TSP Challenge week 6

last week's gist : https://gist.github.com/hono-mame/42a169bcdedfb296f94b5e61a504caed
last week's document : TSP challenge
gist : https://gist.github.com/hono-mame/86c4728a9452d87003465ae126a693ec
results : TSP Results

By introducing annealing method, and combining it with the greedy and 2-opt methods, I investigated several ways to find the shortest path.

① greedy_and_2-opt.py

Using the greedy method, first obtain the route taken by connecting the shortest points at that time. For that result, the shortest path is obtained by performing 2-opt and eliminating the intersecting points.

Improvements from last week

・Update search_all function
→ Find the shortest distance more accurately when the data is less than 10

Results

challenge	Challenge 0	Challenge 1	Challenge 2	Challenge 3	Challenge 4	Challenge 5	Challenge 6	Challenge 7
data	N = 5	N = 8	N = 16	N = 64	N = 128	N = 512	N = 2048	N = 8192
score	3291.62	3778.72	4494.42	8970.052801	11489.79	21,363.60	42,712.37	84190.72157

② simulated_annealing _and_2-opt.py

The annealing method is run repeatedly and the shortest distance among them is taken as the result. Then, 2-opt is performed to see if there is a path with an even shorter distance.

Results (improve : compares scores of ①)

challenge	Challenge 0	Challenge 1	Challenge 2	Challenge 3	Challenge 4	Challenge 5	Challenge 6	Challenge 7
data	N = 5	N = 8	N = 16	N = 64	N = 128	N = 512	N = 2048	N = 8192
score	3291.62	3778.72	4494.42	8326.79	11388.84	22,370.53	44,025.94	88844.84
improve	0	0	0	643.26	100.95	-1006.93	-1313.57	-4654.12

Consideration for ②

This method was very effective for challenge3 and challenge4. However, as the number of data increased, the method ① was superior. I believe that this is because the number of data is too large, since tours are first randomly created and then randomly re-connected, and the optimization criterion may have become too high before the number of data is sufficiently optimized, and thus the ideal result could not be obtained. Therefore, to improve this, I considered using greedy instead of creating a tour at random.

③ greedy_simulated_annealing_and_2-opt.py

Instead of making a tour at random when we execute simulated annealing, we make a tour by using greedy methods. The resulting value will be the same as or better than ①, since we make a tour by using greedy first.

Improvements

made a new function to make a tour by using greedy methods so that simulated_annealing function became simpler :)

Results (improve : compares scores of ①)

challenge	Challenge 0	Challenge 1	Challenge 2	Challenge 3	Challenge 4	Challenge 5	Challenge 6	Challenge 7
data	N = 5	N = 8	N = 16	N = 64	N = 128	N = 512	N = 2048	N = 8192
score	3291.62	3778.72	4494.42	8209.32	11230.47	21,363.60	42712.37	84190.72
improve	0	0	0	760.73	259.32	0	0	0

Consideration for ③

The values were highly variable, but better results were obtained after several runs.
However, I could not get better results than ① (= greedy) when the data is very large.

<Conclusion>

Good results have been obtained with a moderate number of data. However, I have not found a better method than greedy when the number of data is larger.
Better results could be obtained by adjusting the parameters of the annealing method, but adjusting these parameters was difficult.

(Extra) tsp.c

I rewrote the ① greedy_and_2-opt.py in C. I could not finish “search_all”, so this is not perfect.

Results

challenge	Challenge 0	Challenge 1	Challenge 2	Challenge 3	Challenge 4	Challenge 5	Challenge 6	Challenge 7
data	N = 5	N = 8	N = 16	N = 64	N = 128	N = 512	N = 2048	N = 8192
score	0	0	4494.88	8970.05	11489.79	21159.13	42712.37	84190.72

In all cases, it was executed within **3 seconds**, confirming that it was very fast.
(in ①, it was executed in **323 seconds** in challenge7.)
The results were almost identical to ① greedy_and_2-opt.py.

<References>

https://qiita.com/take314/items/7eae18045e989d7eaf52
https://github.com/hayatoito/google-step-tsp
https://en.wikipedia.org/wiki/2-opt
http://www.nct9.ne.jp/m_hiroi/light/pyalgo64.html
https://en.wikipedia.org/wiki/3-opt