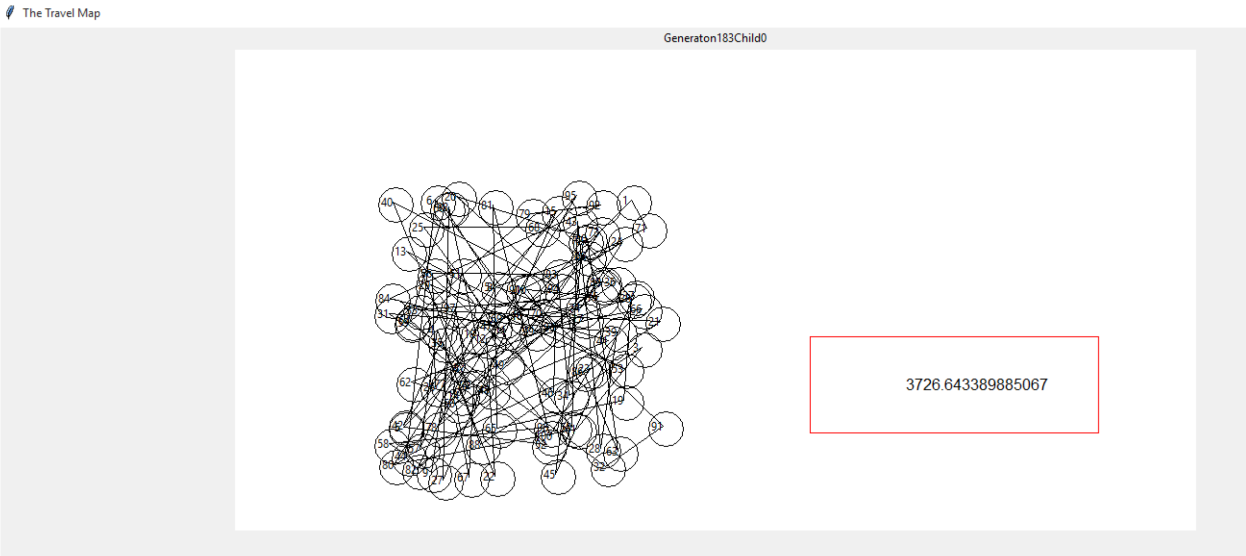

The Travel Map
Generaton674Child0
3703.4541821705548

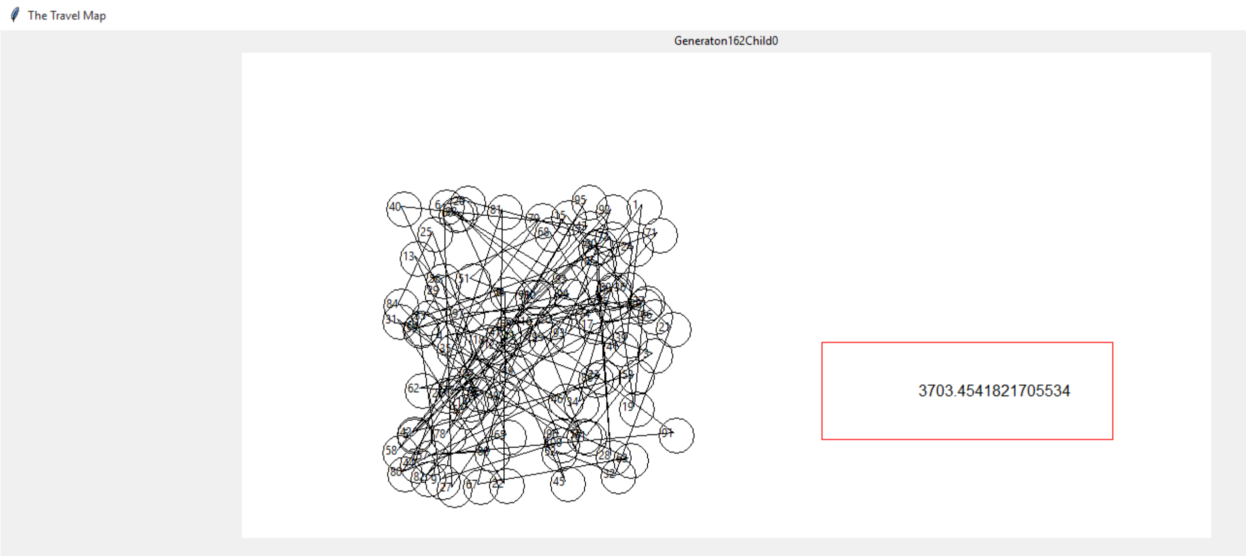

The Travel Map
Generaton256Child1
3728.4097033392886

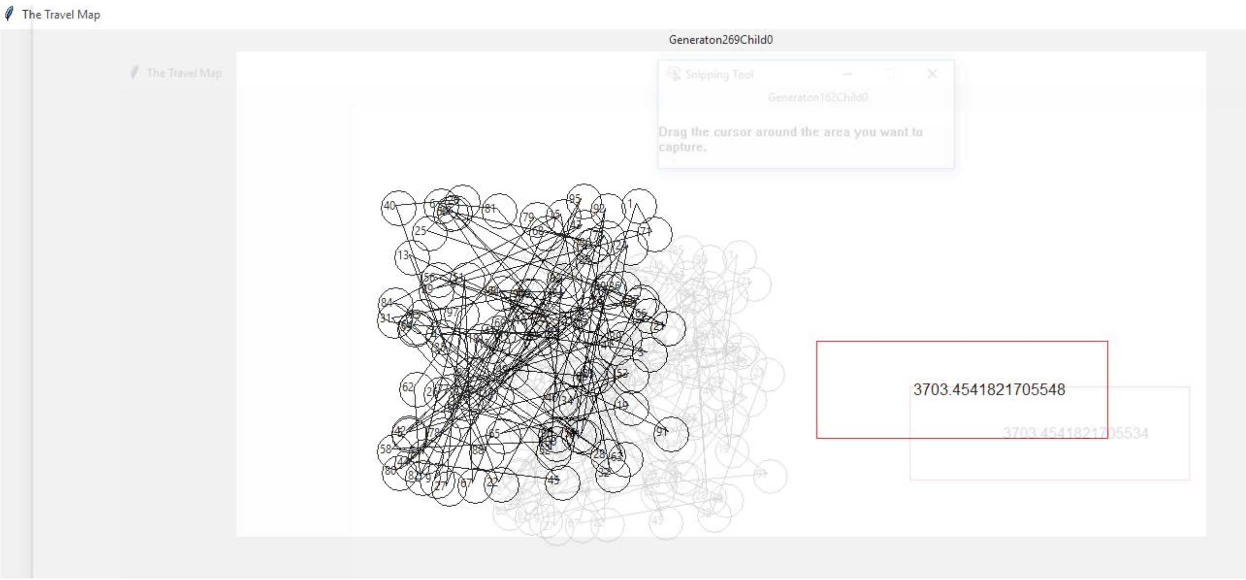

The Travel Map
Generaton519Child0
3728.4097033392873

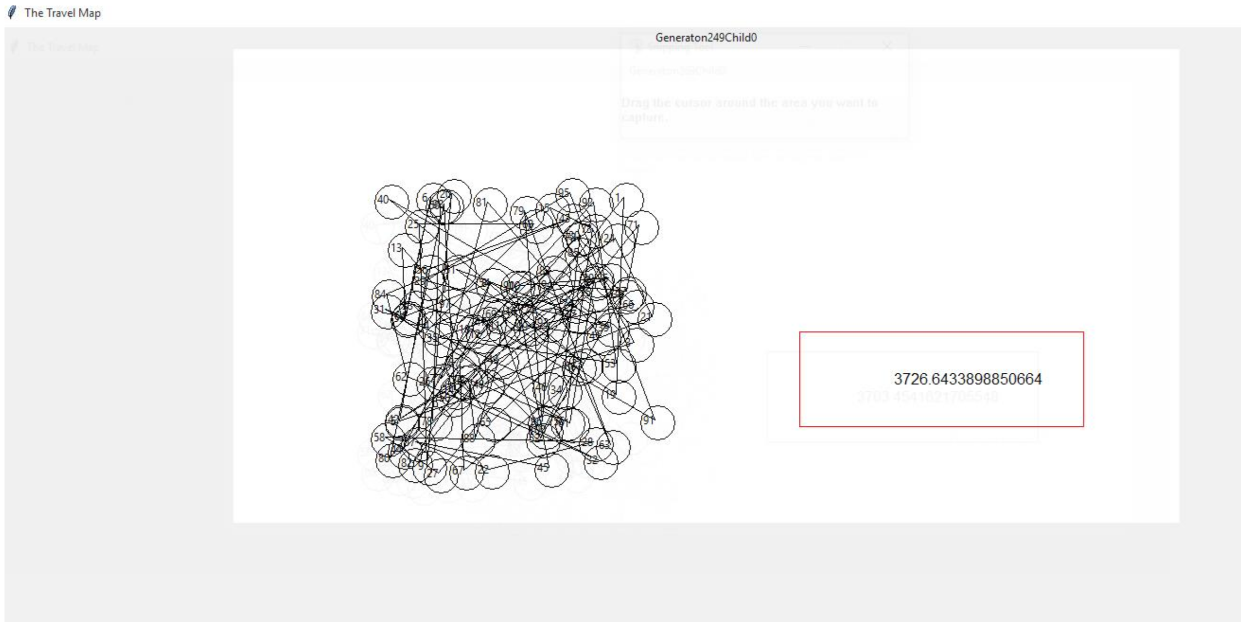

The Travel Map
Generaton148Child0
3726.6433898850664

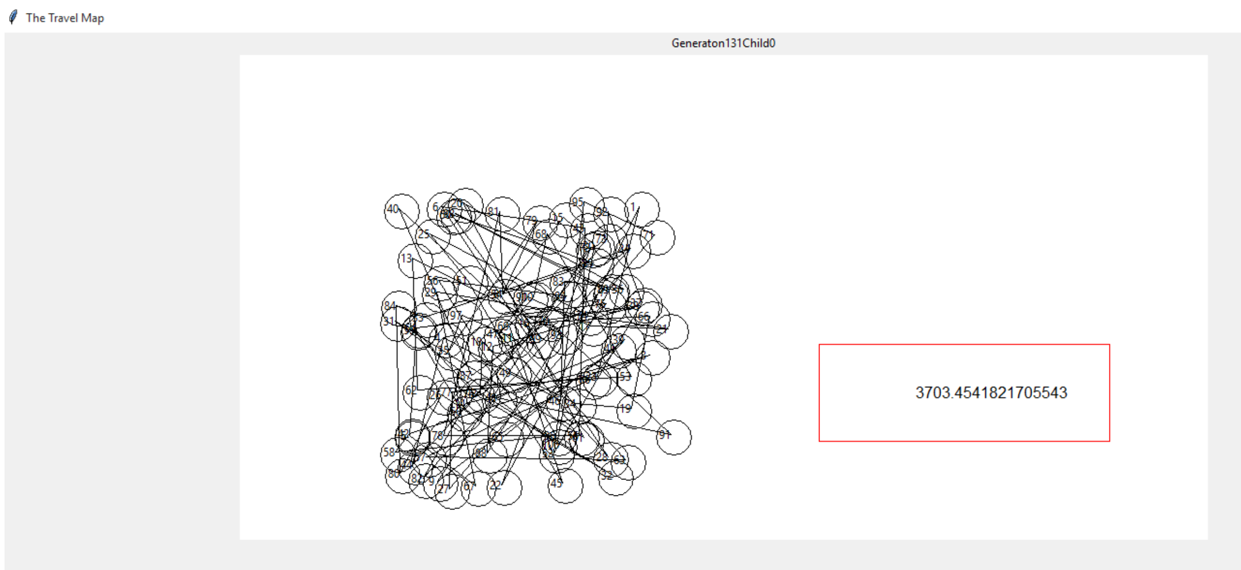

The Travel Map
Generaton99Child0
3703.454182170554

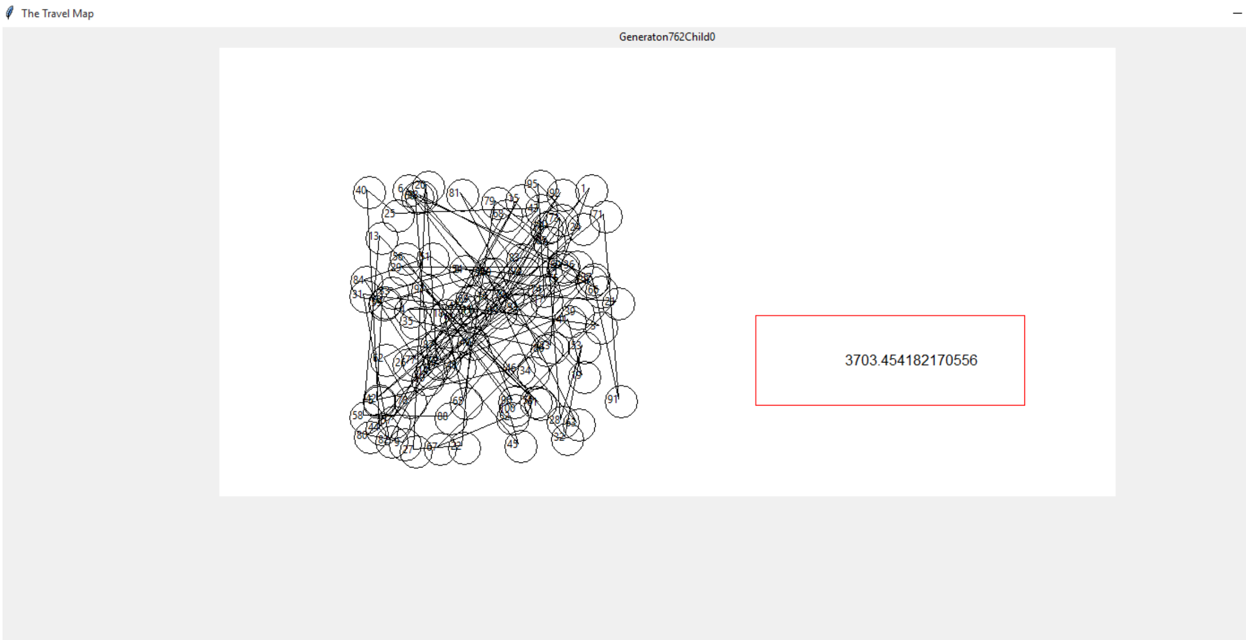

The Travel Map
Generaton183Child0
