The Traveling Salesman Problem: Adaptation of 2-Opt Local Optimization in Comparison to Branch & Bound Techniques

For project spec – see
`http://axon.cs.byu.edu/~martinez/classes/312/Projects/GroupTSP.pdf`

Adara Christensen, Ammon Christensen, Matthew Christensen

C S 312 – Algorithm Design & Analysis, Dr. Martinez Tony R, Brigham Young University, Winter 2021

Abstract. The 2-opt and 3-opt algorithms are heuristic methods that involve local search algorithms to solve the traveling salesperson problem. Our algorithms are a combination of both 2-opt and 3-opt local search problems. This paper will discuss and compare the benefits of 2-opt and 3-opt algorithms in comparison to performance on branch and bound and greedy tsp algorithms as well as an analysis of empirical data from both algorithms.

Keywords: Traveling Salesman Problem, Algorithm Design and Analysis, 2-opt, 3-opt, Local Search

1 Introduction

1.1 History

The traveling salesperson problem (TSP) was initially defined in the early 1800s by two men W.R. Hamiliton and Thomas Kirkman, solving puzzles based on the Hamiltonian cycle. Then the concept was transformed in the 1920s by Karl Menger. He called it the “messenger problem” [2]. Discovering the method of merely traveling to the closest city from one's current location did not result in the lowest travel distance. This traveling salesperson problem has since then received much attention from mathematicians and computer scientists alike.

1.2 **Background**

The TSP algorithm is defined as NP-Hard. This means that the complexity will always increase as the number of destinations is increased and there is no “quick” solution [1].

Due to the difficulty in solving a complete solution when considering both time efficiency and lowest/shortest cost there are many algorithms that have been created.

Some of the algorithms that have been discovered and that will be discussed and analyzed within this paper are the Greedy Algorithm, Branch and Bound Algorithm, 2-Opt Algorithm, and 3-Opt Algorithm. The greedy algorithm is one of the most simple in objective. This algorithm finds the shortest path always and travels there. However as discovered by Karl Menger this solution is not always the most optimal. The branch and Bound algorithm works to find the most optimal solution through traveling down branch networks and testing all the possible solutions. This algorithm may find the most optimal path but the time complexity to receive this solution is slow. Then there are the 2-Opt and 3-Opt algorithms which begin with a path and then work to improve it by swapping two to three paths at a time and replacing if the cost decreases. This method is faster than the Branch and Bound algorithm and also is able to improve upon the greedy algorithms lowest path.

2 Algorithm Analysis

We will now walk through the greedy, branch and bound, and 2-opt/3-opt algorithms to conduct theoretical and empirical analysis. This includes the advantages and disadvantages of the given approach. Additionally, this includes a time and space complexity analysis.

2.1 **Greedy Algorithm**

The concept of a greedy algorithm is for the choice with the most immediate benefits to be used first. For our greedy TSP algorithm, the route always adds on the neighboring city that has the lowest cost. Our algorithm begins with a random starting city and follows the pattern of continually choosing the lowest path. If the final route does not connect to every city the route is cleared and a new starting city is chosen. This method will continue until it finds a complete path.

Our greedy TSP algorithm has an average time complexity of O(n^2). This algorithm's main big-O effectors are a for loop that goes through all cities and finds the lowest cost neighboring city with a complexity time of O(n) . This city is then added to an array in O(1). If the length of the array matches the number of cities to be traveled to, the function exits the while loop and returns a solution. Otherwise it returns to the beginning of the while. This while loop since it is often traveled to at least n times runs

in O(n) time. Making the total running time for the greedy algorithm O(n^2). The space complexity is simply O(n) because we never store more than 3 copies of the route or given solution. Thus O(n) + O(n) + O(n) = O(3n) = O(n).

2.2 Branch and Bound Algorithm

The Branch and Bound Algorithm implementation allows for a better solution than that of the greedy algorithm. This method finds the optimal solution by traveling through the many possible paths that can be taken. Path options that exceed the value of the initial BSSF are filtered out and pruned off the list of possible solutions.

