GA with Wisdom of Artificial Crowds

Lomus Hona
[Computer Science and Engineering]
Speed School of Engineering
University of Louisville
l0hona01@louisville.edu

1. What did you do in this project and why?

For this project I used genetic algorithm with tournament selection, cycle crossover and mutation rate of 50% to create an Artificial crowd for given tsp files, one solution per generation. Mutation rate is 50% because more diverse solutions give better results when applying wisdom of crowd. After getting the all the solutions via genetic algorithm, frequency matrix was created to calculate votes as in which pair of cities (from x city to y city) was most frequently occurring in crowds solution.

2. Describe In detail the algorithm you used to aggregate opinions. Did you have to alter the combined solution to make it a valid TSP solution?

Figure 1

```
[[0, 1, 18, 0, 0, 18, 43, 2, 0, 16, 2],
 [1, 0, 2, 7, 65, 0, 0, 2, 21, 1, 0],
 [6, 0, 0, 0, 0, 0, 56, 8, 0, 19, 11],
 [0, 68, 0, 0, 13, 1, 0, 1, 1, 1, 15],
 [0, 0, 0, 0, 0, 0, 0, 0, 78, 0, 0],
 [6, 1, 0, 58, 0, 0, 0, 22, 0, 8, 5],
 [37, 0, 38, 0, 0, 1, 0, 3, 0, 12, 9],
 [1, 1, 0, 6, 0, 49, 0, 0, 0, 19, 24],
 [0, 1, 0, 0, 22, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 3, 0, 18, 0, 44, 0, 0, 34],
 [0, 24, 0, 26, 0, 8, 0, 18, 0, 24, 0]]
```

Figure 2

vote,	from,	to
78	4	8
68	3	1
65	1	4
58	5	3
56	2	6
49	7	5
44	9	7
43	0	6
38	6	2

Figure 3

vote,	from,	to
78	4	8
22	8	4
1	8	1
65	1	4
21	1	8
7	1	3

```
[4, 8, 1, 3, 10, 9, 7, 5, 0, 6, 2]
```

First of all, I generated a frequency matrix to calculate how many times a city x is visited from city y. For example if an individual from the crowd has picked to go from city 5 to city 8, row 5 and column 8 of the matrix is incremented. Example of frequency matrix for 11 cities with 100 crowds shown by figure 1.

After the frequency matrix has been created, I utilize it to create and initialize an array of objects its properties count, from_city, and to_city, where count is number of individuals that picked the given path from_city to to_city. The array of objects was then sorted in descending order using the count/vote property shown on figure 2. And to pick a city I used a greedy method of pick the highest city from and to, then picked found the highest vote for the city to and went to its city to. For example shown in figure 3:

the path initially starts with $4 \rightarrow 8$ because it has the highest count/vote then we find the highest vote starting from city 8 which is $8 \rightarrow 4$, but since 4 is already in path we go to next largest vote $8 \rightarrow 1$ giving us $4 \rightarrow 8 \rightarrow 1$ and similarly $1 \rightarrow 3$ which results in $4 \rightarrow 8 \rightarrow 1 \rightarrow 3$ and so on.

Drawback to my approach:

The best approach to consider best solution based on votes would be a path with highest vote count. But that would require algorithm to generate every possible path or 11 factorial in the given case, and since we have test cases with up to 222 cities which is impossible to compute, and I couldn't figure out any other solution to it so I went with greedy approach which doesn't represent the votes efficiently all the time but I think represents it well enough to be above average.

3. On Average how well did the Wisdom of Crowds approach perform compared to the standard GA?

For GA with 500 generation and 100 population per generation, 10 trials were performed. And the results for running each of the tsp files was performed. results are as follows (GA&WOC) generated using GA_all_gen:

Random11.tsp

GA_all gen	GA & WOC	GA_Final
282.08	321.45	319.70
282.08	326.80	282.08
282.08	298.72	290.33
282.08	300.85	293.68
282.08	321.45	287.21
282.08	347.31	299.02
282.08	298.72	298.43
282.08	387.60	282.63
282.08	327.67	295.47
282.08	321.45	295.19

count WOC was better than GA_Final: 0
count WOC was better than best_GA_all_gen: 0

Random22.tsp

GA_all gen	GA & WOC	GA_Final
413.17	444.39	460.87
410.06	413.16	447.07
439.64	397.23	482.91
406.78	450.32	453.83
446.52	441.71	540.14
421.53	457.35	544.75
411.54	413.50	455.51
417.30	490.66	472.44
397.69	476.74	509.49
399.63	396.38	454.21

count WOC was better than GA_Final: 9
count WOC was better than best_GA_all_gen: 3

Random44.tsp

GA_all gen	GA & WOC	GA_Final
724.86	833.29	788.88
735.67	849.52	821.60
687.85	726.82	751.47
689.69	793.60	740.03
707.32	782.36	765.45
823.74	910.13	897.27
707.44	718.36	799.02
707.20	654.63	742.21
772.19	792.19	830.44
733.42	843.86	809.76

count WOC was better than GA_Final: 4
count WOC was better than best_GA_all_gen: 1

