Sketch Path Algorithm by using Shortest Distance Computing Mechanism

Miss. Swati Ramteke, Prof. Komal Ramteke
Dept. of Computer Engineering,SKN sit's Lanavala,
Ravij Gandhi college of engg. Nagpur
India,
swati19910618@gmail.com , *komalramteke03@gmail.com*

***Abstract*—**Nowadays various application domains is dramatically increasing, the number of nodes may reach the scale of hundreds of millions or even more. Computing shortest paths between two given nodes is a fundamental operation over graphs, but known to be nontrivial over large disk-resident instances of graph data. While a number of techniques exist for answering reachability queries and approximating node distances efficiently, determining actual shortest paths (i. e. the sequence of nodes involved) is often neglected. However, in applications arising in massive online social networks, biological networks, and knowledge graphs it is often essential to find out many, if not all, shortest paths between two given nodes. Shortest distance query is a fundamental operation in large-scale networks. Many existing methods in the literature take a landmark embedding approach, which selects a set of graph nodes as landmarks and computes the shortest distances from each landmark to all nodes as an embedding. To answer a shortest distance query, the pre computed distances from the landmarks to the two query nodes are used to compute an approximate shortest distance based on the triangle inequality. In this paper, I analyze the factors that affect the accuracy of distance estimation in landmark embedding. In particular, find that a globally selected, query in dependent landmark set may introduce a large relative error, especially for nearby query nodes. To address this issue, I propose a query-dependent local landmark scheme, which identifies a local landmark close to both query nodes and provides more accurate distance estimation than the traditional global landmark approach. We propose efficient local landmark indexing and retrieval techniques with a scalable sketch-based index structure that not only supports estimation of node distances, but also computes corresponding shortest paths themselves to achieve low offline indexing complexity and online query complexity.

***Keywords*—** *Local landmark, Sketched-based index, Time complexity, Space complexity, Shortest distance query, cycle elimination, shortcutting, path sketch.*

*****

I. INTRODUCTION

As the size of graphs that emerge nowadays from various application domains is dramatically increasing, the number of nodes may reach the scale of hundreds of millions or even more. Due to the massive size, even simple graph queries become challenging tasks. One of them, the shortest distance query, has been extensively studied during the last four decades. Querying shortest paths or shortest distances be ten nodes in a large graph has important applications in many domains including road networks, communication networks, social networks, the Internet, and so on. Graphs are routinely used in the modern digital world in a number of settings, such as online social networks (like LinkedIn and Facebook), biological interaction models, transportation networks, the massive hyperlink graph between documents of the World Wide Web, XML data, and many more. Due to the ever increasing size of the graphs of an interest, many seemingly straightforward operations become challenging. In this paper, we turn our attention to the computation of shortest paths between any two nodes in the graph, a problem with long algorithmic history. This operation forms a building block for many mining tasks, and is also an increasingly important application in itself over instances such as online social networks. Consider the following two application scenarios for shortest path queries:

A. Social networks such as LinkedIn enable professional networking among individuals. A person interested in reaching out to the hiring manager of his favourite future employer would seek a shortest path to reach that person, starting from his friends.

B. Biological (metabolic) networks (such as the Biochemical Network Database) model the complex chemical processes within an organism. A biologist may be interested in identifying biotransformation paths between two target metabolites to help in designing experiments in the wet lab.

II. LITERATURE REVIEW

What makes the shortest path computation particularly hard on large graphs? Dijkstra's algorithm, the classical technique to compute the shortest path between two nodes in a graph has the asymptotic runtime complexity of $O(m + n \log(n))$, where n is the number of nodes and m is the number of edges [5]. On one of the benchmark datasets that I use in this paper – any social network comprising about 3 million nodes and 220 million edges – a straight forward implementation of Dijkstra's algorithm takes more than 500 seconds on average. This is way too slow for most applications. The reason for this is that Dijkstra's algorithm has to construct and maintain shortest paths to all nodes in the graph whose distance to the source node is smaller than the distance from the source node to the destination node [2].

