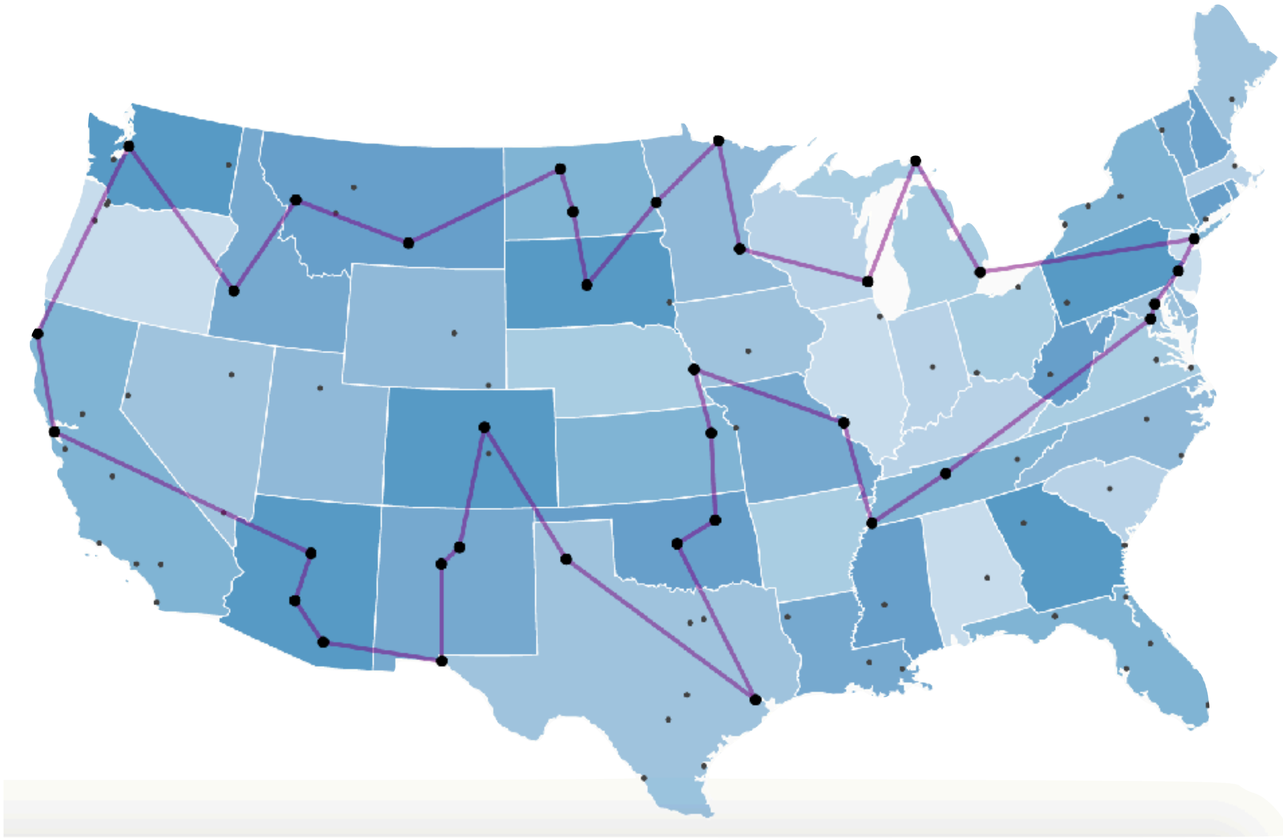
TRAVELLING SALESMAN PROBLEM

Team501: Ang Li , Xiaohan Zhao

PROBLEM DESCRIPTION

The **travelling salesman problem (TSP)** asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