Random77.tsp

GA_all gen	GA & WOC	GA_Final
1139.05	1278.20	1139.05
1187.31	1585.96	1189.21
1136.23	1504.12	1136.23
1099.48	1287.17	1099.48
1092.02	1263.07	1092.02
1177.47	1519.87	1178.68
1238.91	1464.93	1239.28
1147.92	1400.20	1148.11
1184.43	1463.52	1184.43
1206.00	1343.17	1206.00

count WOC was better than GA_Final: 0
count WOC was better than best_GA_all_gen: 0

Random97.tsp

GA_all gen	GA & WOC	GA_Final
1391.26	1838.54	1391.26
1409.41	1881.60	1409.41
1272.29	1610.11	1272.29
1465.58	2020.50	1465.58
1385.49	1737.99	1385.49
1584.37	1953.37	1584.37
1548.44	1926.80	1548.44
1520.55	1895.71	1520.55
1419.98	1896.78	1419.98
1561.49	1989.02	1561.49

count WOC was better than GA_Final: 0
count WOC was better than best_GA_all_gen: 0

Random222.tsp

GA_all gen	GA & WOC	GA_Final
4067.62	5219.51	4067.62
4304.60	5355.74	4304.60
4255.26	5378.32	4255.26
4340.16	5632.65	4340.16
4337.72	5323.06	4337.72
4340.99	5705.71	4340.99
4456.19	5423.89	4456.19
4386.63	5398.57	4386.63
4780.40	6118.74	4780.40
4208.70	5447.59	4208.70

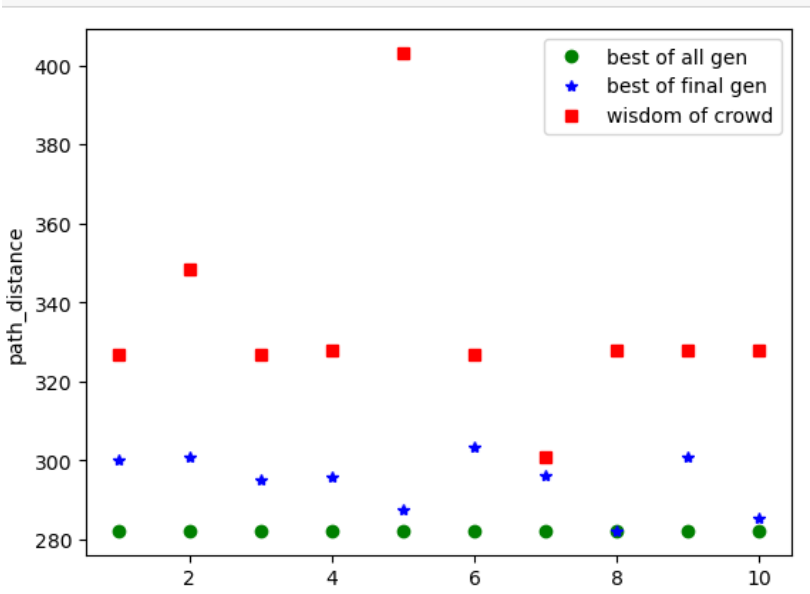
count WOC was better than GA_Final: 0
count WOC was better than best_GA_all_gen: 0

almost all the time, for my current implementation for WOC perform it performs worse than GA.

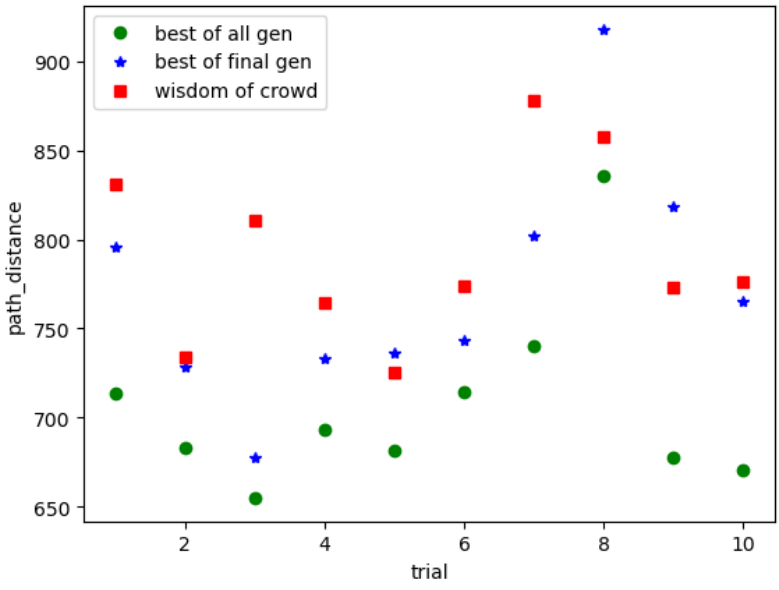
4. Comparison chart for GA vs (GA & WOC) on same problems in terms of performance, speed, optimality of discovered solutions

