TECHNICAL REPORT

TR-CS-NMSU-2022-02-23

Qixu Gong
Huiping Cao

Department of Computer Science
New Mexico State University

Backbone Index to Support Skyline Path Queries over Multi-cost Road Networks

Supplementary materials

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

L_CAL		C9_NY_22K	
Dregree	**# of nodes**	**Dregree**	**# of nodes**
1	182	1	2,836
2	**19,683**	2	3,611
3	915	3	**10,595**
4	255	4	4,843
5	7	5	105
6	5	6	10
8	1	-	-

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 **B**aseline **B**est-first **S**earch method (abbreviated as **BBS**). We randomly generate 300 queries on each road network. We summarize basic statistics of the query results and show them in Table 2.

Table 2: SPQ query performance comparison

	L_CAL	C9_NY_22K
avg. query time (ms)	2,967	68,056
avg. # of hops of the shortest path	287	92
avg. # of skyline path results	82	1,097
avg. coverage	26%	24%
avg. # of nodes in the results	3%	1.5%
# of unfinished queries in 30 min.	0	42

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

# of hops of shortest paths	Query time (ms)	# of skyline paths	coverage (%)	# of unfinished queries
10	4.25	12.88	0.57	-
20	17.69	38.31	1.76	-
30	116.00	64.07	3.37	-
40	455.23	202.59	5.11	-
50	640.65	202.83	8.69	-
60	2459.07	415.63	11.61	-
70	6544.39	613.65	19.28	-
80	11825.90	529.80	23.32	-
90	31577.19	1039.26	29.01	-
100	77220.57	1641.71	36.64	-
110	132296.29	1750.67	46.17	-
120	189357.00	2551.67	46.19	1
130	395374.64	3764.79	59.24	3
140	118297.20	1574.20	48.92	7
150	353904.25	5249.13	62.89	5

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 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_s, and conducts edge removal on $E_s \cup \{(u, v)\}$ 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_v^{i+1}. After this, we find all the clusters $C_{i+1,k}$ in the abstracted graph G_{i+1}, such that $C_{i,j}.V \cap C_{i+1,k}.V \neq \emptyset$, 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 seg is deleted, 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 SPQs 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 $\mathbb{D}$ with all the nodes that have labels at I_0 (Line 4) (instead of the target node v_t). Also, the node having labels at I_i needs to be added to $\mathbb{D}$ if it is not in $\mathbb{D}$ when the loop executes at level i. At last, **m_BBS** is conducted to find skyline paths from $S_{possible}$ to all nodes $G_L.V$ 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 $O((F_{val})^L)$.*

Here, L is the height of the index. F_{val} 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_{val} is calculated as $\zeta(3)/F'(0)$ [8] where ζ 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 following an i.i.d. F is F_{val}. Here, $F_{val} = \zeta(3)/F'(0)$ where ζ 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, $dist(v_s, v_t, G_0) = d_{v_s} + d_{v_t} + dist(h^1_{v_s}, h^1_{v_t}, G_1)$

if v_s and v_t do not belong to the same cluster at G_0. Here, d_{v_s} and d_{v_t} are the distances from v_s and v_t to their corresponding highway entrances $h^1_{v_s}$ and $h^1_{v_t}$ at G_1 respectively. $d_{v_s} + d_{v_t} = O(2 \cdot F_{val})$ because the maximum distance from a node to its highway entrances is the weight of the spanning tree, which is F_{val}. The $dist(h^1_{v_s}, h^1_{v_t}, G_1)$ represents the distance from $h^1_{v_s}$ to $h^1_{v_t}$ at graph G_i where $0 < i < L$ that goes through the summarized graph G_1. Then, $dist(h^1_{v_s}, h^1_{v_t}, G_1) =$ c_1 $\cdot F_{val}$ (c_1 is a constant factor) if $h^1_{v_s}$ and $h^1_{v_t}$ meet in a dense cluster $C_{1,j}$ on G_1. Otherwise, $dist(h^1_{v_s}, h^1_{v_t}, G_1)$ can be defined as $d_{h^1_{v_s}} + d_{h^1_{v_t}} + dist(h^2_{v_s}, h^2_{v_t}, G_2)$, where $d_{h^1_{v_s}}$ and $d_{h^1_{v_t}}$ are the distances from $h^1_{v_s}$ and $h^1_{v_t}$ to their corresponding highway entrances $h^2_{v_s}$ and $h^2_{v_t}$ at a higher level graph G_2. Here, the expected weight of one edge in G_1 is the expected weight of the spanning tree that condenses a dense cluster in G_0, which is F_{val}. The distance from $d_{h^1_{v_s}}$ and $d_{h^1_{v_d}}$ to their corresponding highway entrance on G_2 is $(F_{val})^2$ because F_{val} is a constant. We can get this using a similar analysis as that used to analyze the distance d_{v_s} and d_{v_d} on G_0. Thus, $d_{h^1_{v_s}} + d_{h^1_{v_t}} = c_2 \cdot 2 \cdot (F_{val})^2$ where c_2 is a constant factor.

