The Traveling Salesman Problem:

Diving into genetic algorithms

Srimalaya Ladha

301367941

CMPT 310: Artificial Intelligence Survey

Instructor: Toby Donaldson

April 18, 2020

Index

*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

Abstract

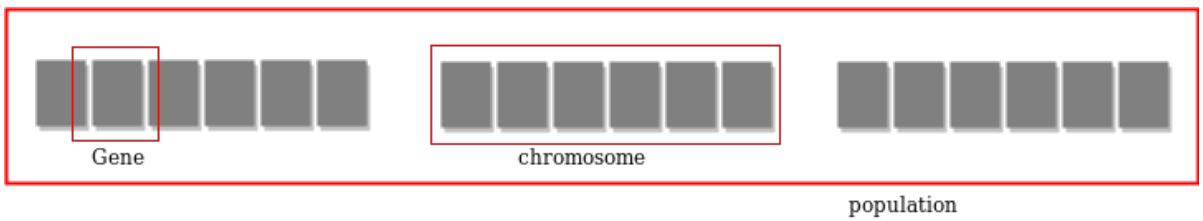
Traveling Salesman Problem (TSP) is an optimization problem that aims traversing through a given list of cities and their coordinates in the shortest possible route while visiting each city exactly once. Using the traditional mathematical methods, the problem of TSP becomes unsolvable as the number of cities to be visited increases. Therefore, we try to solve this problem using heuristics just like we did with other problems in this course. This assignment uses genetic algorithms based on different heuristics and compares the data procures using them to find out which setting is most optimal to find a good score.

*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

Introduction to Genetic Algorithms

Genetic algorithms are a heuristics that belong to the larger class of evolutionary algorithms which are inspired by the principle of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. GA's are commonly used to generate high-quality solutions to problems which rely heavily on biologically inspired algorithms like mutation, crossover, and selection. In this assignment, we use the Genetic Algorithm provided by Dr. Toby Donaldson which contains random best search, crossover search and mutation search. As per Dr. Donaldson, the TSP is a NP-hard problem which means that it doesn't have a polynomial running time function. Genetic algorithms are especially useful for solving NP-hard problems. They approach the problem in the following manner:

1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring
3. Genes from "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent
4. Each successive generation is more suited and is maintained in search space as a solution
5. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components)



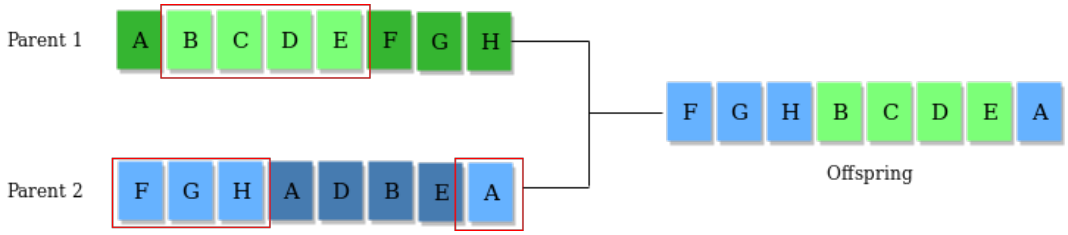
*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

Algorithm for this assignment

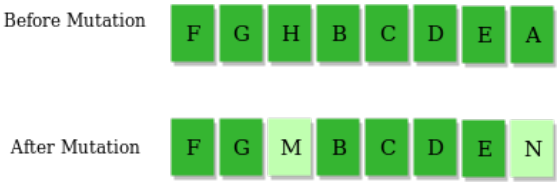
Since we are using Dr. Donaldson's tsp.py, we needed to make some changes pertaining to read/write/execution of the program. The first step is to write a function which generates a file such that it writes my space separated solution to the file. Then, the input condition was changed to accept any file the user may want to use to solve the TSP for such that they have to simply enter the file name when prompted to do so. The procedure generates an output which shows the actual best score for the selected heuristic. We followed the textbook approach which states that each genetic operator be used separately.

