20160025 고건호

CS454 AI Based Software Engineering

Assignment #2: Traveling Salesman Problem

Due by 23:59, 12 November 2021

Abstract

Since Traveling Salesman problem(TSP) is a NP problem, it is difficult to find a optimal solution when n is large. So instead, we can use meta-heuristic algorithm to get 'good enough' solution. In this solver, I adopted Genetic Algorithm(GA). And because we aim for a case where n is large, I think fast exploitation is more important, which means that it is more efficient to reduce the fitness value within a given time than to find a global optima. To make this idea come true, I used greedy solution as a seed, and made edges who are longer than other more likely to be mutated.

Interface

This solver provides following parameters.

- Population (-p) : number of population in a generation. Default value is 100.
- Maximum generations (-g) : number of generations to execute. Default value is 1000.
- Save log (-s, --save) : Save the last generation of solutions so that they can be used as seeds for the next execution. We can specify the name of the file, or it will be saved in "solution_log.csv".
- Load log (-u, --upload) : Load solutions for the seeds. You can specify the name of the file, or it will be loaded from "solution_log.csv"
- Repeat (-r, --repeat) : Repeat algorithm with given number of times.
- Show fitness value (--show-fitness) : After the execution, it will show graph of fitness value by generation.

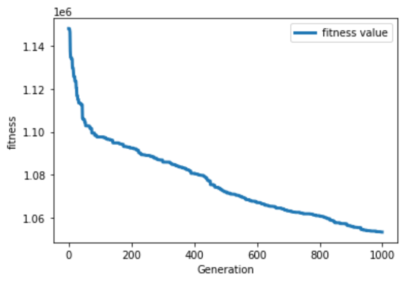


Figure1 Fitness value by Generation

Implementation

To be honest, my first goal was to win the leaderboard. And since the size of the problem is large(11849), I should think of an efficient way to reduce the fitness value. One way was to give greedy solution as a seed, and modify the mutation rate according to the length of the edge.

Initialization – Greedy Seed

In fact, greedy solution can be a pretty good solution. Figure2 is comparison between greedy solution and optimal solution for a280.tsp. Some edges are bad, but most edges are almost the same. We can develop solution by preserving good edges, and improving bad edges.

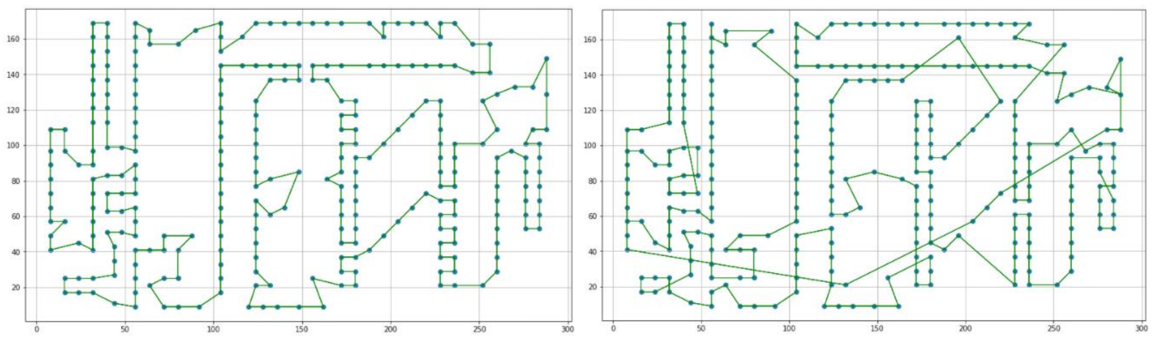


Figure2 Optimal solution(left) and greedy solution(right) for a280.tsp

One concern is that starting with one seed can be a factor that harm diversity. It might be true, but when I compared fitness values of 'greedy seed' and 'random seeds', 'random seeds' solution never outpaced 'greedy seed' solution.