In the worst case, v_s and v_t can not meet until G_L. We can derive the distance from v_s and v_t that goes through G_{L-1} to G_L as $O_{L-1} = c_2 \cdot 2 \cdot (F_{val})^L + dist(h^L_{v_s}, h^L_{v_t}, G_L)$. Let k be the length of a skyline path connecting $h^L_{v_s}$ and $h^L_{v_t}$ (the highway entrances of v_s and v_t on G_L respectively) on G_L. The distance go along the G_L, $O_L = dist(h^L_{v_s}, h^L_{v_t}, G_L) = c_{l+1} \cdot k \cdot (F_{val})^L$ (c_{l+1} is a constant factor), where one edge on G_L is the condensed edge from G_0 through L levels. Overall, the bound of the worst case is $O((F_{val})^L)$ because

$$\begin{aligned} & c_1 \cdot 2 \cdot F_{val} + c_2 \cdot 2 \cdot (F_{val})^2) + \cdots + c_L \cdot 2 \cdot (F_{val})^L + O_L \\ \approx & c_L \cdot 2 \cdot (F_{val})^L + c_{l+1} \cdot k \cdot (F_{val})^L. \end{aligned}$$

□

4.2 Complexity of index construction

LEMMA 2 (INDEX CONSTRUCTION TIME). *Given a graph G, the parameters p and m_{max}, the time to construct the Backbone index of G is $O(|G.V|log(|G.V|))$. Here, $|G.V|$ is the number of graph nodes.*

***Proof*:** Recall that L is the level of the index and m_{max} is the maximum size of clusters. The time of building the index at level i consists of three major parts.
(i) For level i, the time for finding skyline paths on a small cluster $C_{i,j}$ is

$$\begin{aligned} C_{Dij} &= c_0 \cdot (|(C_{i,j}.V| + |E^i_r|) \log(|C_{i,j}.V|) \\ &\leq c_0 \cdot (m_{max} + |E^i_r|) \log(m_{max}) \end{aligned}$$

, where c_0 is a constant factor, E^i_r is the set of removed edges from $C_{i,j}$, and $m_{max} = max\{|C_{i,j}.V|\}$.
(ii) The time of finding the spanning tree over a dense cluster is

$$C_{spa} = c_1 \cdot (|C_{i,j}.E| log(m_{max}))$$

where c_1 is a constant factor.
(iii) The time to find dense clusters on G_i is

$$\begin{aligned} C_{cluster} &= c_2 \cdot (|G_i.E| log(|G_i.V|)) \\ &\leq c_2 \cdot (deg \cdot |G_i.V| log(|G_i.V|)) \\ &= c_3 \cdot |G_i.V| log(|G_i.V|) \end{aligned}$$

Here, c_2 is a constant factor and $c_3 = c2 * deg$ where 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_L. To sum up, the time complexity for building the indes at level i is:

$$\begin{aligned} Compl_i = & \frac{|G_i.V|}{m_{max}}(C_{Dij} + C_{spa}) + C_{cluster} \\ = & \frac{|G_i.V|}{m_{max}}(c_0 \cdot (m_{max} + |E^i_r|) + c_1 \cdot |C_{i,j}.E|) \log(m_{max}) \\ & + c_3 \cdot G_i.|V| log(G_i.|V|) & (1) \\ \leq & \frac{|G_i.V|}{m_{max}}(c_4 \cdot m_{max}) \log(m_{max}) + c_3 \cdot |G_i.V| log(|G_i.V|)) & (2) \\ = & c_4 \cdot (|G_i.V| \log(m_{max})) + c_3 \cdot |G_i.V| log(|G_i.V|) & (3) \\ \leq & c_5 \cdot (|G_i.V|(\log(m_{max}) + log(|G_i.V|))) & (4) \\ \leq & c_5 \cdot (|G_i.V|(\log(|G_i.V|) + log(|G_i.V|))) & (5) \\ = & 2 \cdot c_5 \cdot |G_i.V| log(G_i.|V|)) & (6) \end{aligned}$$