Due to the checking of many potential solutions the time complexity of Branch and Bound is rather high at O(b^n (n+1)!). This big-O time complexity is a worst-case scenario and will not reach this speed very often. However due to the slower speed, cut-off times are often provided to limit the time taken to receive a solution. Space complexity is a similar story, worst case scenario compounds to O(!n), while the average case tends to be closer to O(p * n^2). While the cutoff prevents total optimal solutions, it provides for a better solution than that of the greedy algorithm.

Additionally, this is the algorithm we use to provide 2-opt and 3-opt with their initial route (which will be covered further in the preceding section).

Finally, if more detail is desired concerning the implementation or discussion of the Branch and Bound Algorithm please refer to previous publications (see lab 5 reports).

2.3 2-opt and 3-opt Algorithm

The 2-opt Algorithm is implemented by taking a given valid route and looking to improve it by comparing edge costs when two edges are interchanged. Simply put, we disconnect the path at a given point and see if by switching the order of two nodes and reconnecting the path to see if we can find better improvement (as seen in fig. 1 below).

Fig. 1. Pseudo code for 2-opt Algorithm

```
While currentPath is better than previousPath:
    For each edge:
        For every other edge:
            Swap edge destinations
            If swap results in better path:
                Keep changes
```

This provides a relatively simple way to improve a given solution in polynomial time. The time complexity of 2-opt is O(n^2). This is because the algorithm must compare every edge in the path to every other edge in the path and may do this multiple times if improvements continue to be found. The 3-opt Algorithm does the same thing but switches all combinations of 3 edges and runs in O(n^3) time complexity. Additionally, both algorithms have a space complexity of O(n) because only 3 solutions/routes are ever stored - O(n) + O(n) + O(n) = O(3n) = O(n).

In theory, the 3-opt version should result in a better path, since more total path combinations are considered. Our empirical results showed that when the starting path is derived from a greedy algorithm, there is rarely a difference between the 2-opt and the 3-opt algorithm. However, when the starting route is chosen at random, there is a significant improvement when using 3-opt, at the cost of greater time complexity. In all cases, the empirical time complexity was as expected, with variability in the improvement of the path travelled. When compared to the branch and bound algorithm, this provides a less optimal solution, but finishes in a much faster time. Our empirical analysis also found that as the city count increased, the improvements upon Greedy with the 2-opt algorithm also increased (see table in fig. 2 below).

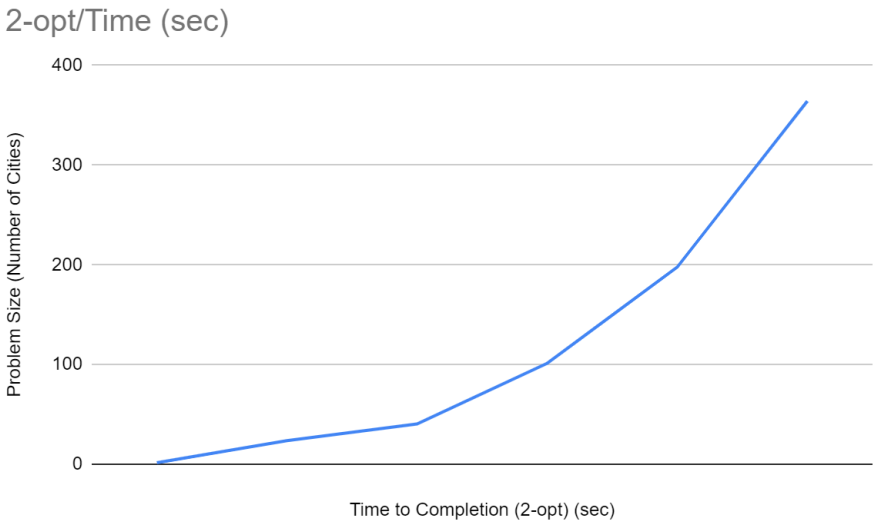
Fig. 2. Empirical Data for Random, Greed, Branch and Bound, and 2-opt Algorithms

	Random		Greedy			Branch and Bound			2-opt		
Cities	Cost	Time (sec)	Cost	Time (sec)	% of Random	Cost	Time (sec)	% of Greedy	Cost	Time (sec)	% of Greedy
15	21105	0.001	12925	0.000999	61.24%	9380	18.301747	72.57%	11773	0.004001	91.09%
18	21254	0.000898	12447	0.001001	58.56%	9538	350.210651	76.63%	11402	0.01499	91.60%
30	42244	0.083951	15858	0.002989	37.54%	TB	TB	TB	14274	0.032918	90.01%
60	82143	6.98557	29465	0.04675	35.87%	TB	TB	TB	26283	0.29259	89.20%
100	TB	TB	41188	0.057197	N/A	TB	TB	TB	35630	1.407514	86.51%
200	TB	TB	59334	0.117628	N/A	TB	TB	TB	51851	23.595704	87.39%
300	TB	TB	75357	0.251833	N/A	TB	TB	TB	66691	40.390397	88.50%
400	TB	TB	88899	0.466197	N/A	TB	TB	TB	74487	101.162358	83.79%
500	TB	TB	101666	0.738771	N/A	TB	TB	TB	82735	197.685956	81.38%
600	TB	TB	117996	1.081589	N/A	TB	TB	TB	96294	364.306545	81.61%
690	TB	TB	124899	3.051987	N/A	TB	TB	TB	100381	545.009223	80.37%
700	TB	TB	129039	1.491016	N/A	TB	TB	TB	TB	TB	TB

The efficiency of the 2-opt algorithm, compared to the cost of the greedy results (2-opt cost/greedy cost), increases because of the likelihood that an improvement in an early iteration will make future changes and iterations possible. This means that our rate of improvement only increased with the size of the problem, as the gains of future iterations compound with each improvement. This can be seen in the table of empirical analysis figures as seen above. We start to see steady improvement as the problem size grows past 100 cities. However, because early changes create more possibilities for future iterations, which also create improvements that enable more

iterations and thus start a positive feedback loop. This causes the time to completion to increase also.

Fig. 3. Time to Completion for 2-opt Algorithm



Graphing the time to completion for 2-opt (see figure 3 above), we can see that because of the positive feedback loop, the time scales on an exponential level. Thus, the sweet spot for this algorithm is for relatively large scale problems (about 100-1000 cities for consumer grade hardware) where the 2-opt algorithm has a larger search base to find improvements to the cost of each tour significantly while also maintaining a manageable time to completion.

It should also be noted that the results for each cell (all 4 algorithms) are the average of 5 district runs, each with different random seeds for that number of cities.

2.4 **Test Cases**

In conclusion, the 2-opt algorithm offers a comparable solution to a greedy approach to the Traveling Salesperson problem for small scale problems, both in time/space complexity and time to completion, with around a ~10% improvement for problems below a size of 10. However, with larger problems a bigger improvement can be seen while also dramatically increasing the time to completion.

Thus, the cons of the 2-opt algorithm are little improvement over a greedy approach for small problem sizes (less than 100 cities) in addition to not being able to handle incredibly large problem sizes of 1000+ cities due to time complexity. However, the major pro to this approach is being able to compute a significant and important range of problems (100-1000 cities) in a very reasonable time (less than 10 minutes) with significant improvements over a greedy approach (up to 20% improvement of cost).

3 Future Work

In terms of future work, we also investigated the possibility of using a 3-opt algorithm approach in addition to 2-opt. With the 3-opt algorithm, we investigate a larger search space, at the cost of having a larger time complexity of O(n^3). This is because when we want to reconnect three different edges, there is more than one way to reconnect them. As we disconnect 3 edges from the route there are 7 possible new ways to reconnect the route (and we try all of them to find the most optimal path). As the number of edges increases, the number of possibilities increases as well. Thus, a k-opt algorithm will always have a time complexity of O(n^k). We anticipate that this may lead to lower cost paths, but this is not guaranteed.

Improvements to be found may be very small or nonexistent, and as the complexity increases, we should expect to approach the ideal path. This is especially prominent when using a greedy solution for the initial route. In our initial empirical testing, we did not find any significant difference between 2-opt and 3-opt when the size of the problem was below 50 cities. This leads us to believe that in order to find significant differences, we would need to run tests on increasingly larger problem sizes. This might result in testing taking more than a reasonable amount of time on consumer grade hardware. While this may produce a better solution, the time and computation required for such marginal improvements may not be reasonable. However, while using a random solution for the initial route we saw noteworthy improvements between 2-opt and 3-opt. This is because of how optimized the cost of routes are for greedy solutions, which lead to little room for improvement between 2- and 3-opt approaches. However, this is not scalable, because on an asymmetric problem with some infinite distances the random algorithm is unable to find a solution on problem sizes over 100 cities.

4 Conclusion

The traveling salesperson problem has many potential solutions depending on the need for optimization and time efficiency. Due to our focus on an optimal time solution with improved cost path for large scale problems (100-1000), we found the 2-opt algorithm was the most beneficial. This algorithm provided significant improvements from greedy, with only modest increases in time complexity, both theoretically and empirically. Furthermore, the space complexity remains the same, and the 2-opt algorithm is handling problem sizes well into the hundreds. Our algorithm can also be

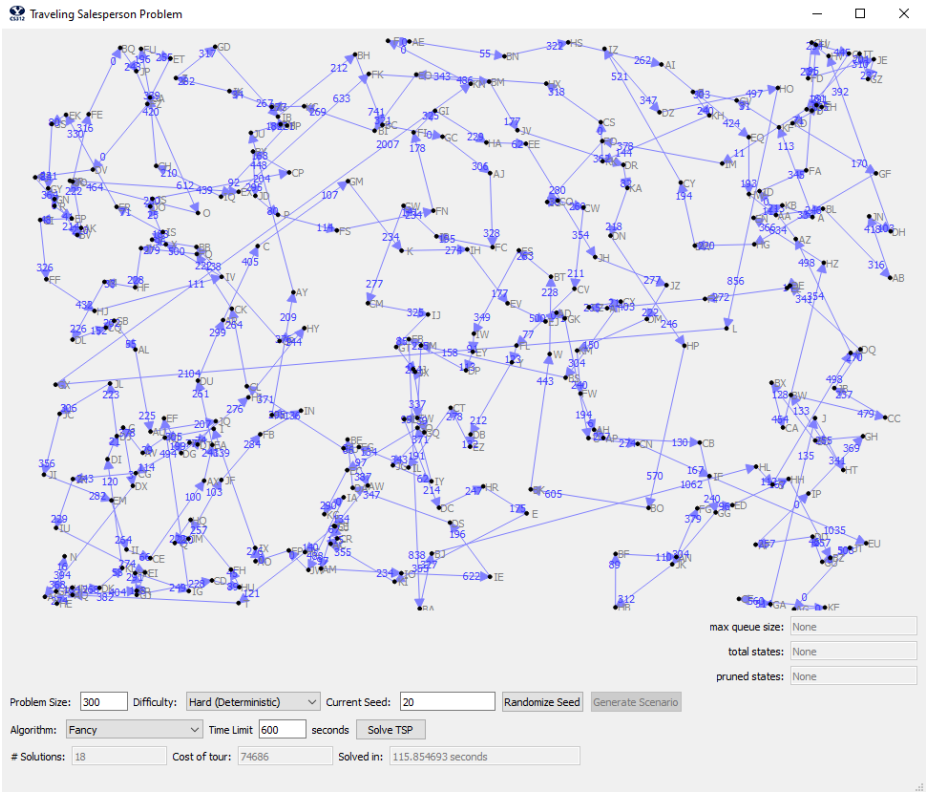
used well beyond that, provided it is given enough time to find a solution. If a more optimal solution is required, more sophisticated methods may be used, but our solution is relatively low-cost both in time and space complexity and provides a solution that is very near optimum.

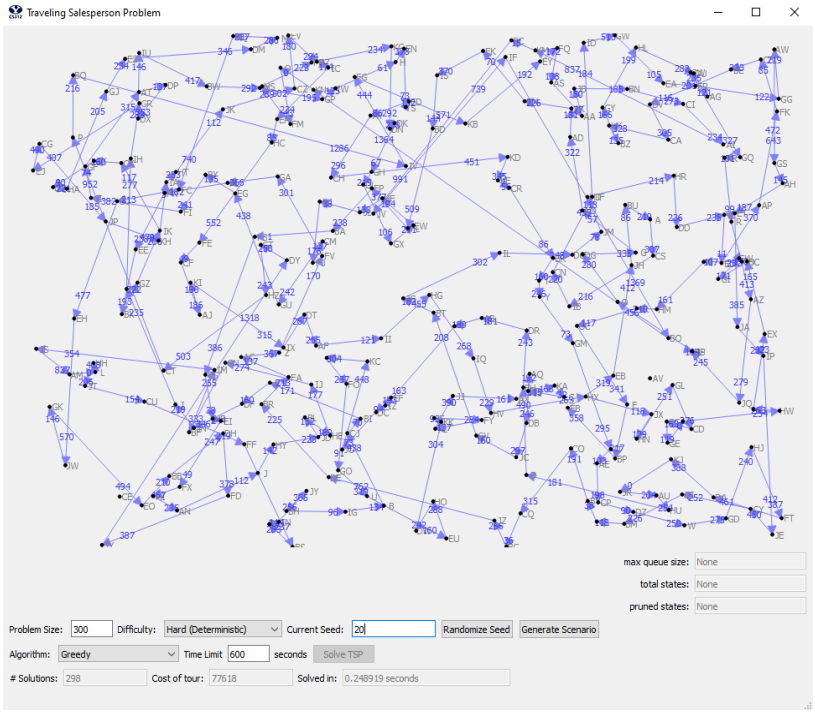
5 References

GitHub Repository: https://github.com/mrchristensen/TravelingSalesperson

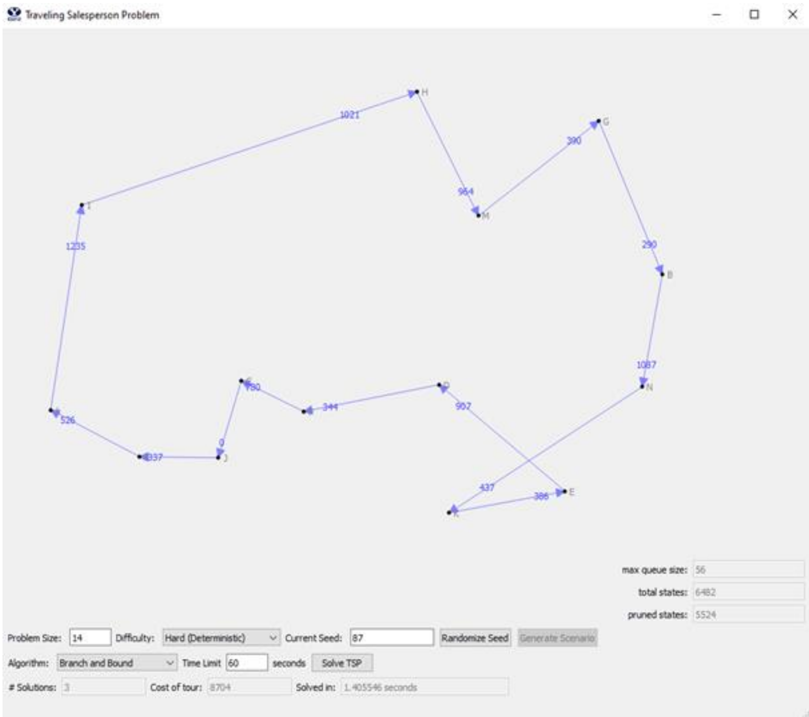
1. Ma, S. (n.d.). Understanding The Travelling Salesman Problem (TSP). Retrieved from https://blog.routific.com/travelling-salesman-problem#:~:text=The Travelling Salesman Problem
2. Home. (n.d.). Retrieved from https://www.theorsociety.com/about-or/or-methods/heuristics/a-brief-history-of-the-travelling-salesman-problem/

