Experimental Analysis for CS720 Course Project

(2019CSM1008, 2019CSM1010, 2019CSm1018)

May 2020

Instructed by: Dr. Venkata M Viswanath Gunturi

1 Introduction

Here we analyze the result of our own implemented "MS(WBS)" algorithm. MS(WBS) algorithm is used for finding the Most Navigable Path(MNP) between a given pair of source and destination. We also compare MS(WBS) with Dijkstra with respect to different parameters. Here we mainly focus on four parameters to compare with Dijkstra:

1. Navigability score with respect to different overhead.
2. Runtime with respect to different overhead.
3. Runtime with respect to different path cost.
4. Score per unit distance.

1.1 Runtime Environment:

We do our all experiment in a 64 bit server with 32 core, max clock speed 2.80 GHz, 55 GB RAM.

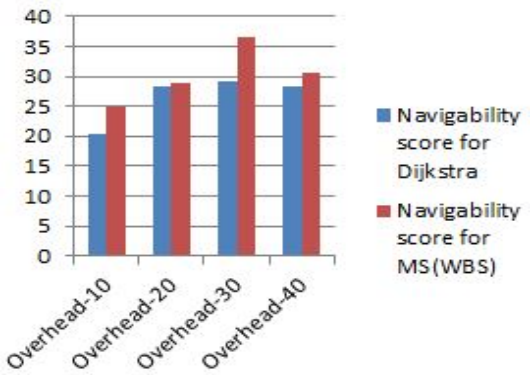
1.2 Implementation Details:

Here we take two files as input. One is a node file containing all node IDs of the graph, another is an edge file containing all edge details(source node, destination node, cost). Then we generate a non-zero navigability score for 40% edges. We take overhead as 10%, 20%, 30%, and 40% of the cost of the shortest path between the corresponding pair of source and destination. In order to find the runtime of MS(WBS) algorithm for different length of path, we use average runtime for ten paths. As an example to find runtime of path length 11-20, we use a bucket of size 10 containing runtime for path length 11-20, then we take the average of those runtimes as runtime for path length 11-20.

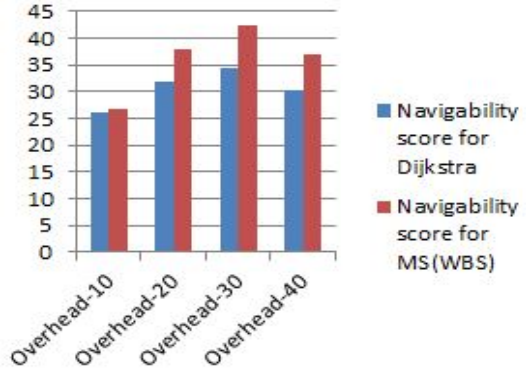
Here we use two datasets to analyze the performance of MS(WBS) algorithm with respect to Dijkstra. Dataset1 contains 23839 nodes and 51794 directed edges, while dataset2 contains 65902 nodes and 144410 directed edges. In both cases we assign a non-zero navigability score between 1-15 for 40% of edges and zero navigability score for rest of the edges.

2 Comparison with Dijkstra:

Main goal of this algorithm is to maximize the navigability score of a route within a limited increase of distance.



(a) Dataset1

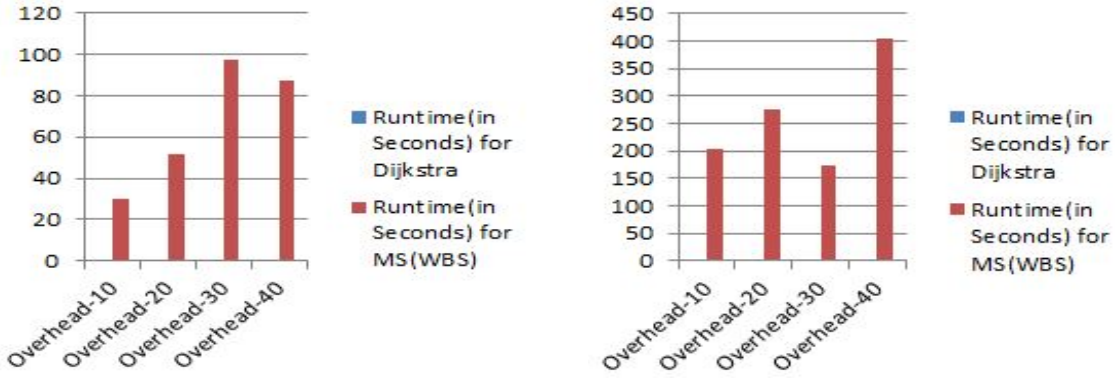


(b) Dataset2

Figure 1: Comparison of navigability score achieved by both algorithm for different cost overhead

Here in Figure 1, we plot the average navigability score achieved by Dijkstra and MS(WBS) algorithm against different distance overhead values. We compute the navigability score for path length 31-40. We take such 10 paths and take the average of their navigability score. It is clear from Figure 1 that MS(WBS) always achieves a higher navigability score than Dijkstra.

Now it is obvious that MS(WBS) takes more time to calculate the most navigable path because we use the same Dijkstra algorithm in WBS algorithm in order to find the best possible replacement of a given segment. So it is unfair to compare MS(WBS) with Dijkstra with respect to runtime. We plot the comparison in Figure 2 and it gives us the result as expected. Average runtime for Dijkstra was 0.23, 0.225, 0.224 and 0.215 seconds respectively for 10%, 20%, 30% and 40% overhead for dataset1 and 0.63, 0.66, 0.66 and 0.65 seconds respectively for 10%, 20%, 30% and 40% overhead for dataset2. For MS(WBS) it was 30.26, 51.49, 97.02, 87.16 seconds and 204.61, 275.97, 173.5, 405.14 seconds respectively for dataset1 and dataset2.



