TECHNICAL REPORT

TR-CS-NMSU-2022-02-23

Qixu Gong
Huiping Cao

Department of Computer Science
New Mexico State University
1 DETAILED ANALYSIS OF FACTORS AFFECTING SPQ EVALUATION

To get a better understanding of the factors that affect the performance of Skyline Path Query (SPQ) evaluation, we conduct a detailed analysis of the relationships between SPQ answers and the characteristics of the underlying graphs using two real-world road networks, California [2] (L_CAL) with 21,048 nodes and 21,693 edges and a subgraph of the road network in New York City [1] (C9_NY_22K) containing 22,000 nodes and 30,900 edges. The L_CAL and the C9_NY_22K datasets are used in previous works [5] and [6, 9] respectively. These two networks have similar size from the perspective of the number of nodes and edges. There is only one cost dimension for the edges in the original graphs. We generate two more cost dimensions for each edge by letting the cost follow a uniform distribution in the range of [1, 100], which is a commonly used strategy when the evaluation is conducted on multi-cost road networks [3–5]. In total, three cost weights are associated with each edge. The detailed node degree distributions of the two networks are listed in Table 1.

Table 1: Node-degree distribution of two graphs

<table>
<thead>
<tr>
<th>Degree</th>
<th>L_CAL # of nodes</th>
<th>C9_NY_22K # of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>152</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>19,683</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>915</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>255</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>76</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

With these two similar size graphs, we conduct a detailed analysis to understand how a SPQ algorithm performances differently. We implement the SPQ method in [5] that grows skyline path solutions using the best-first search strategy and applies pruning techniques with landmark indexes. Additionally, we speed up the query process by initializing the result set with the shortest path on each dimension. We call this improved version of [5] the Backbone Index to Support Skyline Path Queries over Multi-cost Road Networks.

Despite the two road networks have similar numbers of nodes and edges, the results show a vast difference in query performance. The query time on C9_NY_22K is almost 23 times more than that for queries on L_CAL. The number of the skyline paths returned from C9_NY_22K is 10 times more than that from L_CAL. Even worse, 42 queries cannot finish in half an hour on the C9_NY_22K graph. A major reason behind this performance difference is that the degree distribution is different. Most of the nodes have degree 2 on the L_CAL, but 3 on the C9_NY_22K dataset.

We further analyze the average number of nodes that are visited during the query process (avg. coverage) and the number of distinct nodes showing in the returned paths. The results on these two characteristics are similar. These results indicate that the query process has similar exploration behavior, and the returned paths consist of approximately the same sets of nodes. However, when more nodes have a higher degree (even one more), there are more ways that paths can be constructed because more edges can be selected when a candidate path reaches a high-degree node. It means the probability that other paths do not dominate the candidate path is high when the node’s degree is high.

The average length of the shortest paths on all the dimensions is 287 on the L_CAL, which is much higher than 92 on the C9_NY_22K graph. Interestingly, we note that the number of the skylines paths found on L_CAL is 82, which is much smaller than 1097 on C9_NY_22K. It further confirms that more skyline paths are found when the nodes have a higher degree, despite that these skyline paths have fewer hops. This detailed analysis shows that the skyline query’s performance is highly sensitive to the degree distribution of a network.

