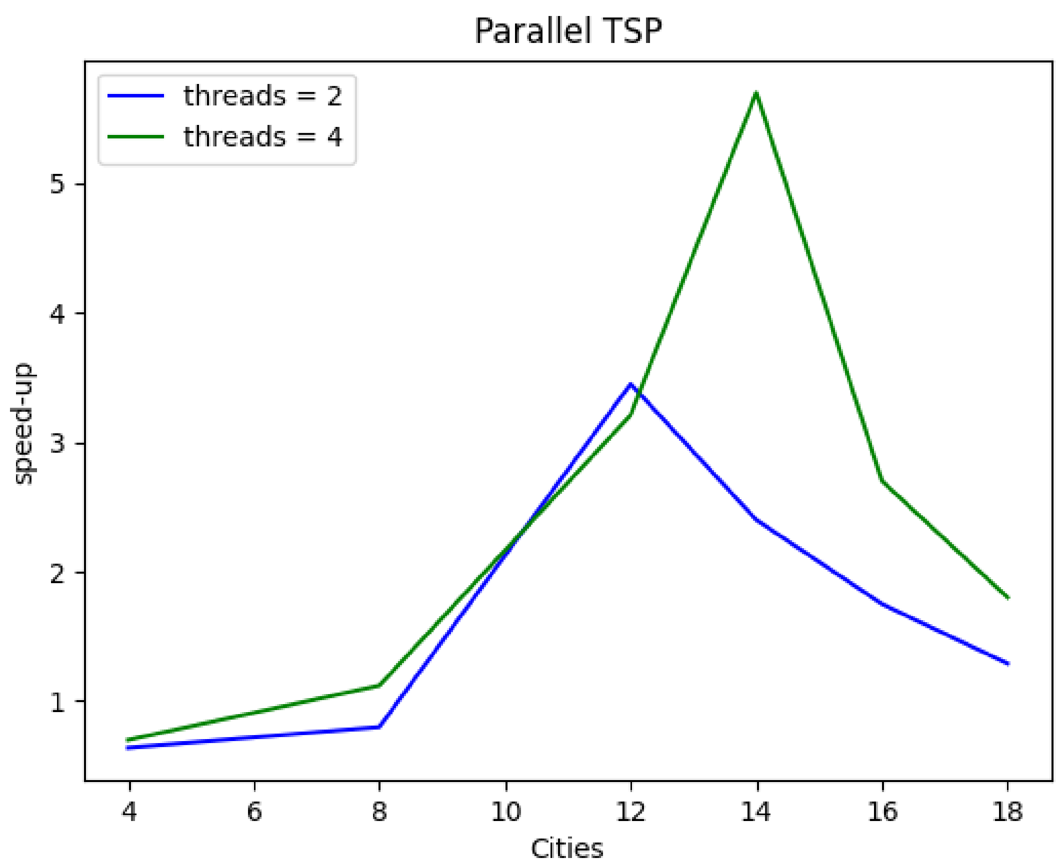
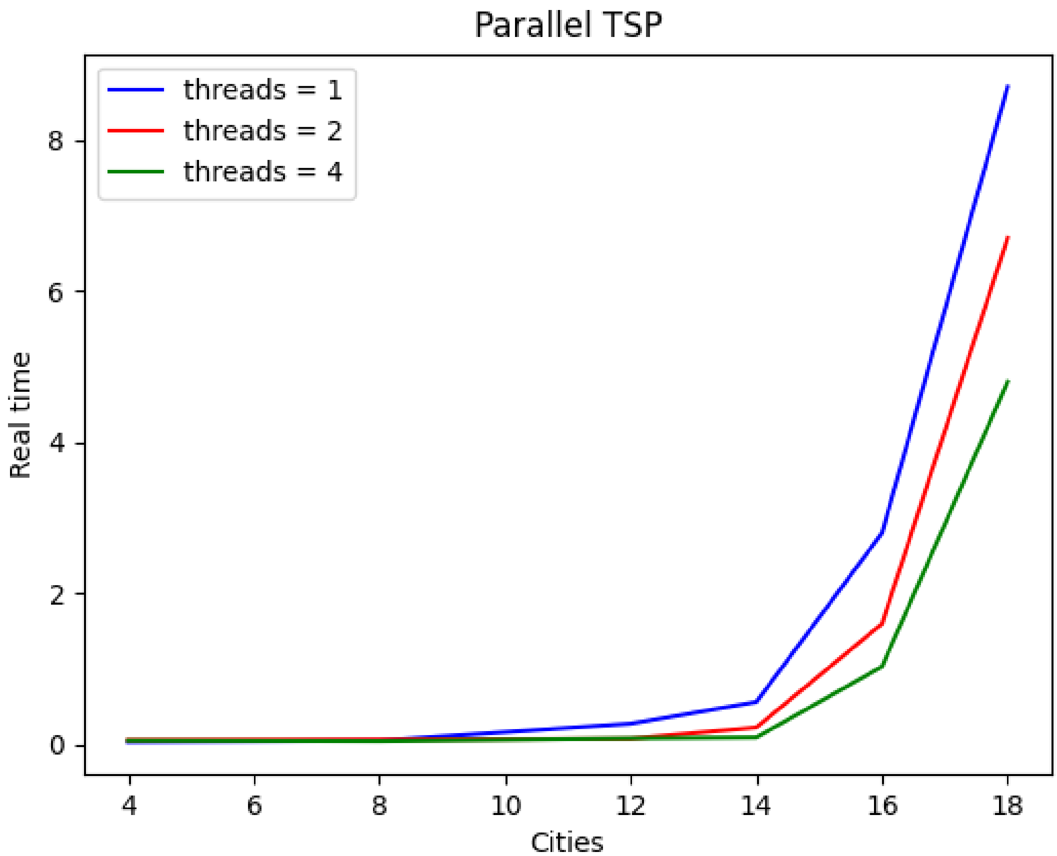
There are two implementations for this problem: One is trying all n! Combinations. The other one is with bit-masking with n*2^n complexity which is more efficient. I have implemented both but this analysis is being done with the bit-masking algorithm.