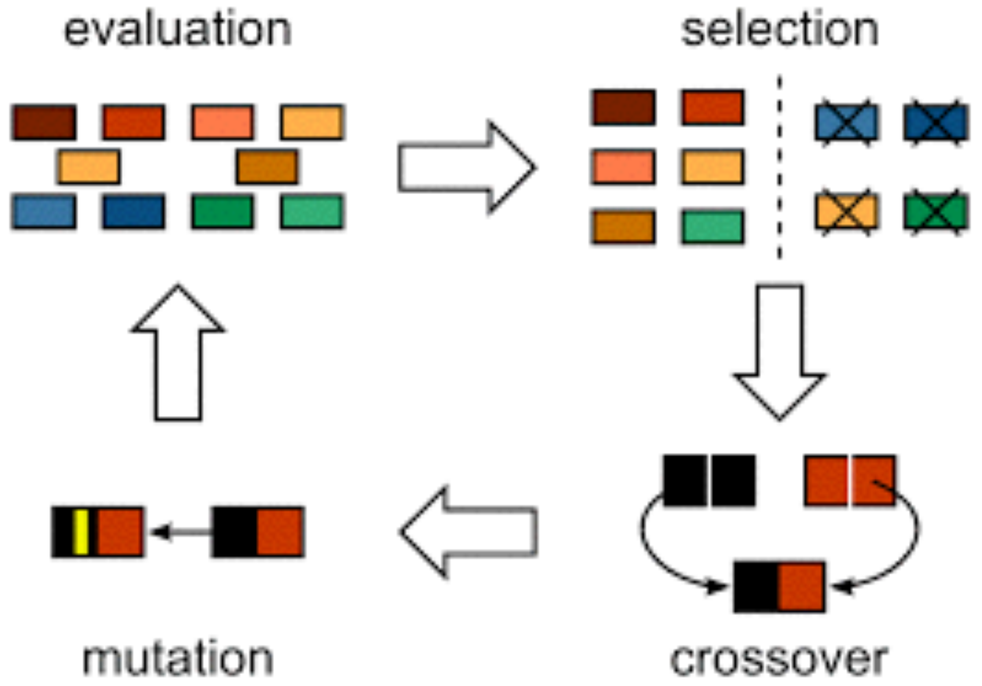


Source: Wikipedia

WHAT WE DID

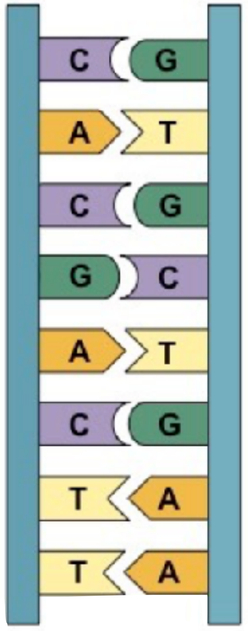
- Created a **genetic algorithm** to solve the travelling salesman problem with clear comments
- Used **parallel computation mechanism** to divide population up into sub-populations and merge the next generations for each colony in parallel
- Created **unit tests** to test most of the methods to keep the project operating properly
- Created several input data files to be used in this algorithm
- Created a **user interface** using java Swing to show the progress of the evolution

GENETIC ALGORITHM



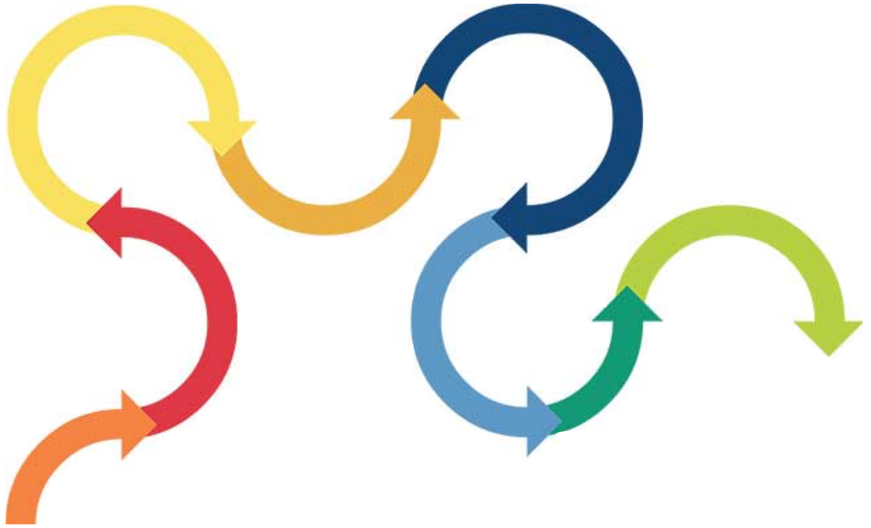
DESIGN OF TSPCHROMOSOME

- GenotypeList is used to represent the genotype of the corresponding chromosome



[AGAT, AGAA, AGAG, AGAC]