3726.643389885067


The Travel Map
Generaton162Child0
3703.4541821705534


The Travel Map
Generaton269Child0
3703.4541821705548


The Travel Map
Generaton249Child0
3726.6433898850664

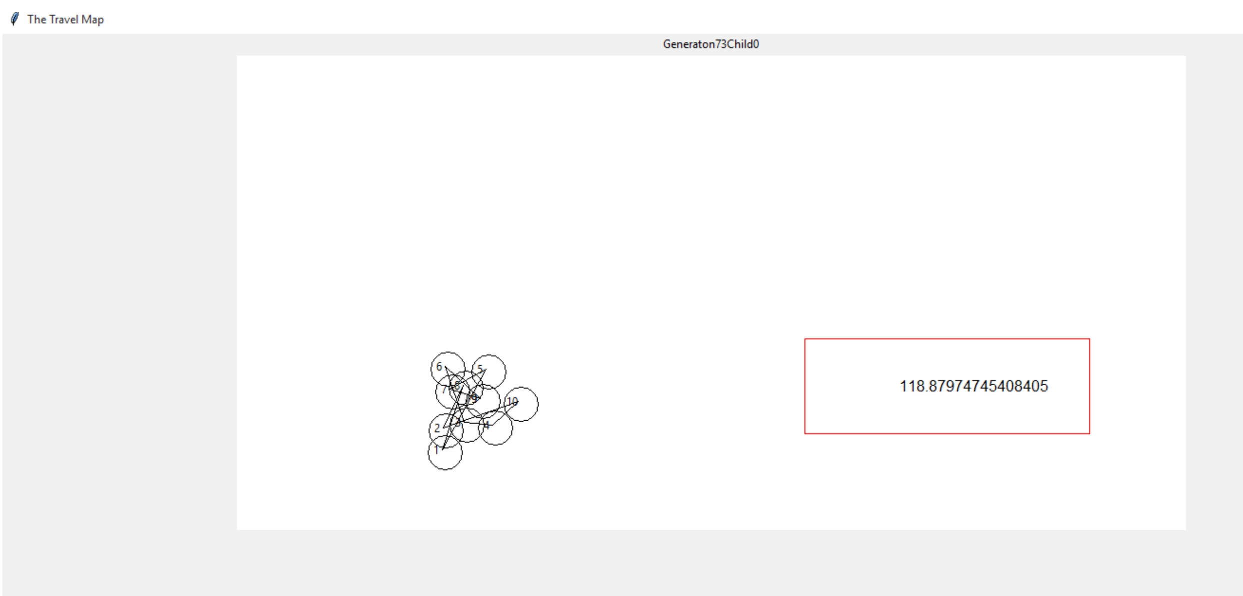

The Travel Map
Generaton131Child0
3703.4541821705543



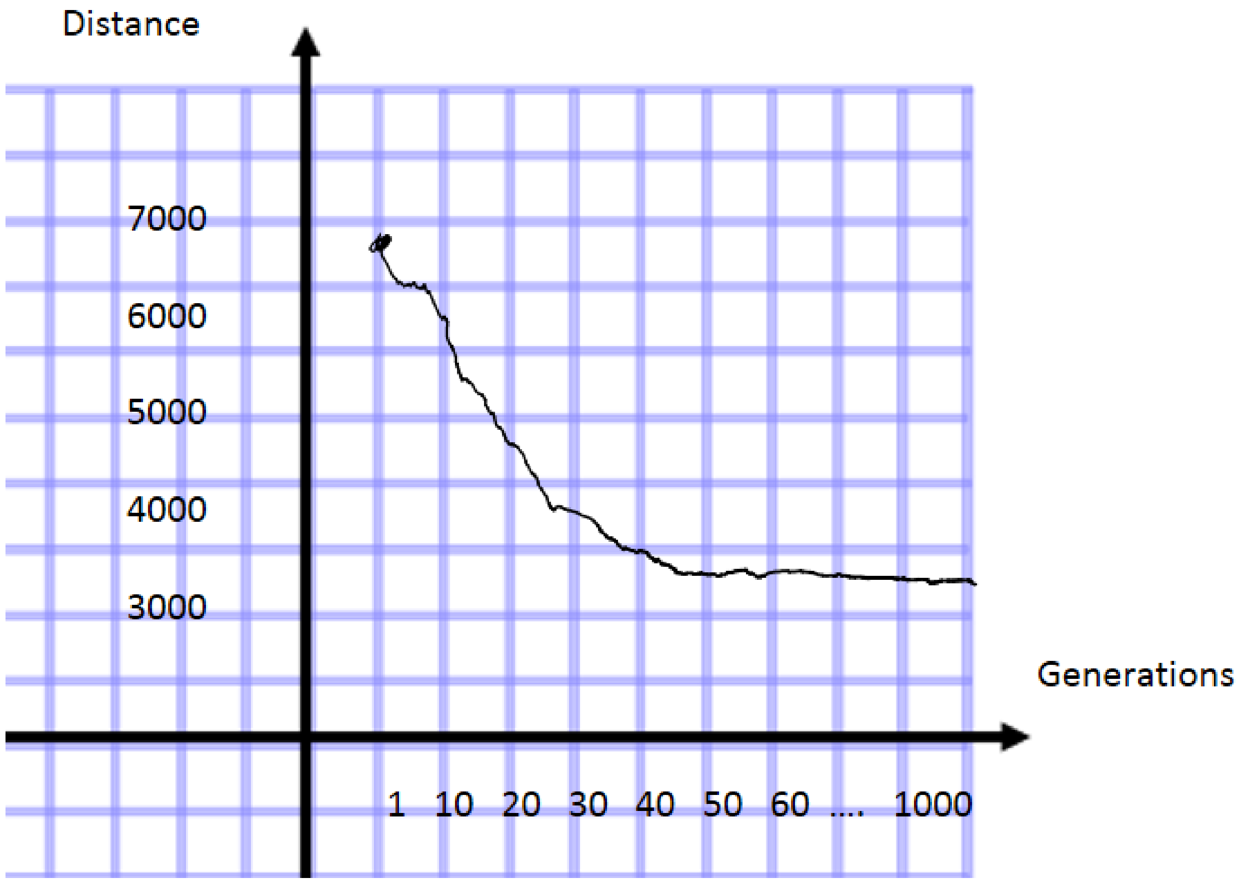
Results for 10 cities

[[[['9', '36.029412', '70.886076'], ['7', '25.245098', '67.721519'], ['5', '38.071895', '60.759494'], ['1', '22.549020', '89.029536'], ['8', '30.065359', '66.244726'], ['2', '23.039216', '81.434599'], ['10', '49.264706', '71.