According to the findings in the literature [3], in the context of road networks and moving objects, the original space contains two-dimensional of the objects: intersections (original nodes) connected by some streets. The query points in such spaces are usually moving objects (e.g. cars) travelling through the streets from a source to a destination and the KNN problem is defined as finding the closest points of interest (e.g. hospitals, gas stations) to the moving objects.

According to the findings in the literature [4], the problems with finding a path through a graph have usually been approached in one of two ways, which shall call the Mathematical approach and Heuristic approach. The mathematical approach typically deals with the properties of abstract graphs and with algorithms that prescribe an orderly examination of nodes of a graph to establish a minimum cost path.

The Heuristic approach typically uses special knowledge about the domain of the problem being represented by a graph to improve the computation efficiency of solutions to particular graph-searching problems .In order to expand the fewest possible nodes in searching for an optimal path, a search algorithm must constantly make as informed a decision as possible about which node to expand next. If it expand nodes which obviously cannot be an optimal path, it is wasting effort. On the other hand, if it continues o ignore nodes that might be on an optimal path, it will sometimes fail to find such path and thus not be admissible. An efficient algorithm obviously need some way to evaluate available nodes to determine which one should be expanded next.

III. PROPOSED WORK

we build upon the recently proposed sketch-based framework , which in turn is based on the classical landmark based approach used in distance oracles. The goal is to pre-compute and store a O(n) sized *sketch* of the graph so that any distance query can be answered approximately but with high accuracy.

In summary, contributions made in this paper are as follow:

A. we introduce path-sketches, which can be effectively used in large graphs with small diameters (e.g., almost all online social networks). The path-sketches maintain the complete path information between every node and a selected set of landmark nodes, computed as part of a graph preprocessing step.

B. we develop a set of lightweight, yet highly effective, techniques that use path-sketches to significantly improve the quality of shortest path estimations.

C. Along with estimates of shortest path distances, I show how to generate corresponding instances of shortest paths themselves with no computational overhead – a feature not found so far in most distance estimating techniques. In fact, our algorithm generates a queue of paths in increasing order of their lengths, an important need in many applications over social and biological networks.

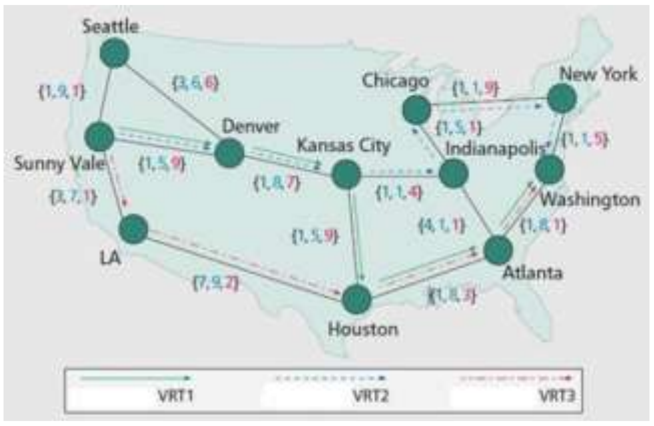


Fig. 1. Providing path diversity in the network topology

D. Finally, implement all the proposed methods in a fully functional large-scale graph processing engine, and evaluate against a number of real world large-scale graphs. In most of the datasets used, the estimates are almost always exact, and generate a number of alternative shortest paths between a given pair nodes.

IV. EVALUATION METHODS

In this section I explain our algorithms for shortest path approximation in detail.

A. Preliminaries

Let G = (V,E) denote a directed graph with vertex set V and edge set E.