Table 3: Analysis of running BBS algorithm on C9_NY_22K

<table>
<thead>
<tr>
<th># of hops of shortest paths</th>
<th>Query time (ms)</th>
<th># of skyline paths</th>
<th>coverage (%)</th>
<th># of unfinished queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4.25</td>
<td>12.88</td>
<td>0.57</td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>17.69</td>
<td>38.31</td>
<td>1.76</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td>116.00</td>
<td>64.07</td>
<td>3.37</td>
<td>-</td>
</tr>
<tr>
<td>40</td>
<td>345.23</td>
<td>202.59</td>
<td>5.11</td>
<td>-</td>
</tr>
<tr>
<td>50</td>
<td>640.65</td>
<td>202.83</td>
<td>8.69</td>
<td>-</td>
</tr>
<tr>
<td>60</td>
<td>2457.07</td>
<td>415.63</td>
<td>11.61</td>
<td>-</td>
</tr>
<tr>
<td>70</td>
<td>654.39</td>
<td>613.65</td>
<td>19.28</td>
<td>-</td>
</tr>
<tr>
<td>80</td>
<td>11825.90</td>
<td>529.80</td>
<td>25.32</td>
<td>-</td>
</tr>
<tr>
<td>90</td>
<td>31537.19</td>
<td>1039.26</td>
<td>29.01</td>
<td>-</td>
</tr>
<tr>
<td>100</td>
<td>77220.57</td>
<td>1641.71</td>
<td>36.64</td>
<td>-</td>
</tr>
<tr>
<td>110</td>
<td>132296.29</td>
<td>1730.67</td>
<td>46.17</td>
<td>-</td>
</tr>
<tr>
<td>120</td>
<td>189357.00</td>
<td>2551.67</td>
<td>46.19</td>
<td>-</td>
</tr>
<tr>
<td>130</td>
<td>393314.64</td>
<td>3764.79</td>
<td>59.24</td>
<td>3</td>
</tr>
<tr>
<td>140</td>
<td>118297.20</td>
<td>1574.20</td>
<td>48.92</td>
<td>7</td>
</tr>
<tr>
<td>150</td>
<td>353904.25</td>
<td>5249.13</td>
<td>62.89</td>
<td>5</td>
</tr>
</tbody>
</table>

We further examines the effect of path hops on the performance of SPQ evaluations by running the BBS method over C9_NY_22K. The results are reported in Table 3. The results show that the number of path hops profoundly influences the number of results and the number of unfinished queries.
query time. Even when the number of hops increases with 10 more hops, the query time and the size of the result set can be doubled. For example, when the number of hops increases from 50 to 60, the result size and the query time are increased from 202.83 skyline paths and 640.64 ms to 415.63 paths and 2459.07 ms (on average) respectively. The number of queries that cannot finish in 30 minutes increases when the number of hops reaches 140. The query time and the number of skyline paths have a sudden drop at 140 and jump back at 150. This is due to the increasing number of unfinished queries that lead to more missing results.

We also examine the effect of the node coverage to the query performance. Although BBS uses strategies and auxiliary structures to reduce the search space [5], its exploration space can still reach up to more than half of the network nodes. When the number of hops reaches 150, the nodes that are visited during the query process are more than 60% of the nodes in the network. These statistics show that the number of path hops plays a critical role in SPQ evaluation. It further confirms the difficulty of improving SPQ algorithms due to high node coverage.

2 INDEX MAINTENANCE FOR NETWORK UPDATES

Road networks may change. We discuss how to maintain the backbone index when a road network is updated. It has shown [6, 7] that allowing network updates to support shortest-path queries is challenging. Thus, updating the index that supports SQPs is more complicated. We propose a practical method to update the backbone index with low overhead for road network updates compared with the huge index construction time. We consider four types of operations: (i) insertion of a new vertex, (ii) removal of a vertex, (iii) insertion of an edge, and (iv) deletion of an edge.