(a) Dataset1 (b) Dataset2

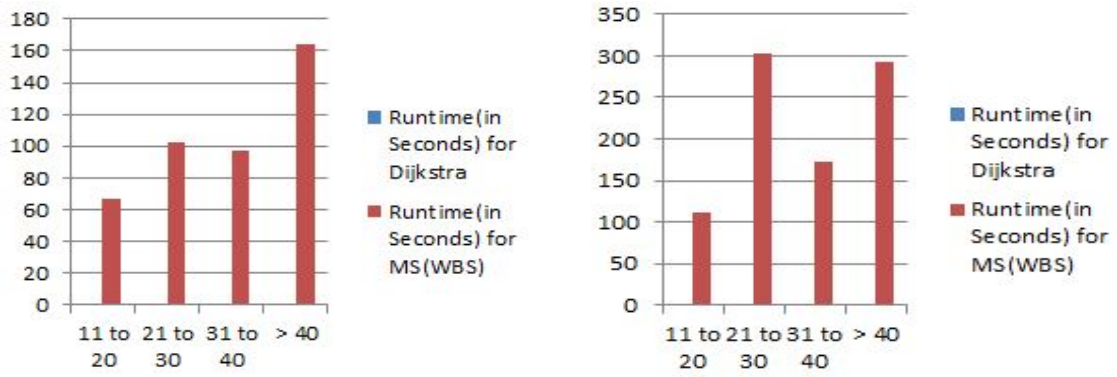
Figure 2: Comparison of runtime of both algorithm for different cost overhead

We also compute the most time consuming part of the MS(WBS) algorithm and here is the result in Figure 3. In Figure 3 we actually compute the percentage of the time it takes to find optimal replacement of each possible segment of given shortest path and percentage of the total time it spends on the rest of the algorithm. As expected we can see in Figure 3 that more than 99% of the total time it spends on calculating optimal replacement of each possible segment of the given shortest path. This is also justified by mathematics also. As if the shortest path contains an 'N' number of nodes then there are $(2^N - 1)$ possible segments available to find the optimal replacement, which makes MS(WBS) such time consuming.

(a) Dataset1 (b) Dataset2

Figure 3: Compare percentage of time taken by finding optimal replacement of all possible segment with rest of the algorithm

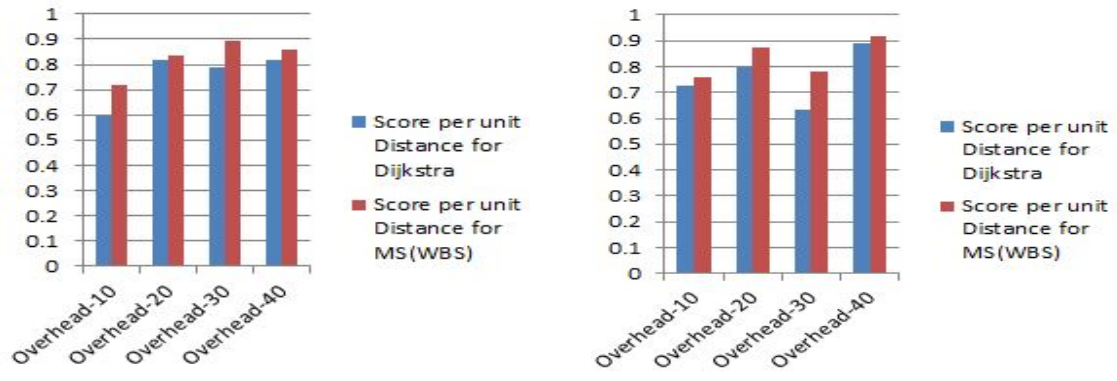
We also analyze the runtime of MS(WBS) with 30% overhead for varying path length and the result in Figure 4 also supports our above explanation why MS(WBS) is such time consuming. Average runtime for Dijkstra was 0.23, 0.238, 0.224 and 0.233 seconds for path length 11-20, 21-30, 31-40 and >40 respectively for dataset1 and 0.654, 0.664, 0.66 and 0.615 seconds for path length 11-20, 21-30, 31-40 and >40 respectively for dataset2. For MS(WBS) it was 67.03, 102.3, 97.03, 164.44 seconds and 111.93, 303.41, 173.5, 291.83 second for dataset1 and dataset2 respectively. From Figure 4, we can see that the difference between the runtime of both the algorithms increases with path length. This is obvious because if we assume each edge has nearly the same cost, then more path length implies more edges in the path as well as more nodes in the path.



(a) Dataset1 (b) Dataset2

Figure 4: Comparison of runtime of both algorithm for different pat length

Finally we compute the navigability score per distance for both algorithms. This comparison is important because one can claim that MS(WBS) achieves more score value with loss of travelling more distance which may decrease the score per distance value. But from Figure 5 we can easily show that whatever score gain MS(WBS) achieves, it always results in an increasing score per distance value also. Here we do our experiment for paths with length 31-40 for both of the datasets.



(a) Dataset1 (b) Dataset2

Figure 5: Comparison of score per unit distance for both algorithm for different cost overhead

3 Conclusion

All above comparisons show every ups and downs of MS(WBS) algorithm. But it always fulfills our main motto of designing this. In every case MS(WBS) gives us positive score gain as well as increases the score per distance value also. We calculate score gain for different overhead as user preference may vary with circumstance. In each case it shows improvement over Dijkstra. That's why it seems like the most reliable algorithm for the MNP problem. The only drawback of this algorithm is calculating optimal replacement for each possible segment. If we can manage this in an optimized way through future work it can be more efficient for solving MNP problems.