Assignment 2

Xiaolan Situ(#20561073)

Q1

1. **State**: A complete tour of all the cities starting from city A and ending with city A.
   The tour visits each city exactly once.
   Neighbour relation: swap neighbouring nodes
   Cost function: The sum of the cost of all edges on the tour

2.

14-cities Instances										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
AvgStep	7.2	6.45	6.85	6.51	6.34	6.52	6.72	6.41	6.61	6.47
AvgQuality	1.51	1.77	1.62	1.74	1.73	1.64	1.57	1.40	1.56	1.62
Avg%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

15-cities Instances										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
AvgStep	7.6	7.59	7.46	7.71	7.76	7.72	7.62	7.05	7.71	7.67
AvgQuality	1.62	1.76	1.82	1.77	1.59	1.67	1.82	1.63	1.67	1.76
Avg%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

16-cities Instances										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
AvgStep	8.58	8.11	7.68	8.33	8.13	8.16	8.14	8.28	8.12	8.46
AvgQuality	1.56	2.05	1.67	1.87	1.92	1.65	1.81	1.75	2.09	1.77
Avg%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

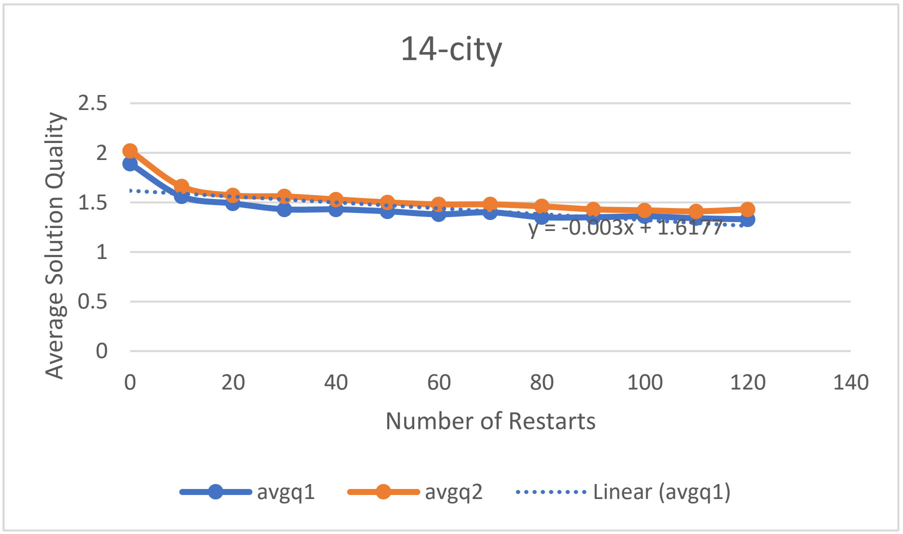
From the data above, we can see that the average number of steps increases while number of cities increases. However, the average solution quality is around 1.5-2.0 for all instances, which is approximate double of the cost of the global best solution. It is surprised that all the matching percentage are 0%, which means that all the local optimal I found do not match the local optimal by A* Search. It might because the neighbour relation I choose is “swapping neighboring nodes”, then the neighbourhood is small, it is hard to meet an exact solution found by A* Search. I have done experiments for small cities instances (eg. 5-city), it will find exact the same solution by A* Search within 100 repetition. It means that this neighbour relation is not fit for large number of cities, maybe “swapping any 2 nodes” is better.