, where $c_4 = c_0 \cdot (m_{max} + |E^i_r|) + c_1 \cdot |C_{i,j}.E|$, $c_5 = max\{c_3, c_4\}$ are constant factors. In a cluster $C_{i,j}$, $|E^i_r| \leq |C_{i,j}.E| \approx deg \cdot |C_{i,j}.V|$ and $|G_i.E| \approx deg \cdot |G_i.V|$. These relations are used in the derivation from (2) to (3). And $\frac{|G_i.V|}{m_{max}}|C_{i,j}.V| \leq \frac{|G_i.V|}{m_{max}} \cdot m_{max} = |G_i.V|$, 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_{max} is generally a constant (e.g., 4 in real-world road networks). Overall, the construction complexity is

$$\sum_{0}^{L} c_6 \cdot |G_i.V| log(G_i.|V|) = c_7 \cdot |G.V| log(|G.V|)$$

where c_6 and c_7 are constant factors. The index level L is controlled by the parameter p because it 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. □

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_{max}, and the number d of weights for each edge in G, the size of the Backbone index is of $O(|G.V| m_{max} S_n d)$, 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 $c_{const} \cdot L \cdot \frac{|G_i.V|}{m_{max}} \cdot m^2_{max} \cdot S_n \cdot d$. This is because the index has L levels. In each index level there are $\frac{|G_i.V|}{m_{max}}$ clusters. In each cluster, there are at most m^2_{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 $O(|G.V| m_{max} S_n d)$.

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

5 MORE EXPERIMENTAL RESULTS

5.1 One-to-all SPQ evaluation

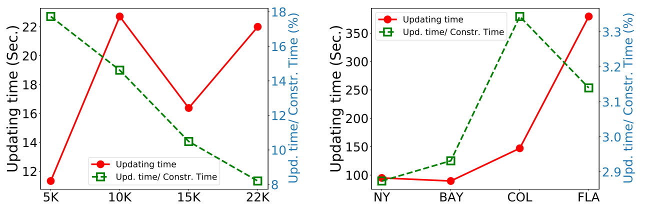
We evaluate 20 one-to-all SPQs on the subgraphs of *C9_NY* that have 5K, 10K, 15K, and 22K nodes, respectively. Since no implementation supports efficient exact one-to-all SPQs on road networks, we implemented a *BBS* variation as the baseline method that supports this type of query by removing the lower-bound 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

	Query time (Sec)		Goodness
	one_BBS	one_backbone	Cosine Similarity
C9_NY_5K	7.17	4.76	0.86759
C9_NY_10K	302.52	40.303	0.88954
C9_NY_15K	383.99	13.99	0.89174
C9_NY_22K	862.66	31.17	0.89825

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.



(a) Size of C9_NY subgraphs **(b) Cities**

Figure 1: Time to index updating

REFERENCES

[1] 9th DIMACS Implementation Challenge. http://users.diag.uniroma1.it/challenge9/download.shtml.

[2] Real Datasets for Spatial Databases: Road Networks and Points of Interest. https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm.

[3] Yi-Chung Chen and Chiang Lee. Skyline path queries with aggregate attributes. *IEEE Access*, 4:4690–4706, 2016.

[4] Qixu Gong, Huiping Cao, and Parth Nagarkar. Skyline queries constrained by multi-cost transportation networks. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 926–937, 2019.

[5] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route skyline queries: A multi-preference path planning approach. *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 261–272, 2010.

[6] Zijian Li, Lei Chen, and Yue Wang. G*-tree: An efficient spatial index on road networks. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 268–279, 2019.

[7] Liam Roditty and U. Zwick. On dynamic shortest paths problems. *Algorithmica*, 61:389–401, 2010.

[8] J. Steele. Minimal spanning trees for graphs with random edge lengths. 2002.

[9] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. G-tree: An efficient index for knn search on road networks. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 39–48, 2013.