- PhenotypeList is used to represent the real path

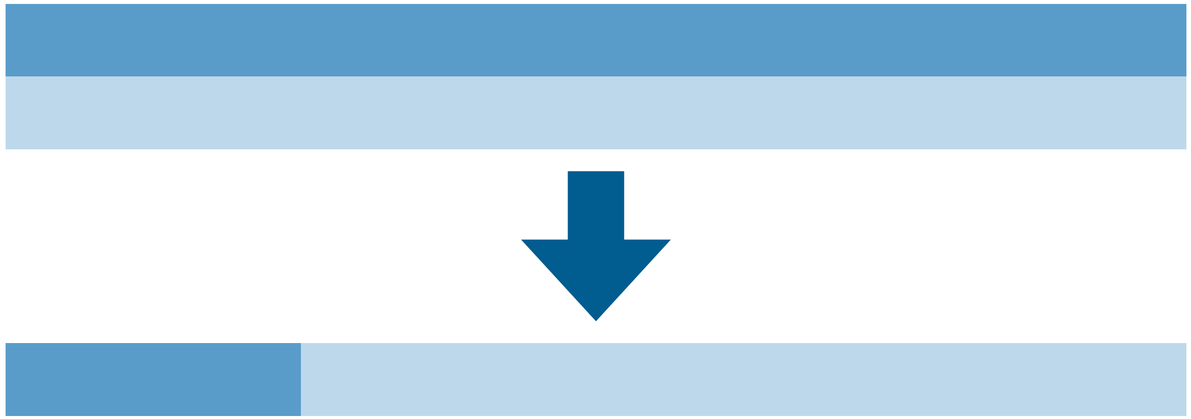


[19, 16, 17, 18]

DESIGN OF TSPCHROMOSOME

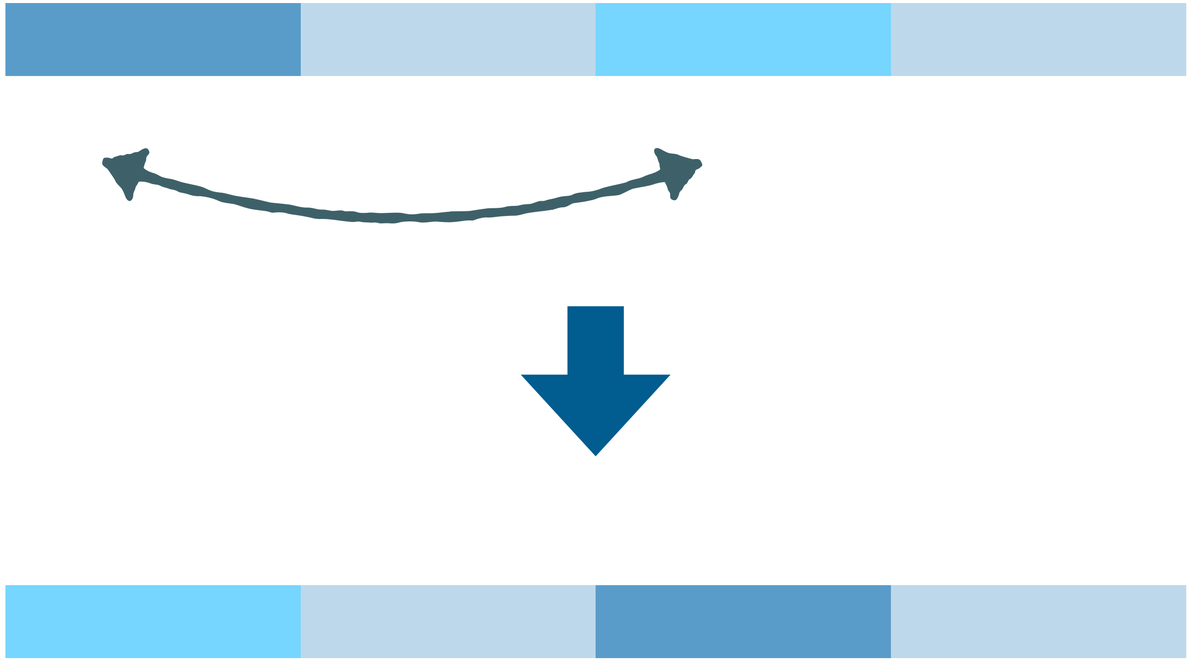
crossover:

- we generate a random number, and the new chromosome is generated by part of the father chromosome and part of the mother chromosome.
- To prevent visiting a city repeatedly, we need to ignore the used genes.