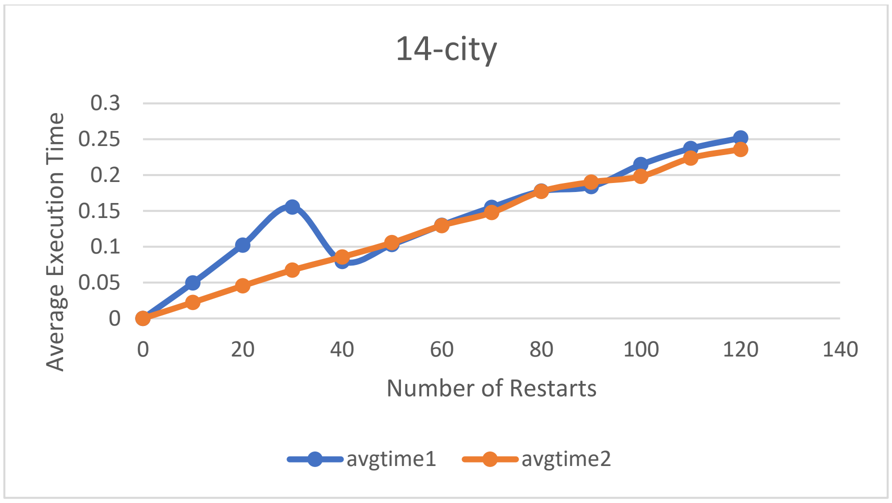
3. **Instance**: 6-1
   6 A 80 76 B 87 38 C 8 14 D 23 71 E 96 8 F 53 56
   Start state: ['A', 'D', 'C', 'E', 'B', 'F', 'A']
   Best neighbour for start state: ['F', 'D', 'C', 'E', 'B', 'A', 'F']
   Current state: ['F', 'D', 'C', 'E', 'B', 'A', 'F']
   Current state cost: 284.24684899273996
   Neighbours: ['D', 'F', 'C', 'E', 'B', 'A', 'D'] (310.479337236481)
   ['F', 'C', 'D', 'E', 'B', 'A', 'F'] (320.48251216691426)
   ['F', 'D', 'E', 'C', 'B', 'A', 'F'] (372.9765446981363)
   ['F', 'D', 'C', 'B', 'E', 'A', 'F'] (309.8253051313809)
   ['F', 'D', 'C', 'E', 'A', 'B', 'F'] (327.65309914282)
   ['A', 'D', 'C', 'E', 'B', 'F', 'A'] (307.75611728454237)
   Best neighbour: ['A', 'D', 'C', 'E', 'B', 'F', 'A']
   Best neighbour cost: 307.75611728454237 (>284.24)
   Strict local optimal: ['F', 'D', 'C', 'E', 'B', 'A', 'F']

4. For hilling climbing with **sideway moves**(100 steps), it did not perform better because the average solution quality is similar to the original hilling climbing (eg. 16-1 instance: avgq = 1.68), and the matching percentage is 0% as well. (You can run TSPLS_test for testing).

   For hilling climbing with **maintaining tabu list**, it did not perform better as well. Since I defined "swap neighboring nodes" as neighbor relation, the range for neighbours would be small, adding a queue with size 100 to maintain neighbours would not help.

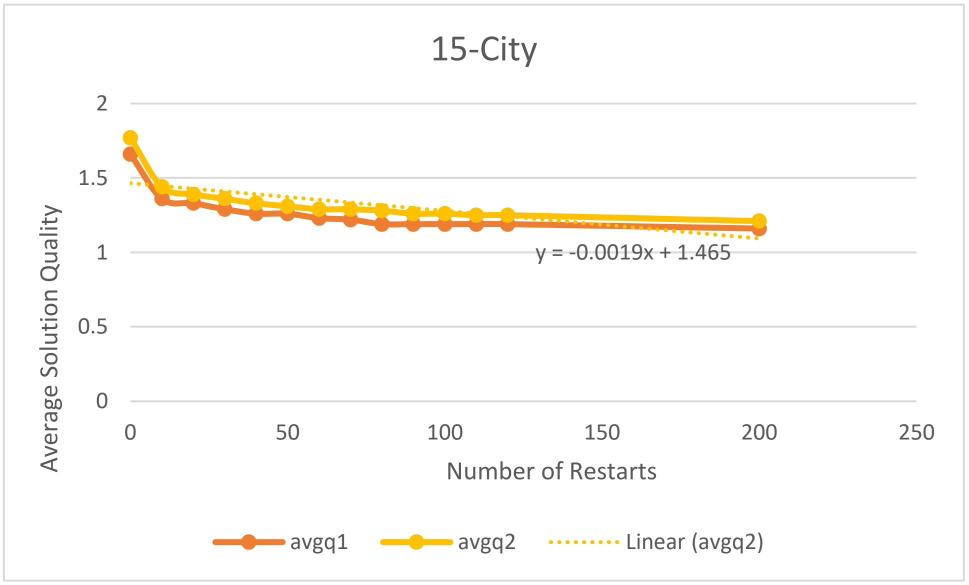
5. See "TSPLC_RS"
   a)

