Solving Travelling Salesman Problem with Genetic Algorithms

Tony Shen

CMPT310 Artificial Intelligence Survey

Simon Fraser University

Email: tsa87@sfu.ca

Abstract

The traveling salesman problem (TSP) is a NP-hard problem; Given the absence of polynomial time algorithms yielding optimal solutions, genetic algorithm (GA) is a candidate to compute an approximation to the solution of TSP.

In this experiment, novel techniques and biologically inspired mutation are first benchmarked, then optimized to generate the solution to "cities1000" TSP. Using Live and Learn method, the total distance of sales tour for "cities1000" converged at 58,650 steps, down from 1,080,000.

Genetic Algorithm Framework

The genetic algorithm framework was implemented in Python3. In addition to features typically found in GA solving TSP[1], this implementation has included the following custom features.

Naïve Greedy Initialization:

Naïve Greedy Algorithm solves TSP by continuously visiting the closest unvisited city to its own list. The solution from this greedy initialization can be set as the initial population of GA, to cut down convergence time. The default initialization method, however, is still a random permutation of the cities.

Growth Factor:

In vanilla GA, each epoch generats just enough offspring to retain a constant population, with all selectivity based on reproduction. In this implementation, the off-spring population is a growth factor of the original population size. Only the top N off-spring are carried over to the next generation. This allows a fitter population.

Pseudo Algorithm:

1. Initialize the population randomly or using naïve greedy algorithm.
2. Create off-springs with Live and Learn method or PMX method.
3. Pick the N off springs with highest fitness level to survive.
4. Randomly apply mutations on the off springs.
5. Repeat step 2 to step 4

Instruction:

- >>> python3 tsp.py -f cities1000.txt
- (optional) change parameters of GA initialization according to line 42.

[1] https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35

Techniques:

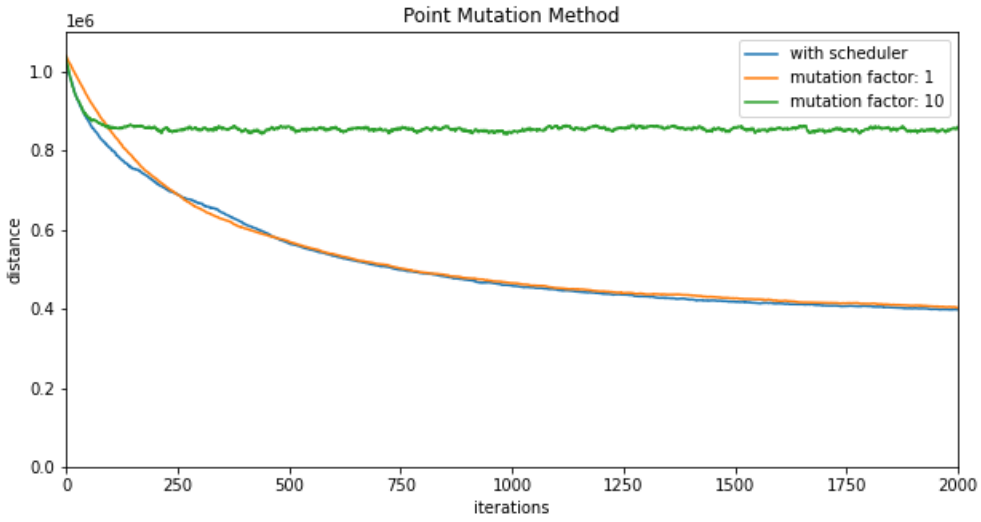
Point Mutation Method:

Point mutation method, inspired by the mutation of the same name found in nature[2], is defined as the swapping of 2 random positions in the sequence of the organism, M (mutation factor) times. This mutation guarantees the preservation of the Hamiltonian Cycle property required for TSP.

Ex. [3-2-6-5-4-1] → A Point Mutation at index 2, 5 → [3-1-6-5-4-2]

This mutation method by itself, is not particularly effective; After 2,000 iterations (with size=10, growth_factor=10, mutation_factor=1) the minimum distance converges around 408,000. With a high mutation factor, the distance metric can fail to converge.