mutation

- generating two different cities randomly and swap them



DESIGN OF TSPGENETICALGORITHM

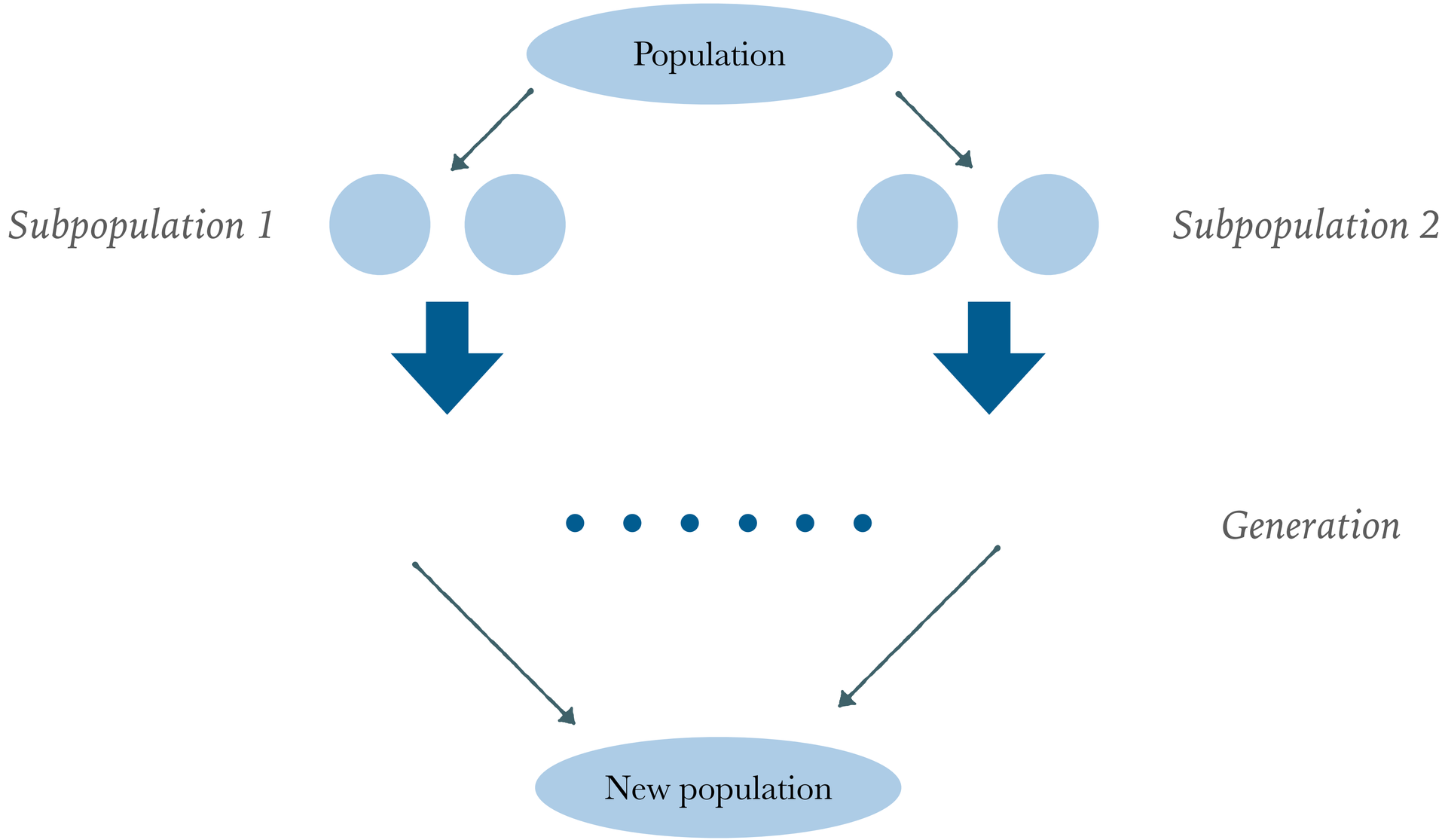
select()

- This method is used to choose good entities to generate a child generation and cull the bad entities
- The chromosomes are sorted by the weight to be chosen properly
- The best entity of each generation will keep alive without any change to keep the best genotype of old generation

evolution()