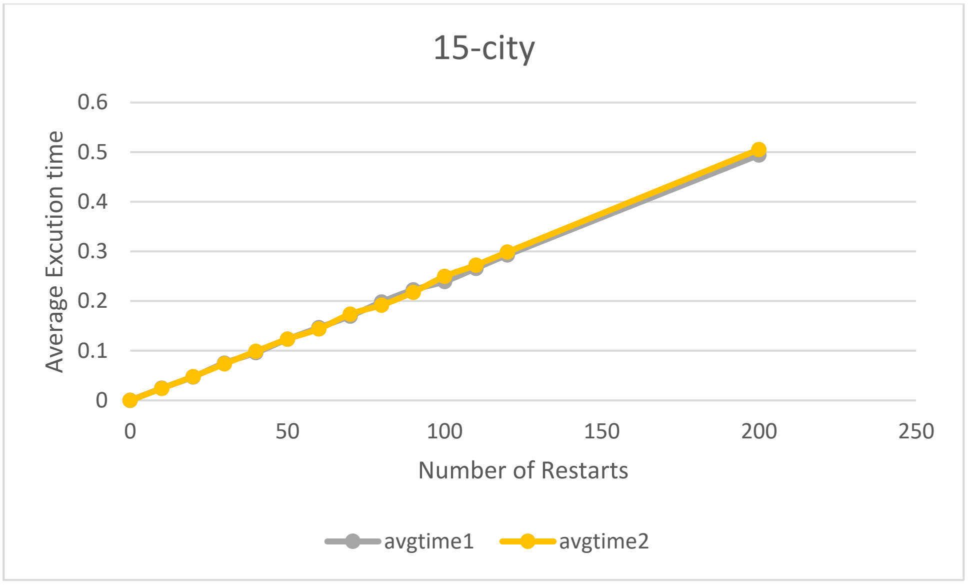




For **14-city**, we could see that the estimated function for the average solution quality is $y = -0.003x + 1.6177$, then when y = 1.01, x = 202, so I expect that it needs more than 200 times to get the quality to be around 1.01. From the graphs, we also see that when number of restarts increases, the average solution quality decreases, but average execution time increases. Thus, we have a trade-off. If we want solutions with less cost, we need more execution time. Overall, for 14-city instances, it is better to take 200 as restart time, the execution time will just take few seconds and it is closed to global optimal.

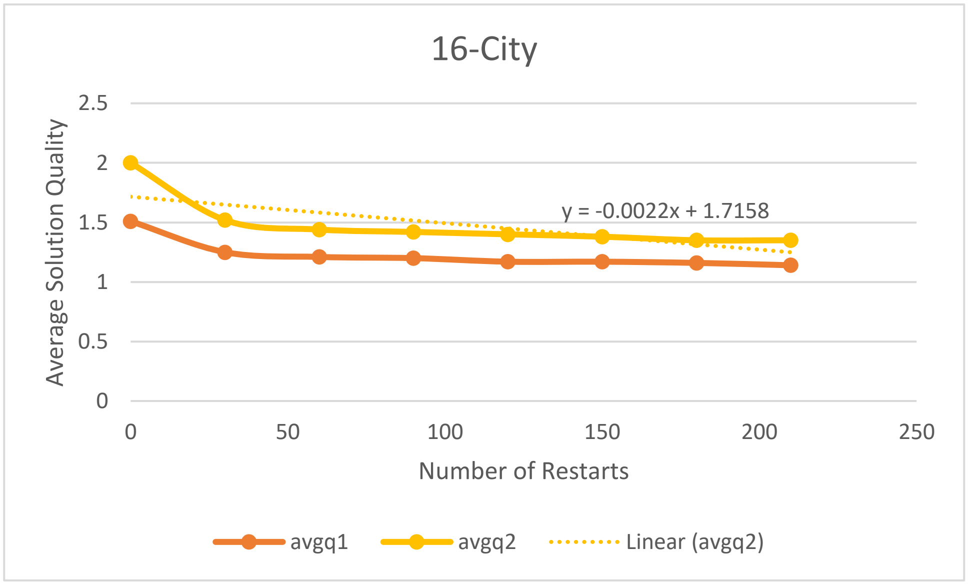
b)