**figure not related to trial above*

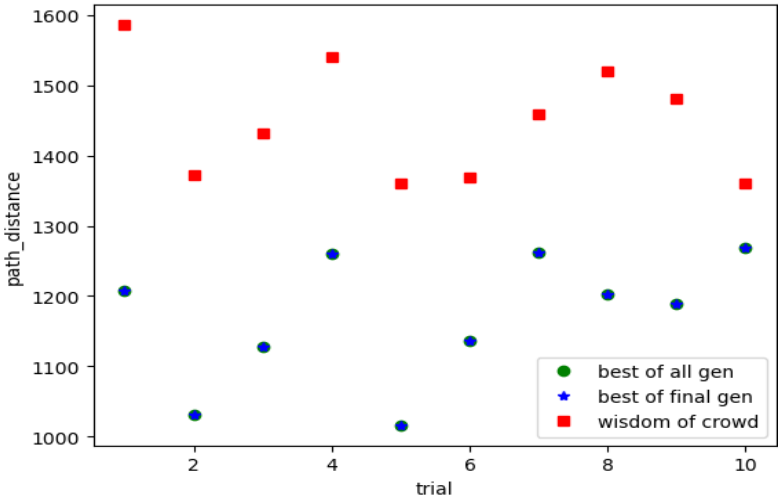
Random11.tsp



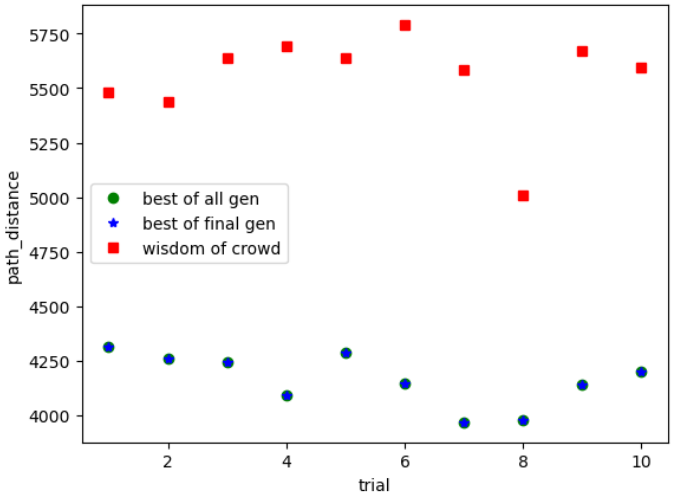
Random44.tsp



Random77.tsp

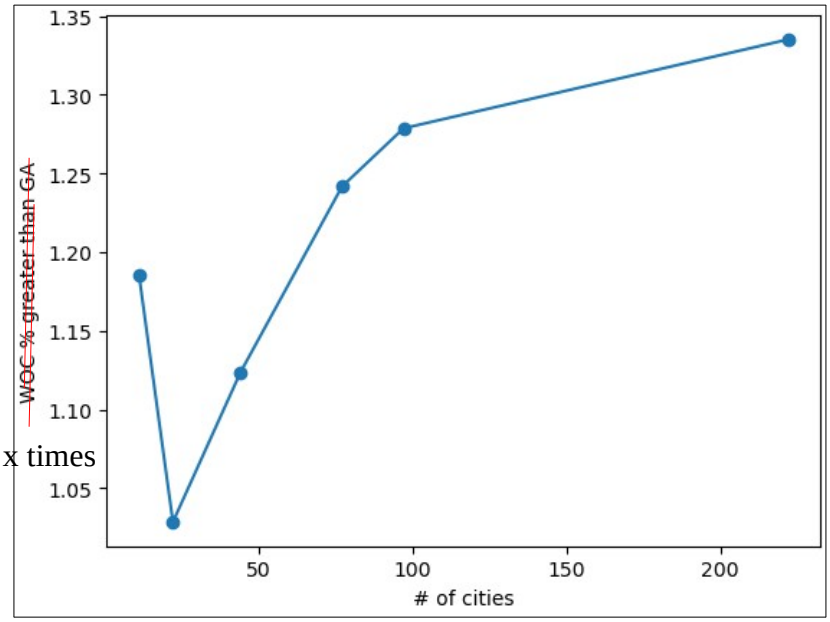


Random222.tsp



wisdom of crowd distance higher by x times compared to genetic algorithm for number of cities

Woc higher by x times



Disscussion/Result:

The results with the greedy approach I used to go to next city didn't efficiently justify crowds vote. I think I should have converted frequency matrix to cost matrix and then utilised LKH heuristic search method, which I had trouble making it work so opted to report what I had thus far. Simply put the whole thing suffers from drawbacks I mentioned part 1. and though the result isn't optimal I think what the method does do is for most part give a path that is sort of average most of the time.