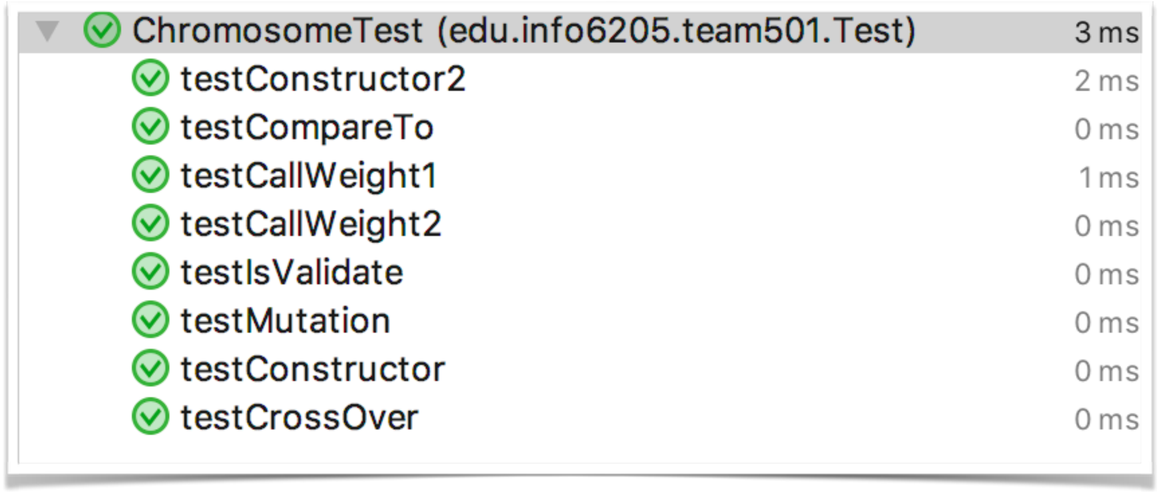
- This method is used to crossover and mutation of the new generation's genotype
- In our algorithm, the possibility of crossover is much higher than the possibility of mutation, this is a experiment conclusion

PARALLEL COMPUTATION MECHANISM

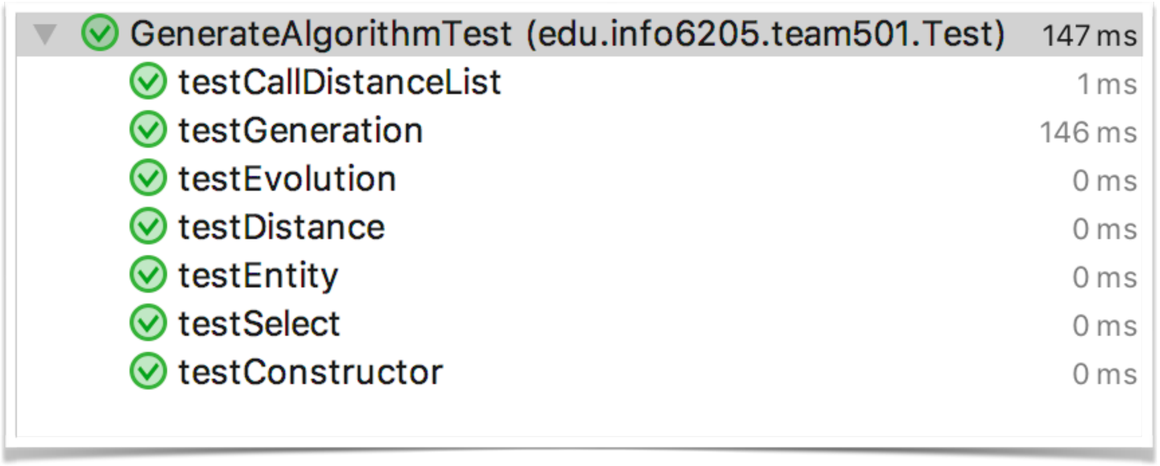


UNIT TEST

Chromosome Test



Generate Algorithm Test



INPUT

Data.txt *Input format of one city: x point, y point*

202 208	295 77
217 326	304 202
290 432	154 148
423 440	99 212
445 345	152 276
360 349	404 262
138 401	420 164
99 301	503 209
328 300	510 306
234 397	480 101

OUTPUT

The Shortest Distance of Generation: 0 is 2542.3078184269507

The Shortest Distance of Generation: 1 is 2469.4023093516944

The Shortest Distance of Generation: 2 is 2276.4421035323244

The Shortest Distance of Generation: 3 is 2254.801958883949

The Shortest Distance of Generation: 4 is 2138.8148348267373

The Shortest Distance of Generation: 5 is 2107.5368818111283

The Shortest Distance of Generation: 6 is 2075.7768899889475

The Shortest Distance of Generation: 7 is 1983.406372018409

The Shortest Distance of Generation: 8 is 1887.8682090142695

The Shortest Distance of Generation: 9 is 1857.1880627403664

The Shortest Distance of Generation: 10 is 1856.3944513950682

The Shortest Distance of Generation: 11 is 1804.6374702253072

The Shortest Distance of Generation: 12 is 1740.9972056296554

The Shortest Distance of Generation: 13 is 1740.9972056296554

The Shortest Distance of Generation: 14 is 1740.9972056296554

(……)

(……)