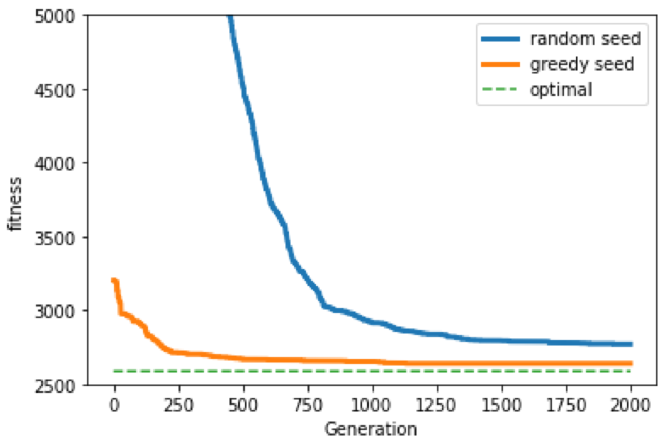


Figure2 Fitness value of 'random seed' solution and 'greedy seed' solution for a280.tsp by generation

Mutation - Longer Mutation

I used 2-opt for basic mutation strategy. Solver choose random two edges and exchange them.

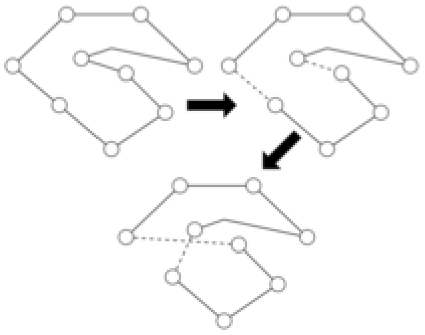


Figure3 Example of 2-opt iteration

However, even after many generations passed, bad edges still remained. This is because the probability that two edges will exchanged as we expected is only $1/n^2$. In fact, our goal is to preserve good edges(whose length is small) and exchange bad edges(whose length is big). And also, it is better for two edges to be exchanged are close with each other. So I modified the probability of mutation as follow

$$\mathrm{prob}_{\mathrm{mutate}}(\mathrm{E1}) \propto E1.length^2$$
$$\mathrm{prob}_{\mathrm{exchange}}(E1, E2) \propto \frac{E2.length^2}{distance(E1, E2)^2}$$

Figure4 shows the fitness value reduction of 'Longer Mutation' is much faster than 'Random Mutation'.

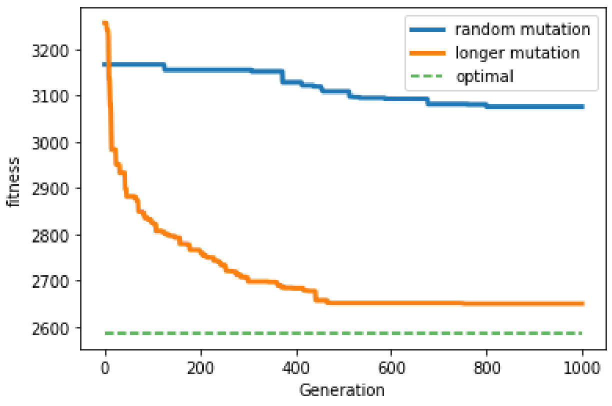


Figure4 fitness value of 'random mutation' and 'longer mutation' for a280.tsp by generation

Crossover

I used typical crossover strategy for sequential solution. Select two cross-point, and extract subsequence between two cross-points from parents. And the rest of the sequence will be filled in the order of the other parent's solution without overlapping numbers.

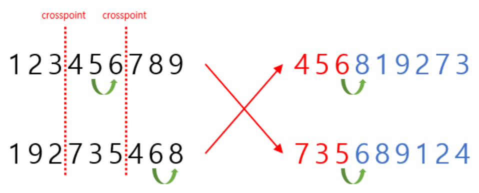


Figure5 Crossover example

Other Strategies

- Children will replace half of selected population.
- Solution always starts with '1' in order to reduce redundancy

  (Limitation : I didn't yet consider the case of reversed solution)
- Duplicate solutions are removed at the beginning of every generation.