Operation (i), node insertion. We define the insertion as adding a new node \( u \) with a set of new edges \( E_u = \{(u,v)\} \). The operations involve two steps. The first step finds all the \( E_u \) edges \( \{(u,v)\} \) that connect to any degree-1 edges \( E_v \), and conducts edge removal on \( E_v \cup \{(u,v)\} \) by running regular summarization (i.e., removing the degree-1 edges from graph \( G_I \) until all the remaining nodes in \( G_I \) have a degree 2 or more). The second step processes the \( E_u \) edges \( \{(u,v)\} \) that do not connect to any degree-1 edges by updating the skyline paths from \( u \) to the highway entrances of the nodes in the influenced dense clusters. A dense cluster is influenced when an \( E_u \) edge connects to it. In particular, we first assign \( u \) to a non-noise cluster \( C_{0,j} \) at \( G_0 \), where more edges in \( E_u \) connect to the cluster \( C_{0,j} \). Then, we put all the clusters that the node \( u \) is connected to into a first-in-first-out (FIFO) queue, and recursively update the backbone index from \( G_0 \) to \( G_L \) by de-queuing one cluster (say \( C_{i,j} \)) from the queue and recalculating the skyline paths from \( v \in C_{i,j} \) to their highway entrance \( h_{t,v}^{\text{ui}} \). After this, we find all the clusters \( C_{i+1,k} \) in the abstracted graph \( G_{i+1} \), such that \( C_{i+1,k} \cap C_{i+1,k} \neq \emptyset \), and put them to the FIFO queue. At the end, the landmark index of \( G_L \) is rebuilt.

If the aggressive summarization strategy, which condenses all single segments, is triggered at one level during index update, the skyline paths consisting of single segments need to be recomputed. If the nodes and the edges on a single segment \( \text{seg} \) is deleted, \( \text{seg} \) is split into small segments and skyline paths are recalculated on them. The new labels are used to update the existing labels subsequently.

Operations (ii) node removal, (iii) edge insertion, and (iv) edge removal. These operations are based on the operation of inserting new nodes. Adding a new edge is a special case of inserting of a new node with one edge. When deleting a node and an edge, the dense clusters containing the target node and edge are determined and the index is updated with a similar process as the node insertion.

3 EXTENDED TO SUPPORT ONE-TO-ALL SPQS

The query processing algorithm (Algorithm 3) can be extended to support one-to-all SQPs that return approximate skyline paths to all other nodes from a given query node in a graph \( G \). The modification mainly occurs in the loop for concatenating the backbone paths from the target node \( v_t \) to the highest level graph \( G_L \) (Lines 16 - 28). At first, we initialize \( D \) with all the nodes that have labels at \( L_0 \) (Line 4) (instead of the target node \( v_t \)). Also, the node having labels at \( L_t \) needs to be added to \( D \) if it is not in \( D \) when the loop executes at level \( l \). At last, \( m\_\text{BBS} \) is conducted to find skyline paths from \( S_{\text{possible}} \) to all nodes \( G_L \) at level \( L \) (Line 32).

4 COMPLEXITY ANALYSIS

4.1 Bound of weights of approximate solutions

To our best knowledge, there is no theoretical analysis about the goodness of the approximate solutions for SPQ on MCRNs. We assume that edge weights follow an independent identically distribution (i.i.d.). We analyze the upper bound by using the worst approximate path result that can be returned by our method.

Lemma 1 (Upper Bound of Path Weights). Given a graph \( G \), its backbone index, a query \( (v_s,v_t) \), the upper bound of an approximate solution path’s weight is \( \Omega((F_{out})^2) \).

Here, \( L \) is the height of the index. \( F_{out} \) is the expected summation of the weights for all the edges in the minimum spanning tree (MST) over a complete graph with \( n \) number of nodes (\( n \) is large number). When \( n \) approaches infinity, \( F_{out} \) is calculated as \( \zeta(3)/F'(0) \) \([8]\) where \( \zeta \) is the Riemann zeta function that edge weights follow. This bound is very loose. In the future, we will explore theories to make this bound tighter.

Proof: Each node belongs to either a dense cluster, a single segment, a degree-1 edge, or the highest level summarized graph \( G_L \). The costs from a node in single segments and degree-1 edges to its highway entrance do not change after condensing and are kept in the index. We abstract each dense cluster to a spanning tree utilizing the degree pair value of edges, which makes it challenging to analyze the expected summation of weights in the spanning tree. We prove it using an extreme case of estimating the weight of the minimum spanning tree (MST) over a complete graph whose edge weights follow an identically independent distribution (i.i.d.) \( F \). According to \([8]\), the expected weight of the minimum spanning tree (MST) over a complete graph whose edge weights follow an i.i.d. \( F \) is \( F_{out} \). Here, \( F_{out} = \zeta(3)/F'(0) \) where \( \zeta \) is the Riemann zeta function and \( F' \) is the derivative of the distribution function \( F \).