1) Paths and Distances : A path p of length $l \in N$ in the graph is an ordered sequence of $l + 1$ vertices, such that there exists, for every vertex in the sequence, an edge to its subsequent vertex, except the last one:

$p = (v1, v2, . . . , vl+1)$ with $vi \in V, 1 \leq i \leq l + 1$,…..……………………………...…(1)

$(vi, vi+1) \in E, 1 \leq i < l$. …………….(2)

The distance from u to v, denoted by dist(u, v), is the number of edges in the shortest such path – or infinity if v is not reachable from u:

$dist(u, v) := (argminP(u,v) |p|$ if $P(u, v) = \emptyset, \infty$ else. …………………………………..……. (3)

2) Path Approximation : Given two vertices $u, v \in V$, let p denote a shortest path (note that there could be many) from u to v, that is, a path starting in u and ending in v with length |p| = dist(u, v). Furthermore, let q be an arbitrary path from u to v. By regarding q as an approximation of the shortest path p, I can define the approximation error of this path as error(q) :=|q| − |p|/|p|=|q| − dist(u, v)/dist(u, v) ∈ [0,∞]…………………………..……. (4)

3) Path Concatenation : Let p = (u1, u2, . . . , ul1 , ul1+1) and q = (v1, v2, . . . , vl2+1) denote paths of lengths l1 and l2 respectively. Suppose ul1+1 = v1, that is, the last node in path p equals the first node in path q. Then, I can create new path, denoted by p◦q, of length l1+l2 by concatenating the paths p and q:

p ◦ q = (u1, u2, . . . , ul1 , ul1+1) ◦ (v1, v2, . . . , vl2+1) := (u1, . . . , ul1 , v1, . , vl−2+1)…………(5)

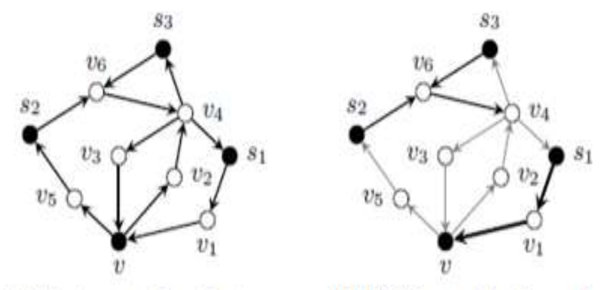
B. Sketch Algorithm

The sketch algorithm approximates the shortest path distance between two given nodes in general graphs using a landmark-based approach. In order to answer a distance query for a pair of nodes (s, d) in real time, the algorithm employs a two-staged approach: a pre-computation step to generate sketches (distances from all vertices to so-called landmark nodes) beforehand, and an approximation step that uses this pre-computed data to provide a very fast approximation of the node distance at query time. In this section I explain these two building blocks of the (modified) sketch algorithm in detail:

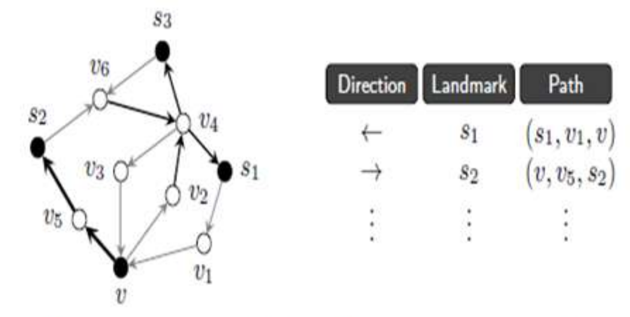
1) Precomputation : The pre-computation step involves sampling some sets of nodes, computing for every node in the graph a shortest path to and from a member of this set and storing the thus obtained set of paths on external memory. These paths will be used in the approximation step later. The preprocessing, illustrated in Figure 2, works as follows:

Seed Set Sampling

Let r := ⌊log (n)⌋ where n = |V |. I uniformly sample r + 1 sets of nodes (called seed sets) of sizes 1, 2, 22, . . . , 2r respectively. The selected sets are denoted by S0, S1, . . . , Sr.