A mutation factor scheduler [mutation_factor=max(((distance - 600,000)/100,000), 0)+1)] was implemented to take advantage of the fast convergence early on. However, it leads to no improvement to the final score, as expected.



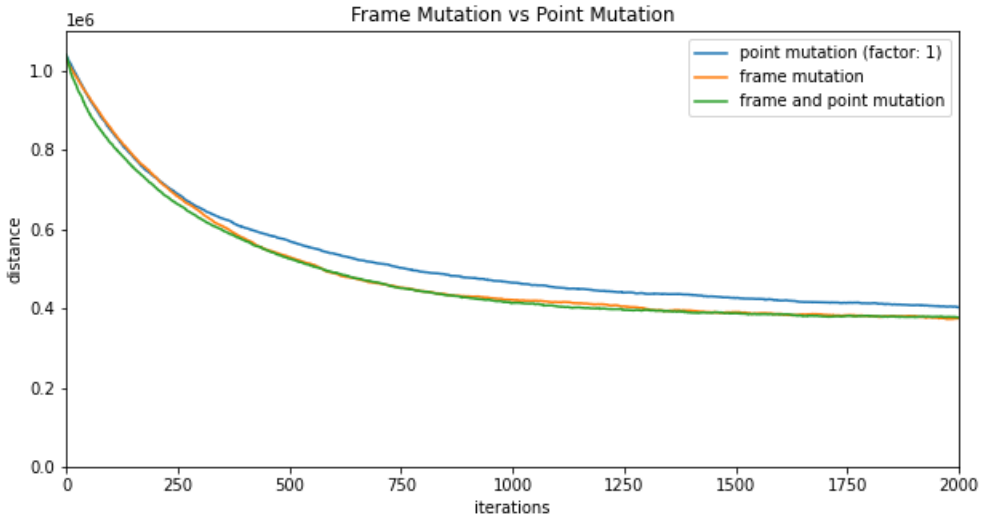
Frame Mutation Method:

Frame mutation method, biologically inspired by Frame-Shift mutation[3], picks an arbitrary segment of DNA from the organism, and inserts the segment back to a random index point. Like Point Mutation, frames mutation also preserves the Hamiltonian Cycle property.

Ex. [6-1-2-4-3-5] → Segment from index 2, 3. Insert at index 4 → [6-1-3-2-4-5]
Ex. [6-1-2-4-3-5] → Segment from index 5, 1. Insert at index 2 → [1-2-5-6-4-3]

[2] https://www.youtube.com/watch?v=xYOK-yzUWSI
[3] https://www.youtube.com/watch?v=xYOK-yzUWSI

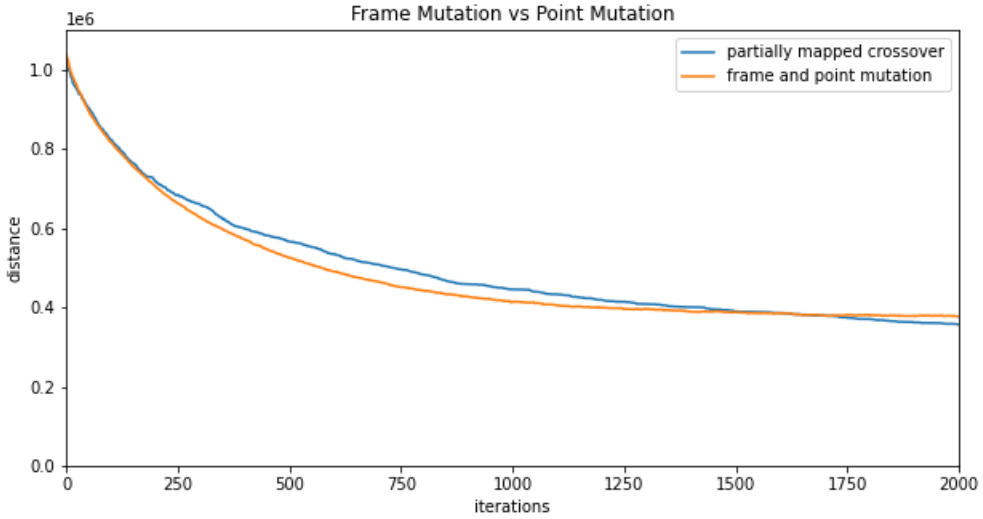
Frame mutation converges at 377,701 after 2,000 iterations, a slight improvement from point mutation holding all other parameter equal. These 2 mutations combined together (each with 50% of happening) yields no noticable benefit.