Given a query \((v_s,v_t)\) and the original graph \( G_0 \), the distance between \( v_s \) and \( v_t \) on \( G_0 \), \( \text{dist}(v_s,v_t,G_0) = d_{v_s,v_t} + \text{dist}(h_{v_s}^L,h_{v_t}^L,G_1) \)
Here, \( c_2 \) is a constant factor and \( c_3 = c2 \cdot \text{deg} \) where \( \text{deg} \) is the largest degree of graph nodes. The landmark building at the highest-level graph is almost constant because of the small size of \( G_0 \). To sum up, the time complexity for building the index at level \( i \) is:

\[
\text{Compl}_i = \frac{|G_i|}{m_{\text{max}}} (C_{\text{Dij}} + C_{\text{spa}}) + C_{\text{cluster}}
\]

\[
= \frac{|G_i|}{m_{\text{max}}} (c_0 \cdot (m_{\text{max}} + |E^1_i|) + c_1 \cdot |C_{i,j}, E|) \log(m_{\text{max}})
\]

\[
+ c_3 \cdot |G_i| \log(|G_i|, |V|) \tag{1}
\]

\[
\leq \frac{|G_i|}{m_{\text{max}}} (c_4 \cdot m_{\text{max}} \log(m_{\text{max}}) + c_5 \cdot |G_i| \log(|G_i|, |V|)) \tag{2}
\]

\[
\leq c_4 \cdot (|G_i|, \log(m_{\text{max}})) + c_5 \cdot |G_i| \log(|G_i|, |V|) \tag{3}
\]

\[
\leq c_5 \cdot (|G_i|, \log(m_{\text{max}})) + \log(|G_i|, |V|) \tag{4}
\]

\[
\leq c_5 \cdot (|G_i|, \log(|G_i|)) + \log(|G_i|, |V|) \tag{5}
\]

\[
= 2 \cdot c_5 \cdot |G_i| \log(|G_i|, |V|) \tag{6}
\]

where \( c_4 = c_0 \cdot (m_{\text{max}} + |E^1_i|) + c_1 \cdot |C_{i,j}, E|, c_5 = m_{\text{max}} |c_3, c_4 | \) are constant factors. In a cluster \( C_{i,j}, |E^1_i| \leq |C_{i,j}, E| \leq |\text{deg} \cdot |C_{i,j}, E, | \) and \( |G_i|, \approx \text{deg} \cdot |G_i|, V, |G_i| \). These relations are used in the derivation from (2) to (3). And \( |G_i|, \frac{|G_i|}{|m_{\text{max}}|} \leq |G_i|, m_{\text{max}} \leq |G_i|, \) which means the production of the number of clusters and the average number of nodes in each cluster equals to the number of nodes of a graph. In (4), \( m_{\text{max}} \) is generally a constant (e.g., 4 in real-world road networks). Overall, the construction complexity is

\[
\sum_{0}^{L} c_0 \cdot |G_i| \log(|G_i|, |V|) = c_7 \cdot |G_i| \log(|G_i|, |V|) \tag{7}
\]

where \( c_0 \) and \( c_7 \) are constant factors. The index level \( L \) is controlled by the parameter \( p \) because \( p \) is decided by the number of edges and nodes removed in each abstracted graph. \( L \) is almost constant in the backbone index once \( p \) is set. \( \Box \)

### 4.3 Space complexity

**Lemma 3 (Index space).** Given a graph \( G \), the parameter \( p \) (used to control the percentage of the nodes removed in a cluster), the maximum size of a dense cluster \( m_{\text{max}} \), and the number \( d \) of weights for each edge in \( G \), the size of the Backbone index is

\[
O(|G|, V|m_{\text{max}}, S_n, d),
\]

where \( S_n \) is the average number of skyline paths between every node to its highway entrance in each dense cluster.