940928'], ['4', '40.277778', '80.379747'], ['3', '30.392157', '79.324895'], ['6', '23.774510', '59.704641'], ['9', '36.029412', '70.886076']], 118.87974745408405]]

74



Graph:



4. Discussion:

In conclusion to this genetic algorithm, I believe that somewhere in my code the algorithm for a PMX greedy genetic algorithm is incorrect. Although, the results in comparison to my Project3 on 10 nodes the genetic algorithm improved its distance by 10 spaces, on the 100 city test the value for Project3 is 1071, whereas the lowest value from the genetic algorithm that I could produce is in the 3000's. I do not believe that my PMX solution is correct.

The biggest problem that I am having implementing this project is the Darwinism of the generations. It is currently randomly selecting a fitness value that will kill the children where as it needs to be an evolutionary adaptation that will reduce with implementation, allowing only the fittest children to live through the crossovers.

If I was to change anything on this project it would be to create a better crossover function. My current crossover function is not good enough for an optimal solution. Maybe a different implementation of crossover would generate better results.

I learned quite a bit about Genetic Programming by creating this program. I didn't quite understand the evolutionary track of a GA prior to starting this project, but after reading a few different documents about the process I feel I have a good grasp on the theoretical side of a GA. I wish that I was able to implement a better solution for this problem, but there is a lot left to learn and explore. Even though this program does not generate an optimal solution I still plan on tweaking and getting help on the project so that I can understand what I have done wrong and improve my GA.

I think that the GA as a problem solving technique is quite interesting. I noticed the improvements of my program throughout the generations, even if it was trivial. I think that with a better implementation than mine the only issue is the time that the program takes when creating the initial parents, due to the limitless possibilities.

5. References:

http://www.ceng.metu.edu.tr/~ucoluk/research/publications/tspnew.pdf