Upon carefully observing the code in tsp.py the following observations were made:

1. The program reads tuples of city coordinates as pairs and then randomly selects a city using permutation.
2. Genetics used:
    a. *crossover_search*: This represents mating between individuals. 50% individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a new offspring.



    b. *mutation_search*: The key idea is to insert random genes in offspring to maintain the diversity in population to avoid the premature convergence.



*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

3. Helper Functions:
    a. *is_good_perm*: Sorts and stores good permutations
    b. *city_dist*: Distance between two pairs of cities using Euclidean distance formula
    c. *do_rand_swap / rand_perm*: Randomizes swapping/permutations
    d. *pmx*: Helper for crossover_search
    e. *file_generator*: Writes to a new file named "cities1000_ladha.txt"

Trials

For this project, it was particularly challenging to find a optimal heuristic and even then, an optimal no of iterations and permutations to undergo with the selected heuristic. There were many trials and as stated below, most of them were too slow to consider.

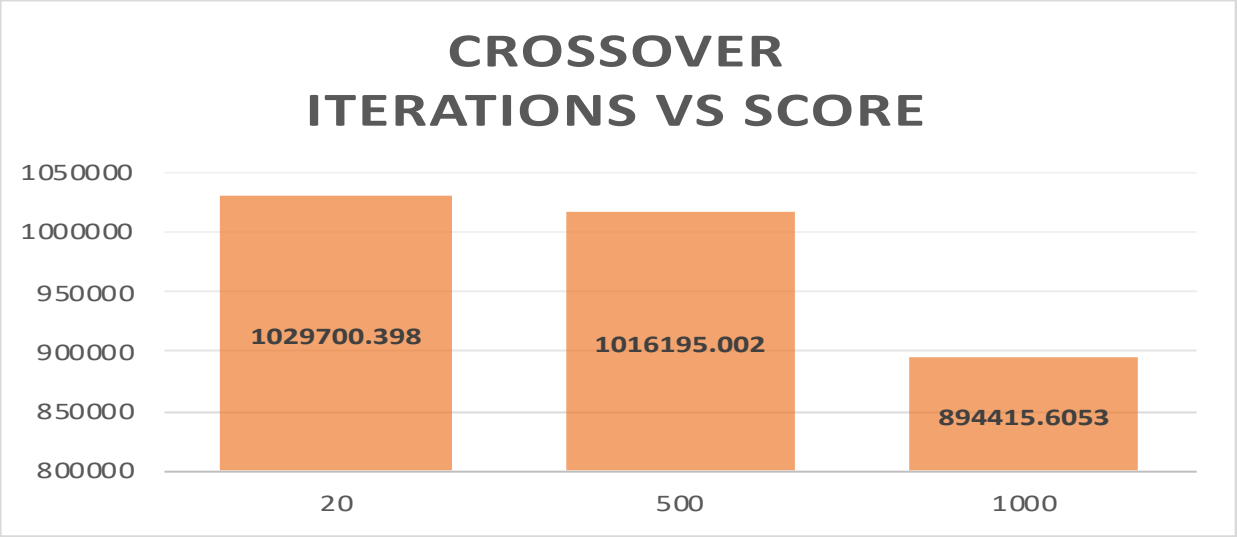
1. **Crossover**: Crossover search was very slow. For a set of 10,000 iterations it was able to generate close to 1,000,000 score which was pretty bad considering the time it was consuming to run. Moreover, the change in *pop_size* (doubling or halving) barely made a difference to that number.
2. **Selection:** Selection search give preference to individuals with good fitness score, and allows them to pass their genes the successive generations. The problem that I faced with selection search was that it was very slow in execution. Slower than crossover search. Therefore, I decided not to go ahead with it's implementation.
3. **Mutation:** Mutation is a very handy heuristic and although not much faster than Crossover, give much lower scores for total distance which is very good. I ran mutation search side by side with Crossover search. The results are on the data page. On average, mutation search was able to get a score around 160,000 which is decent compared to the other two algorithms.