**Proof:** The space use is \( c_{\text{const}} \cdot L \cdot |G_i|, V|/m_{\text{max}}, m^2_{\text{max}}, S_n, d \). This is because the index has \( L \) levels. In each index level there are \( |G_i|, V|/m_{\text{max}} \) clusters. In each cluster, there are at most \( m^2_{\text{max}} \) node pairs. For each node pair, \( S_n \) is the average number of skyline paths between every node to its highway entrance in each dense cluster. \( d \) is the number of edge weights. The space complexity is \( O(|G|, V|m_{\text{max}}, S_n, d) \).

Here, \( S_n \) is almost constant when \( m_{\text{max}} \) is small. (\( S_n \) is no more than 10 when \( m_{\text{max}} = 200 \) in our experiment setting.) \( \Box \)

### 5.1 One-to-all SPQ evaluation

We evaluate 20 one-to-all SPQs on the subgraphs of \( C9NY \) that have 5K, 10K, 15K, and 22K nodes, respectively. Since no implementation supports efficient exact one-to-all SPQs on road networks,
we implemented a BBS variation as the baseline method that supports this type of query by removing the lower-bound evaluation and target checking of the BBS. We denote this method as one_BBS. Similarly, we call our extended method one_backbone. Table 4 shows that our method one_backbone outperforms the one_BBS on query time. The query time of one_BBS increases exponentially as the graph size increases, but our one_backbone offers stability in this aspect. Further, we verify the quality of the approximate results by calculating the average goodness from the query node to all other nodes. The results are also shown in Table 4. The quality is similar to what we observe from the single-destination queries. We note that the query performance on C9_NY_10K does not grow exponentially in graph size, but even worse. This is because the C9_NY_10K has higher node degrees than other subgraphs. This further confirms that the degree distribution is a critical factor for any skyline path queries.

Table 4: Evaluation on one-to-all SPQs

<table>
<thead>
<tr>
<th></th>
<th>Query time (Sec)</th>
<th>Goodness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>one_BBS</td>
<td>one_backbone</td>
</tr>
<tr>
<td>C9_NY_3K</td>
<td>7.17</td>
<td>4.76</td>
</tr>
<tr>
<td>C9_NY_10K</td>
<td>302.52</td>
<td>40.303</td>
</tr>
<tr>
<td>C9_NY_15K</td>
<td>383.99</td>
<td>13.99</td>
</tr>
<tr>
<td>C9_NY_22K</td>
<td>862.66</td>
<td>31.17</td>
</tr>
</tbody>
</table>

5.2 Evaluation on Backbone index updating

We evaluate the performance of the index updating using 4 subgraphs of C9_NY and four real-world small road networks, C9_NY (abbreviated as NY), C9_BAY (BAY for short), C9_COL (COL for short), and C9_FLA (FLA for short)). The results are shown in Figure 1. As mentioned in Section 2, the node insertion is the most fundamental and complicated operator of the four operations, and the other three are based on the insertion. So, we evaluate the maintenance cost only considering the node-insertion operation. We evaluate 20 node-insertion operations where each insertion is to add one random node with five new edges connected to its 5 nearest neighbor (NN) nodes. The overall updating time is proportional to the size of the graph. The updating only needs 350 seconds on the FLA dataset that contains one million nodes. As the graph size increases, the ratio of the index update time over the complete index construction time decreases. The ratio drops from 17.7% to 8.2% when the size of the C9_NY subgraphs grows from 5K to 22K (Figure 1(a)), and settles at around 3% on real-world datasets (Figure 1(b)). As road networks are not updated frequently, thus the index updating time is acceptable compared with the large full-index construction time.

REFERENCES