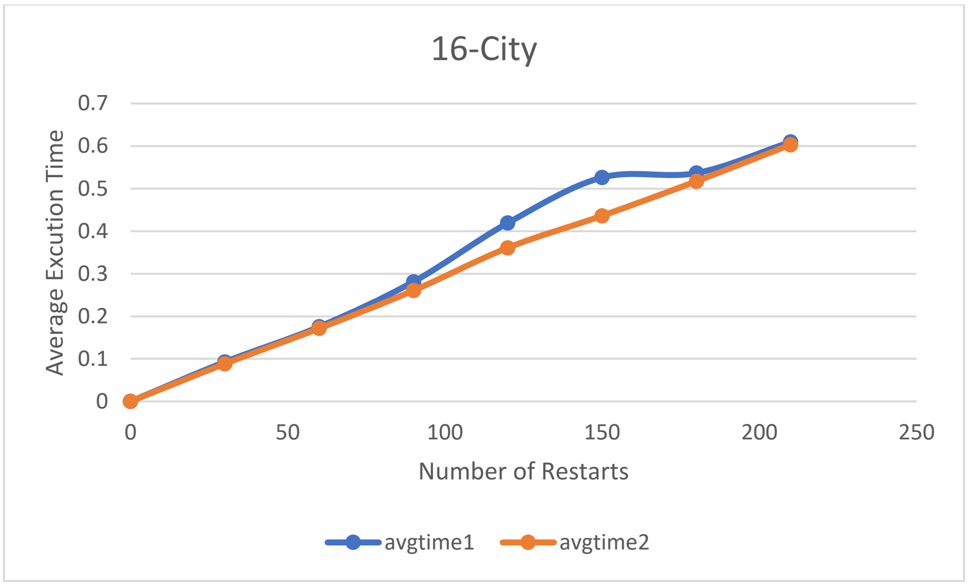




For **15-city**, the estimated function for solution quality is y = -0.0019x + 1.465, so when y = 1.01, x = 239, we might need 239 times of restarts to reach average solution quality of 1.01. The trade off property is similar to 14-city, better solution for longer time. Overall, we might pick 250 as restarts because its average quality is around 1, and the estimation time is not long.

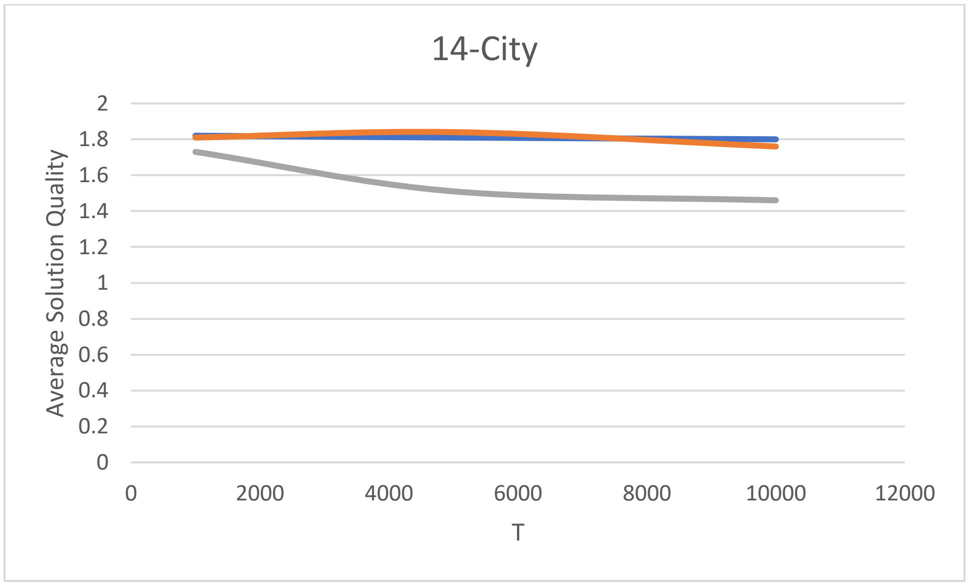
c)