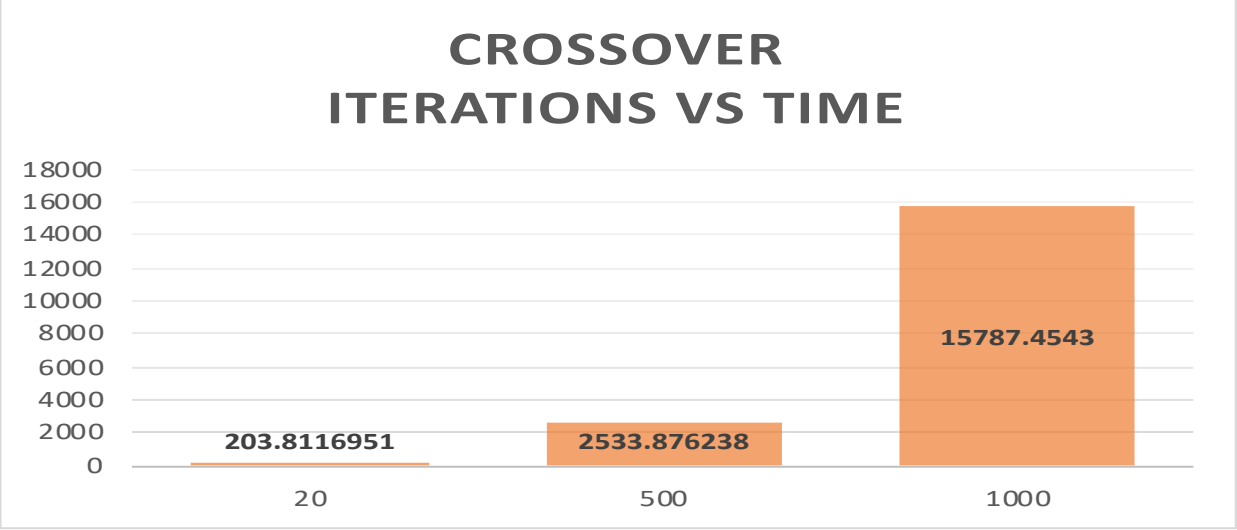
*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

Data (for 'cities1000.txt')

Crossover			
Permutations	Iterations	Score	Time (in S)
20	10000	1029700.398	203.8116951
500	10000	1016195.002	2533.876238
1000	10000	894415.6053	15787.4543
20	100000	1039922.119	12546.02354

Maximum Score:	1039922.119	Maximum time:	15787.4543
Minimum Score:	894415.6053	Minimum time:	203.8116951
Average Score:	995058.2811	Average time:	7767.791444