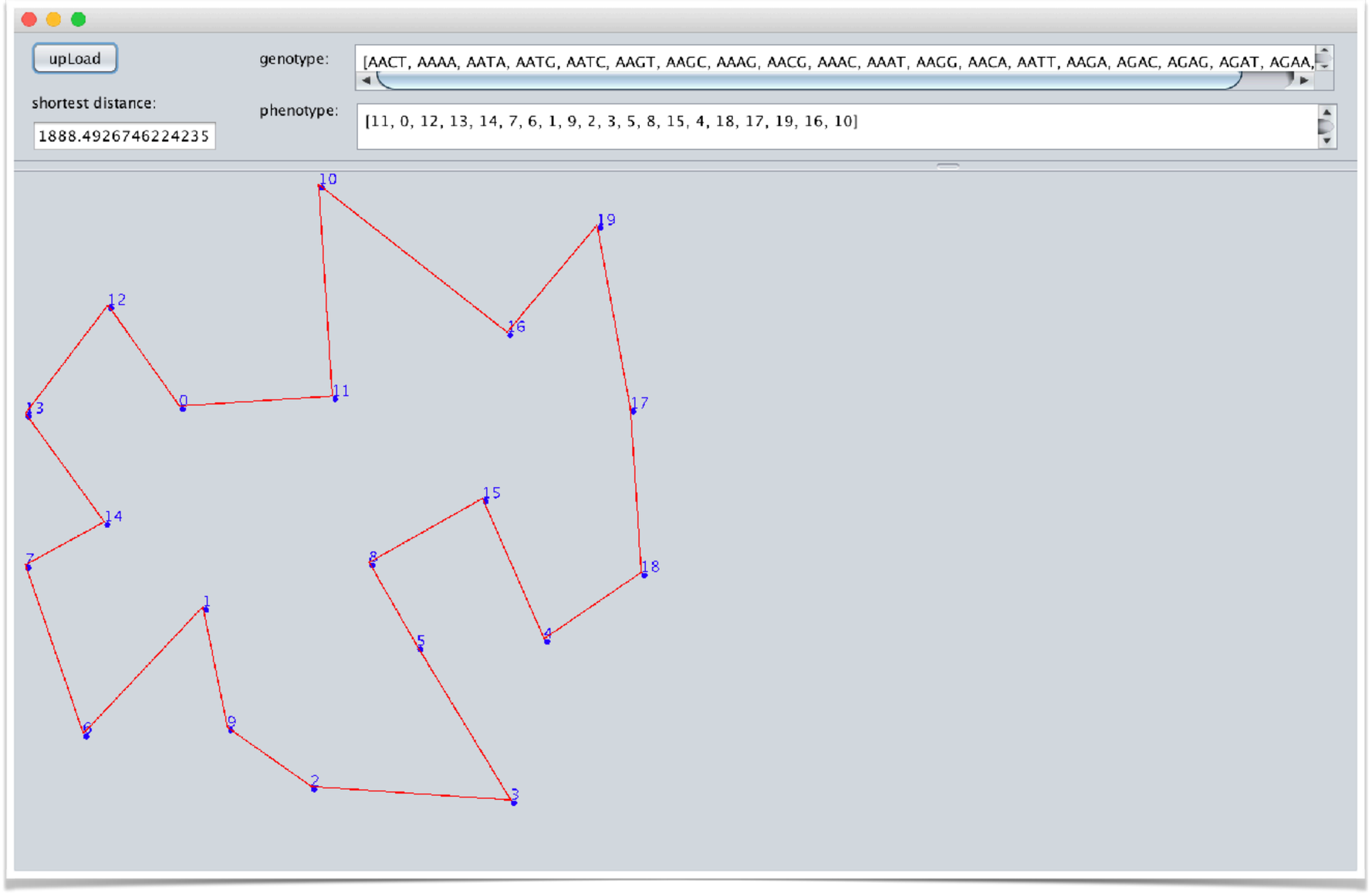
The Shortest Distance of Generation: 99 is 1740.9972056296554

The best phenotype is [10, 11, 8, 5, 15, 16, 19, 17, 18, 4, 3, 2, 9, 1, 6, 7, 14, 13, 12, 0]

The best genotype is [AACC, AACT, AACA, AAGG, AATT, AGAA, AGAT, AGAG, AGAC, AAGA, AAAT, AAAC, AACG, AAAG, AAGC, AAGT, AATC, AATG, AATA, AAAA]

OUTPUT

Data.txt



INPUT

Test2.txt

6734 1453	*4706 2674*	*4307 2322*	*7762 4595*
2233 10	*4612 2035*	*675 1006*	*7392 2244*
5530 1424	*6347 2683*	*7555 4819*	*3484 2829*
401 841	*6107 669*	*7541 3981*	*6271 2135*
3082 1644	*7611 5184*	*3177 756*	*4985 140*
7608 4458	*7462 3590*	*7352 4506*	*1916 1569*
7573 3716	*7732 4723*	*7545 2801*	*7280 4899*
7265 1268	*5900 3561*	*3245 3305*	*7509 3239*
6898 1885	*4483 3369*	*6426 3173*	*10 2676*
1112 2049	*6101 1110*	*4608 1198*	*6807 2993*
5468 2606	*5199 2182*	*23 2216*	*5185 3258*
5989 2873	*1633 2809*	*7248 3779*	*3023 1942*

OUTPUT

The Shortest Distance of Generation: 0 is 102097.31830401563

The Shortest Distance of Generation: 1 is 96040.111289962

The Shortest Distance of Generation: 2 is 87855.50710363482

The Shortest Distance of Generation: 3 is 86925.33780863695

The Shortest Distance of Generation: 4 is 82485.29897071126

The Shortest Distance of Generation: 5 is 81844.61361691824

The Shortest Distance of Generation: 6 is 75196.31287773496

The Shortest Distance of Generation: 7 is 68162.63450118943

The Shortest Distance of Generation: 8 is 66905.01454856411

The Shortest Distance of Generation: 9 is 64050.9194112519

The Shortest Distance of Generation: 10 is 64050.9194112519

The Shortest Distance of Generation: 11 is 60520.28557682435

The Shortest Distance of Generation: 12 is 59183.82900351878

The Shortest Distance of Generation: 13 is 58294.368755292606

(……)

(……)

The Shortest Distance of Generation: 97 is 34043.55902117573

The Shortest Distance of Generation: 98 is 34043.55902117573

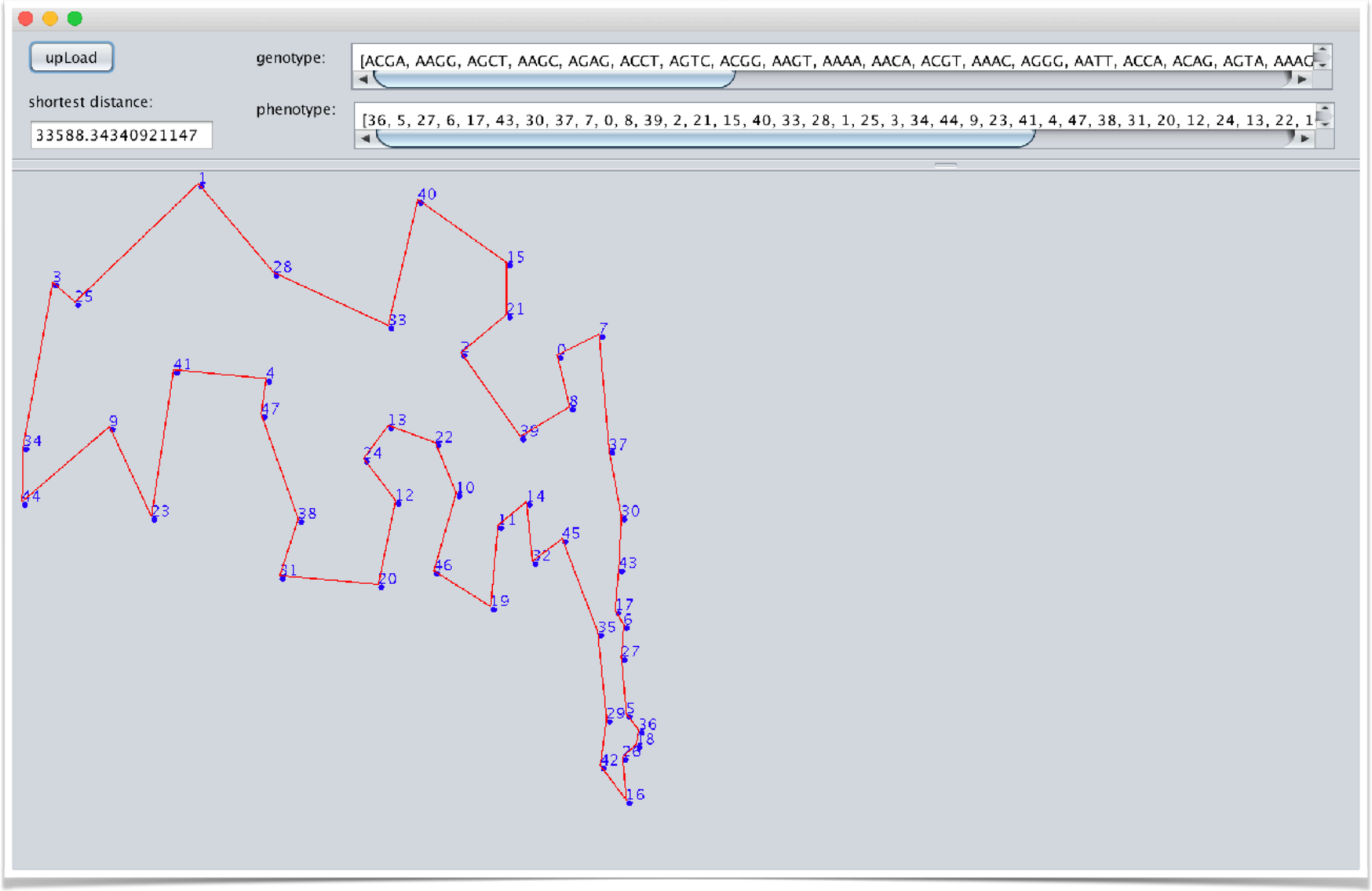
The Shortest Distance of Generation: 99 is 34043.55902117573

The best phenotype is [36, 5, 27, 6, 17, 43, 30, 37, 8, 7, 0, 15, 21, 2, 22, 12, 24, 13, 33, 40, 28, 1, 25, 3, 34, 44, 9, 23, 41, 4, 47, 38, 31, 20, 46, 10, 39, 14, 11, 19, 32, 45, 35, 29, 42, 16, 26, 18]

The best genotype is [ACGA, AAGG, AGCT, AAGC, AGAG, ACCT, AGTC, ACGG, AACA, AAGT, AAAA, AATT, AGGG, AAAC, AGGC, AATA, AGCA, AATG, ACAG, ACCA, AGTA, AAAG, AGCG, AAAT, ACAC, ACTA, AACG, AGGT, ACCG, AAGA, ACTT, ACGC, AGTT, AGGA, ACTC, AACC, ACGT, AATC, AACT, AGAT, ACAA, ACTG, ACAT, AGTG, ACCC, AGAA, AGCC, AGAC]

OUTPUT

Test2.txt



CONCLUSION

By changing some parameters, we found many parameters have influences to the algorithm:

Population Number: If we choose **large population number**, the algorithm will get the **best entity earlier**. That means to get the best entity in a single generate algorithm experiment, the larger the population is, the smaller generation number will be. And we need to choose larger population number for larger city number, or we will not get the correct answer even the generation number is very large.

Generation Number: We need **enough generations** to guarantee the best genes are chosen together and put in a single entity. In most case, if we keep the population number constantly, the more city we need to visit, the larger generation number will be needed.

CONCLUSION

Crossover possibility: We found crossover is an **amazing way to evolve better** entities. That is because, in every generation, we choose good entities to generate new population, so the crossover will combine good genes in different entities together as a new entity. This is pretty efficient. So we set the crossover possibility very close to "1" to make our algorithm efficiently.

Mutation possibility: Mutation is **not a good idea** in our algorithm, we found if we set the mutation possibility, the program will be less efficient (need larger generation number to get the best entity). That is because most mutations in nature are bad, and mutation is not as efficient as the crossover to combine good genes together. But what's interesting, the mutation is helpful to prevent local optimum happening. We set the mutation possibility close to "0" because our biodiversity is pretty well (population number is large enough to cover almost all the possible genotype segments). So if you need to solve a TSP with very large city number and limited population number, mutation is required.

CONCLUSION

Fitness and cull: We use the **total distance of each entity** to calculate the fitness function, fitness of each entity is the reciprocal of distance divided by the sum of fitness of all the entities in the given population. And we need to select some good entities to generate the new population. In this algorithm, we select top 10% of sorted entities. A bigger number will be less efficient because too many bad genes included while a smaller number will lose the diversity and run into the local optimum, so 10% is good. Self-crossover is allowed and the best entity of each generation will keep going without any mutation.

CONCLUSION

Although we did so much, the algorithm will still not generate the optimum solution (very very rare when the input is exactly large but exist). That is because the purposed idea of generation algorithm is use evolution to generate better and better entities but optimum cannot be guaranteed. Bigger population number and more generations can make the algorithm better but more run time needed which is a natural contradiction between performance and efficiency. All in all, the generation algorithm is a great idea to solve such problems (like NPC problems) and can be used in our natural life. The only thing we need to know is how to write a fitness function of genotype and choose the proper parameters. The algorithm itself will do other things to generate a good entity.