Partially Mapped Crossover Method:

Adapted from "Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation"[4], Partially Mapped Crossover (PMX) method, combined with asexual mutation (p=10%) resulted in slow, but steady downward convergence.

After 2,000 iterations, PMX methods still hasn't flattened out its downward trend at 357,086 steps.



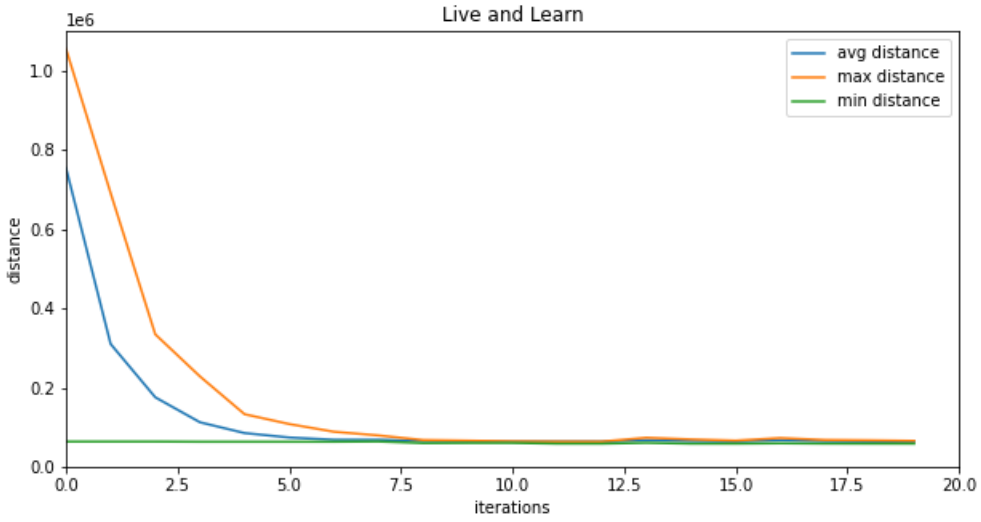
[4] http://user.ceng.metu.edu.tr/~ucoluk/research/publications/tspnew.pdf

Live and Learn Method:
In real life, your gene doesn't deterministically decide the outcome of your life. From this insight and inspiration from the Ant Colony Algorithm[5], partially mapped crossover method is modified to create the Live and Learn method.

In Live and Learn method, an offspring inherits its ancestor's genome up to a randomly selected index P. To complete its sequence, the offspring continuously selects the closest unvisited city, until it completes the Hamiltonian Cycle.

Ex. [1-4-3-2-6-5] → P randomly chosen to be 4 → [1-4-3-2-5-6]
(assuming 2 is closer to 5 than to 6)

Combined with some asexual mutation (p=10%), Live an Learn offers incredibly fast convergence for the "cities1000" problem. Within 20 iterations, Live and learn method converged on paths requiring only 58,650 steps, down from more than one million steps.



Conclusion:

Generic genetic algorithm, albeit appealing due to Theory of Evolution, is unable to converge on some problems in a timely manner. Integrating self-altering and learning frameworks, so the fate of organisms are not purely deterministic of their genetic makeup, could perhaps boost the efficiency of genetic algorithm.

[5] https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms