**Travelling Salesperson Problem ‐ Search with BFS and DFS**

Andrew Nguyen Vo

Computer Engineering and Computer Science

Speed School of Engineering

University of Louisville, USA

[awnguy01@louisville.edu](mailto:awnguy01@louisville.edu)

1. **Introduction** (What did you do in this project and why?)

The problem presented in this project was a modified version of the Traveling Salesperson Problem in which I needed to develop a program to find the shortest weighted distance from a starting node to a target node. The purpose was to develop techniques for traversing a graph, which is a common problem faced by artificial intelligence. This project demonstrated the limitations of traditional algorithms and emphasized the necessity of novel approaches to solving problems.

1. **Approach** (Describe algorithm you are using for this project)

*Solution produced using Python 3.7.4 on Windows 10 operating system*

This project required that two algorithms be implemented in the program: breadth first search (BFS) and depth first search (DFS). However, being search algorithms, both needed some modification in order to keep a record of routes. Both algorithms were implemented iteratively, which means that each algorithm had a “stateful” variable outside the scope of their while loops where the main searching was contained. The resulting modified BFS and DFS algorithms were implemented similarly with the key difference being that the BFS utilized a queue structure to analyze the graph whereas the DFS used a stack. The effect of this was so that the BFS algorithm would explore all the nodes closer to the root before moving onto the next level whereas the DFS algorithm would explore each path to completion before exploring another path. Since both algorithms were originally designed to discover the minimum spanning tree only for unweighted graphs, the modified algorithms could not take advantage of skipping nodes already explored in previous loops but rather had to accumulate every possible path and calculate the shortest weighted path from the final list. In order to generate the routes, for each algorithm, the program used the currently popped or dequeued node and determined which routes in the final list contained the current node and spliced those to a temporary list of “affected routes”. For each possible destination node for the currently popped or dequeued node, the affected routes would be appended with that destination and re-added to the list of final routes. That destination would then be appended to the queue or stack for future looping. The process continued until the queue or stack was empty, resulting in all the possible generated paths.

1. **Results** (How well did the algorithm perform?)
   1. **Data** (Describe the data you used.)

The input data was a standard TSP file consisting of 11 possible nodes, which was parsed by the program. A predefined set of connected nodes was provided by the problem. Each node was connected only in a single direction to prevent backwards traversal (e.g. the program can traverse from Node 1 to Node 2 but not the other way around).

* 1. **Results** (Numerical results and any figures or tables.)

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Route | Distance | Time |
| Breadth First Search | 1 3 5 7 9 11 | 57.96716475441191 | 00:00:00.001 |
| Depth First Search | 1 3 5 7 9 11 | 57.96716475441191 | 00:00:00.001 |

The list of possible complete paths was not sufficiently large enough to determine average real runtimes, which were within the nanosecond range. Since both algorithms were implemented similarly, they share similar time complexities, being that their worst case time was O(n3). However, due to the looping being done on different lists (one of which is the list of destination nodes, which is small and predefined), the time complexity of the modified algorithms is closer to O(n2).

1. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

Due to the modifications in the algorithms, some of the advantages of the breadth first search and depth first search could not be applied. Since all routes had to be explored to guarantee the shortest weighted route, the algorithms could not skip nodes previously visited in a previous iteration. This significantly increased the number of traversals needed. However, because only certain nodes were connected and not all nodes needed to be traversed before reaching the final node, the number of permutations was significantly less than in the previous Hamiltonian variant of this problem, and so the algorithms were able to calculate the shortest path in a reasonable time. As this problem is scaled to include more node connections, the runtime will exponentially increase.

1. **References** (If you used any sources in addition to lectures please include them here.)

<https://www.geeksforgeeks.org/iterative-depth-first-traversal/>

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>