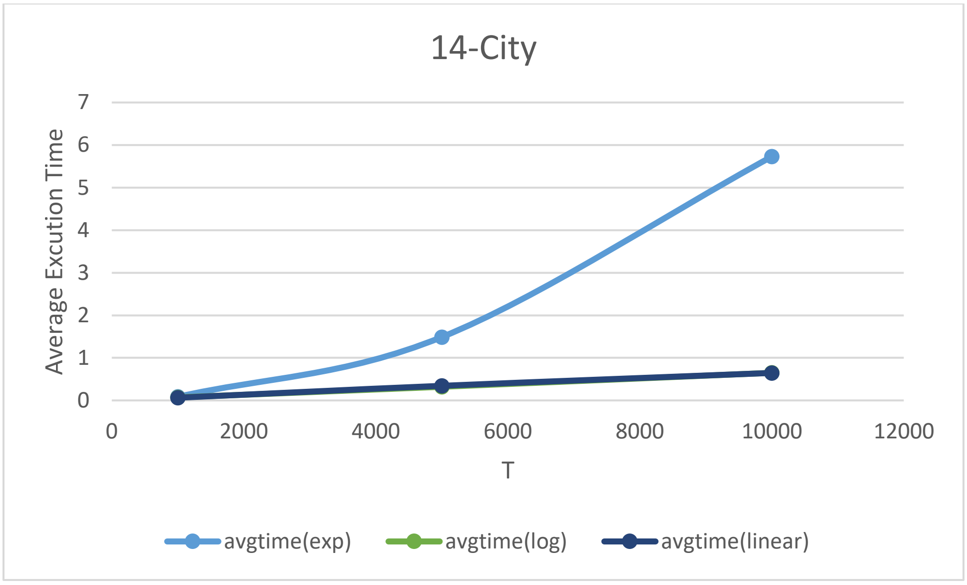




For **16-city**, the estimated function for solution quality is y = -0.0022x + 1.7158, so when y = 1.01, x = 321, we might need 321 times of restarts to reach average solution quality of 1.01. The trade off property is similar to 14-city, better solution for longer time. Overall, we might pick 300 as restarts because its average quality is around 1, and the estimation time is not long.

6. The code is in “TSPLS_SA”.
   a)





Obviously, for **14-City** instances, using log annealing schedule is much better. The average quality is small comparing to other twos. There is no trade-off between annealing schedule and execution time: for same T, using the exponential takes more time, however, using log and linear annealing schedule take less time but provide better solutions.

b)

T	avgq(linear)	avgq(exp)	avgq(log)	avgt(linear)	avgt(exp)	avgt(log)
10000	1.72	1.81	1.31	0.6826	4.1239	0.694

For **15-city** instances, my conclusion is similar to 14-city. Using log annealing schedule is better and there is no trade off.

c)

T	avgq(linear)	avgq(exp)	avgq(log)	avgt(linear)	avgt(exp)	avgt(log)
10000	1.74	1.87	1.24	0.9134	4.1023	0.7554

Same conclusion for **16-city** instances. Log annealing schedule is the best and there is no trade off.

7. 36-city

Local Search:

['AI', 'AH', 'L', 'AB', 'U', 'G', 'Y', 'I', 'AF', 'V', 'Z', 'AG', 'R', 'D', 'J', 'P', 'O', 'E', 'B', 'AJ', 'H', 'S', 'AE', 'M', 'F', 'AC', 'AD', 'Q', 'N', 'T', 'X', 'A', 'K', 'W', 'C', 'AA', 'AI']

1151.8375860604704

Local Search with Restarts:

['AB', 'A', 'J', 'O', 'S', 'G', 'L', 'B', 'AJ', 'H', 'Z', 'K', 'W', 'N', 'D', 'AH', 'I', 'T', 'R', 'AE', 'U', 'P', 'AD', 'E', 'AC', 'F', 'M', 'AG', 'C', 'AA', 'AF', 'AI', 'Y', 'X', 'V', 'Q', 'AB']

1007.1602647439032

Local Search with Log Annealing Schedule:

['V', 'D', 'AB', 'A', 'Q', 'J', 'AC', 'L', 'Y', 'I', 'AI', 'AA', 'R', 'AG', 'F', 'Z', 'K', 'P', 'E', 'C', 'AD', 'O', 'B', 'G', 'M', 'AE', 'AF', 'AH', 'T', 'X', 'N', 'W', 'U', 'S', 'H', 'AJ', 'V']

998.5630202533708

Conclusion: using log annealing schedule for 36-city local search is the most efficient method.