6 Screenshot Appendix

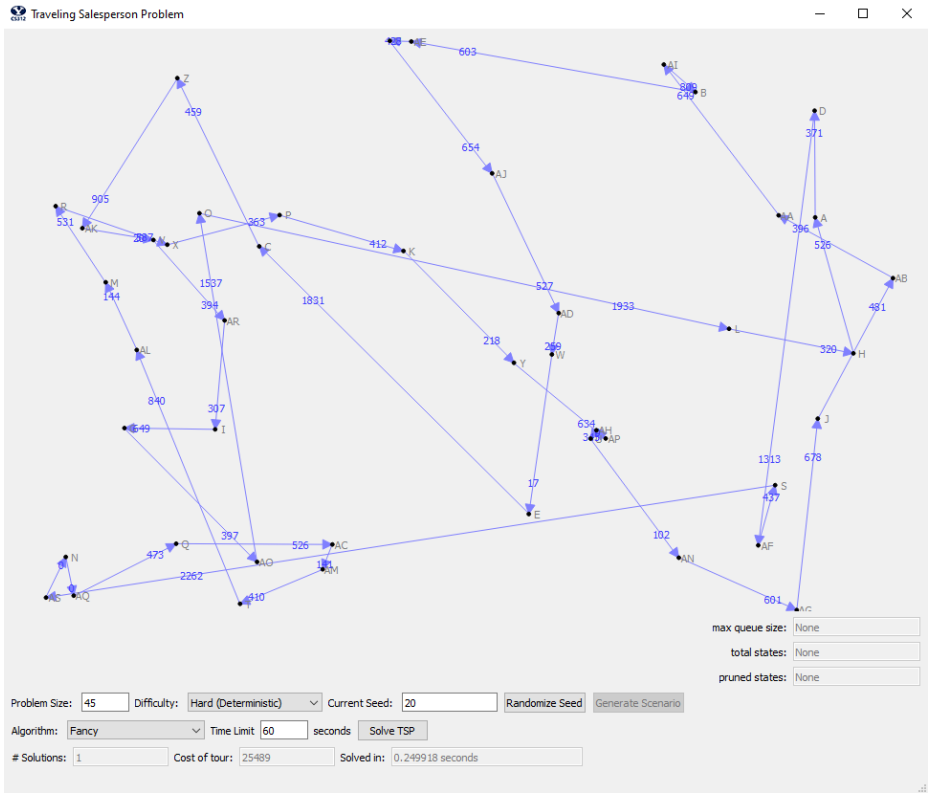



Traveling Salesperson Problem
max queue size:
None
total states:
None
pruned states:
None
Problem Size:
300
Difficulty:
Hard (Deterministic)
Current Seed:
20
Randomize Seed
Generate Scenario
Algorithm:
Greedy
Time Limit
600
seconds
Solve TSP
Solutions:
298
Cost of tour:
77618
Solved in:
0.248919 seconds

p. 9, 2021.


Traveling Salesperson Problem
max queue size: 56
total states: 6482
pruned states: 5524
Problem Size: 14
Difficulty: Hard (Deterministic)
Current Seed: 87
Randomize Seed
Generate Scenario
Algorithm: Branch and Bound
Time Limit 60 seconds
Solve TSP
Solutions: 3
Cost of tour: 8704
Solved in: 1.405546 seconds

p. 10, 2021.


Traveling Salesperson Problem
603
459
654
905
531
363
412
1537
144
394
1831
527
1933
840
307
218
634
1313
678
17
102
397
526
473
2262
410
601
371
396
526
481
320
max queue size: None
total states: None
pruned states: None
Problem Size: 45
Difficulty: Hard (Deterministic)
Current Seed: 20
Randomize Seed
Generate Scenario
Algorithm: Fancy
Time Limit 60 seconds
Solve TSP
Solutions: 1
Cost of tour: 25489
Solved in: 0.249918 seconds