Result

I applied solver to rl11849.tsp. At first iteration(1000 generations), the fitness value reached about 1.05M, and it took about an hour.

```
fitness value 1053471.9233813386: 100%|██████████| 1000/1000 [54:54<00:00,  3.29s/it]
```

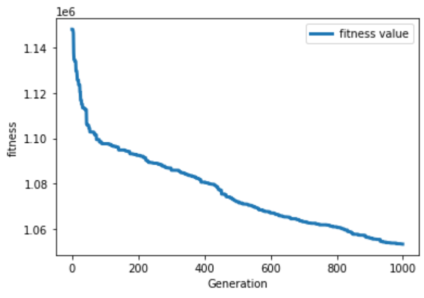
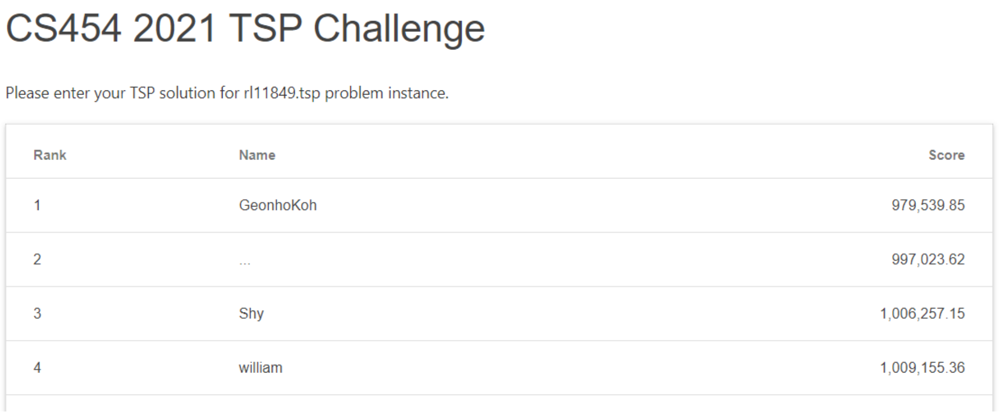


Figure6 First iteration for rl11849.tsp

And after few more iterations(after 10,000 generations), fitness value reached 978K. At this point, I stopped solving because fitness value reduction per iteration was only about 1.5K. This score currently ranks first on the leaderboard.



CS454 2021 TSP Challenge

Please enter your TSP solution for rl11849.tsp problem instance.

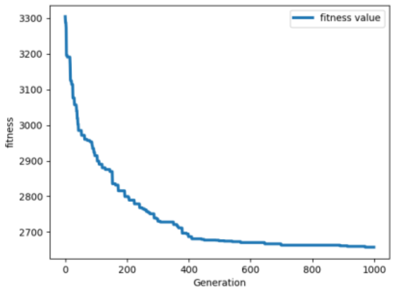
Rank	Name	Score
1	GeonhoKoh	979,539.85
2	...	997,023.62
3	Shy	1,006,257.15
4	william	1,009,155.36

Figure7 Leaderboard score

Solver Examples

1. Change population

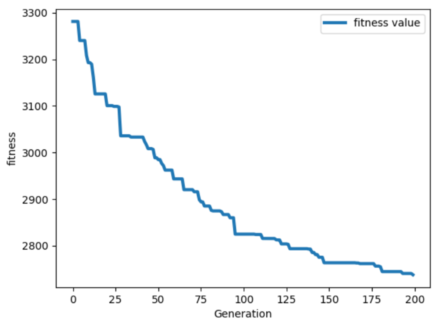
```
> python3 TSP.py -p 200 --show-fitness a280.tsp > solution.csv
fitness value 2687.955556797561: 100%|████████████████████████████████| 1000/1000 [02:13<00:00,  7.47it/s]
```

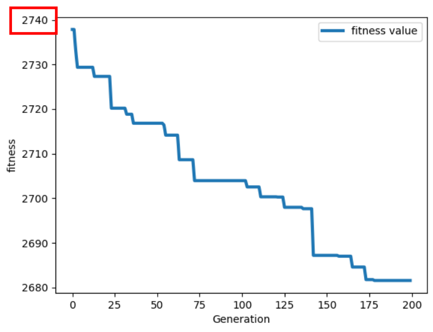


2. Save and Load solution log

```
> python3 TSP.py -g 200 -s --show-fitness a280.tsp > solution.csv
fitness value 2740.6791618620155: 100%|████████████████████████████████| 200/200 [00:13<00:00, 14.64it/s]

> python3 TSP.py -g 200 -u --show-fitness a280.tsp > solution.csv
fitness value 2681.58077777883: 100%|█████████████████████████████████| 200/200 [00:13<00:00, 14.80it/s]
```





When we save and load solutions, second execution starts from last execution's best solution.