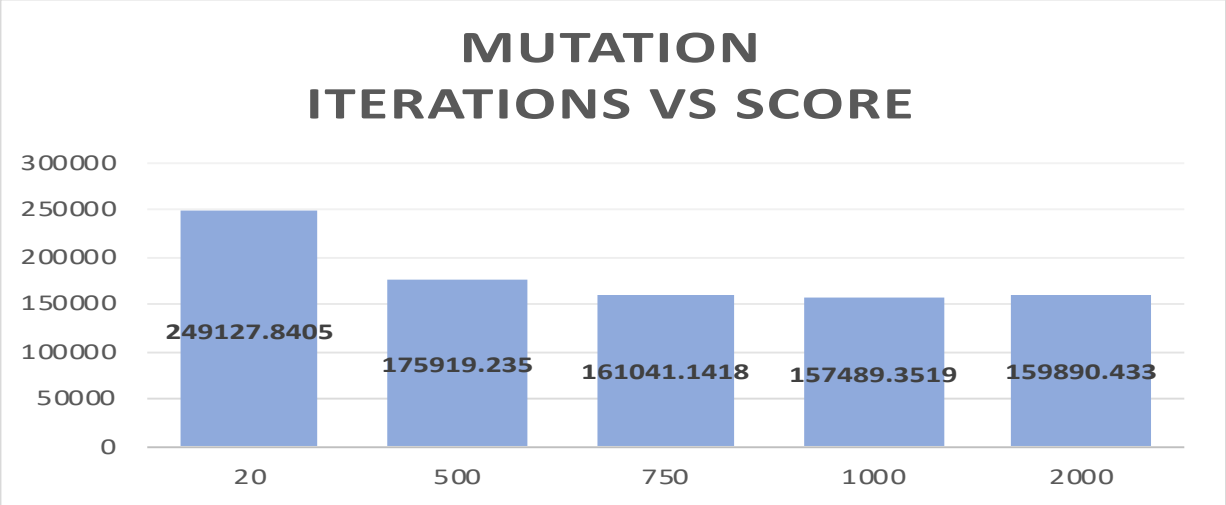


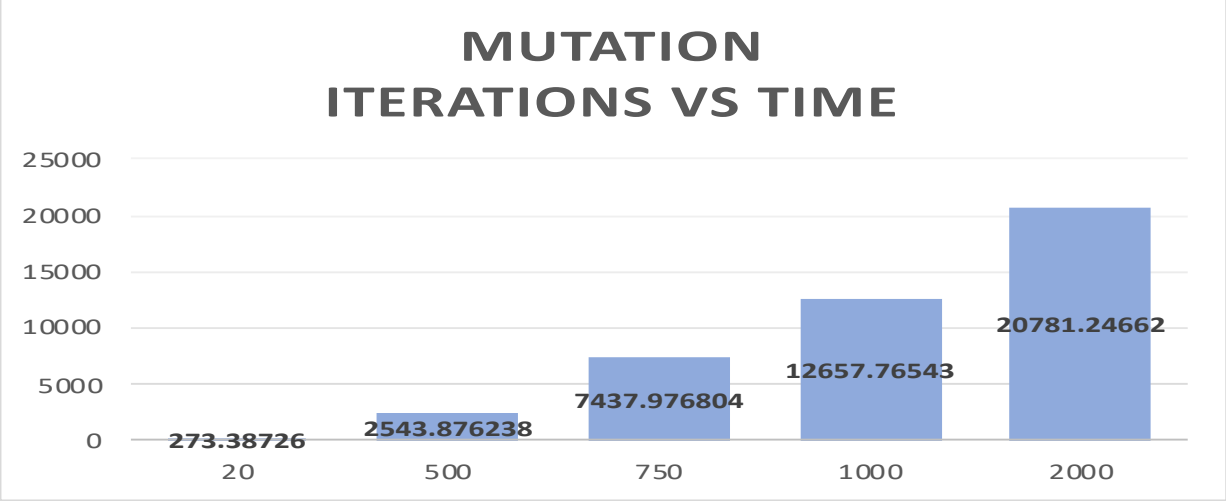


*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

Mutation			
Permutations	Iterations	Score	Time (in s)
20	10000	249127.8405	273.38726
500	10000	175919.235	2543.876238
750	10000	161041.1418	7437.976804
1000	10000	157489.3519	12657.76543
2000	10000	159890.433	20781.24662
20	100000	166494.3749	10359.23466

Maximum Score:	249127.8405	Maximum time:	20781.24662
Minimum Score:	157489.3519	Minimum time:	273.38726
Average Score:	178327.0629	Average time:	9008.914501





*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

Conclusion

As evident by the above data and figures, it can be concluded that mutation is by far the best heuristic technique to use for TSP using Genetic Algorithms. It is superior to crossover_search in every aspect. Moreover, for above data generated for “cities1000.txt”, the best score appears to be 157489.3519 with pop_size = 1000 and max_iterations = 10,000. Furthermore, it was observed that upon further incrementing the pop_size, no visible improvement in score is evident, therefore the maximum pop_size that I would recommend would be 1000.

*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.

Reference(s)

GeeksForGeeks.com: https://www.geeksforgeeks.org/genetic-algorithms/

Genetic Algorithm Notes. Genetic Algorithm Notes - cmpt310spring2020 Documentation, www2.cs.sfu.ca/CourseCentral/310/tjd/geneticAlgorithmNotes.htm

Russell, S., & Norvig, P. (2010). *Artificial intelligence : A modern approach / Stuart J. Russell and Peter Norvig ; contributing writers, Ernest Davis [and others].* (3rd ed., Prentice Hall series in artificial intelligence). Prentice Hall.

*Run assignment by launching FinalProject.py in terminal. Enter the file name that you want to test when prompted.