Experiment1:

Cities	t=1	t=2	t=2 (speed-up)	t=4	t=4(speed-up)
4	0.041	0.064	0.64	0.058	0.7
8	0.056	0.07	0.8	0.05	1.12
12	0.28	0.081	3.45	0.087	3.21
14	0.566	0.23	2.4	0.098	5.7
16	2.8	1.6	1.75	1.034	2.7
18	8.7	6.7	1.29	4.8	1.8

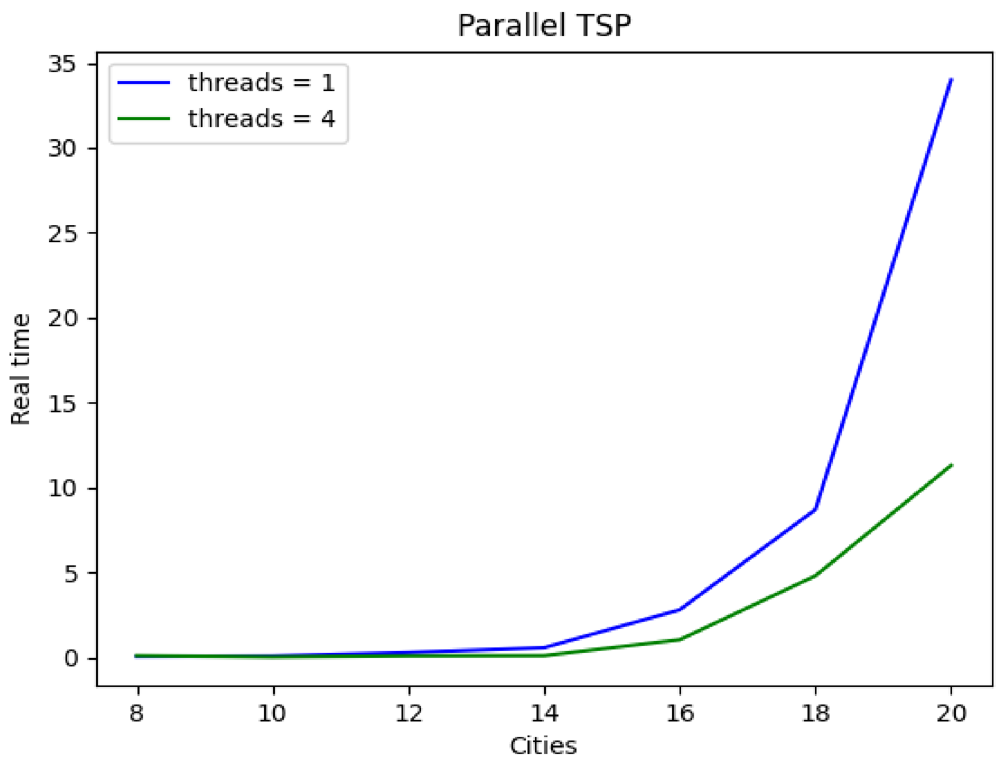


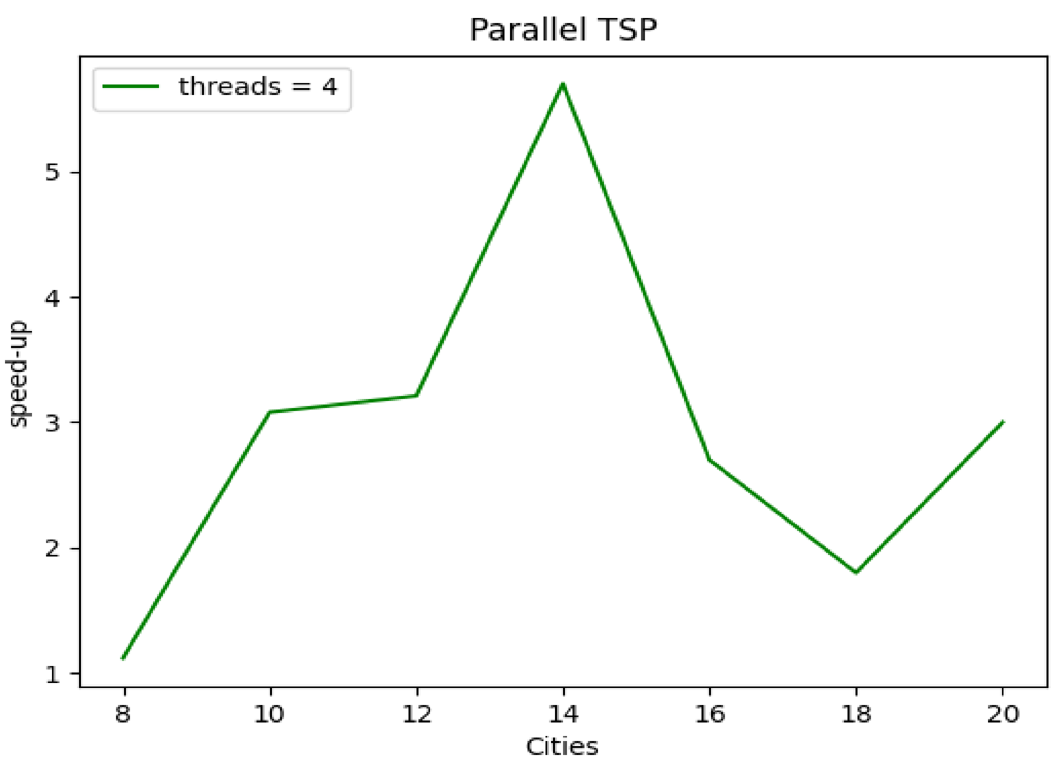


Experiment2:

Here we increase cities number by 2 until we see a stable speed-up of >=2 with threads=4.

Cities	t=1	t=4	Speed-up
8	0.056	0.098	1.12
10	0.077	0.025	3.08
12	0.28	0.087	3.21
14	0.566	0.098	5.7
16	2.8	1.034	2.7
18	8.7	4.8	1.8
20	34	11.3	3.0


Parallel TSP
threads = 1
threads = 4
Real time
Cities
0
5
10
15
20
25
30
35
8
10
12
14
16
18
20


Parallel TSP
threads = 4
speed-up
Cities
1
2
3
4
5
8
10
12
14
16
18
20

Observations:

- Since the time is real time and not wall-time, we see some irregularities over a few cities. When the actual algorithm time takes more effect in the algorithm than the thread creation and sync times, this effect reduces.
- Parallel TSP with threads is performing better with threads 4 than 2 and the actual speed-up is greater than 1 for both threads=2 and threads=4
- This algo runs in O(n*2^n) complexity. The n! algo doesn't even run properly when it crosses 14. tsmoptimal seems to be suffering as well.
- If we see the threads=1,2,4 with increasing cities we see that Real-time(1) > Real-time(2) > Real-time(4)