(a) Vertex v and seed set $S = \{s_1, s_2, s_3\}$ (b) BFS from S in forward direction reaches v



(c) BFS from S in backward direction reaches v (d) Resulting entries in Sketch(v)

Fig. 2. Precomputing

```
Algorithm : Sketch(s, d)
Input : s, d ∈ V
Result : Q – priority queue of paths from s to d, ordered by path length

1. begin
2. Load sketches Sketch(s), Sketch(d) from disk
3. L ← common landmarks of Sketch(s) and Sketch(d)
4. for each l ∈ L do
5. p ← path from s to d through l
6. Add p to queue Q
7. return
```

C. Tree Algorithm

In this section we describe our third contribution, a new algorithm for shortest path approximation that also utilizes the pre-computed sketches. The sketch Sketch(v) of a node v contains two sets of paths:

(1) the set of paths connecting v to landmarks (called forward directed paths) and

(2) the set of paths connecting landmarks to v (called backward-directed paths). In the undirected setting, both sets would correspond to trees having landmarks as leaves and v as a root. Every inner node of each tree corresponds to a vertex contained in one of the paths in the sketch.

In the directed setting, only the forward-directed part of the sketch (from v to landmarks) forms a tree, while the backward-directed paths yield a tree with "reversed edges" (see Figures 3a+b). Our new algorithm, named TreeSketch, takes the two query nodes s, d as input, loads the sketches Sketch(s), Sketch(d) from disk and constructs the tree Ts, rooted at s, that contains all the forward paths stored in Sketch(s). Likewise, the "reversed tree" Td, rooted at d, containing all the backward directed paths from the landmarks to d is being created from Sketch(d). Then, the algorithm starts two breadth-first expansion on the trees simultaneously: BFS(Ts, s) from s in Ts and RBFS(Td, d) (BFS on reversed edges) from d in Td. At any point of time during execution, let VBfs and VRBfs denote the sets of

visited nodes during the respective BFS runs. For every vertex u ∈ VBfs encountered during BFS(Ts, s), the algorithm loads the list S(v) of its successors in the original graph. Then, it checks whether any of the vertices discovered during RBFS(Td, d) is contained in the list S(u). If such a vertex v ∈ S(u) ∩ VRBfs exists (see Figure 3c), I have found a path p from s to d, given by
$p = p_{s \to u} \circ (u, v) \circ p_{v \to d}$,

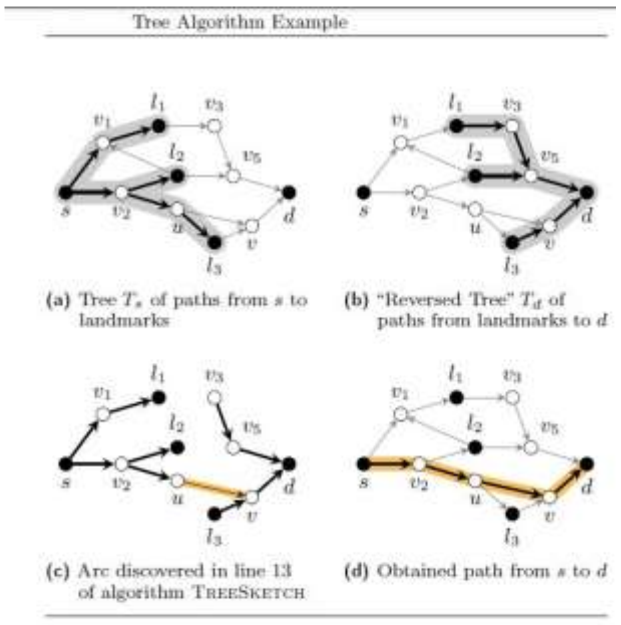

Tree Algorithm Example
l_1
v_3
v_1
l_2
v_5
s
v_2
u
v
d
l_3
(a) Tree T_s of paths from s to landmarks
(b) "Reversed Tree" T_d of paths from landmarks to d
(c) Arc discovered in line 13 of algorithm TREESKETCH
(d) Obtained path from s to d

Fig. 3. Tree algorithm

where $p_{s \to u}$ and $p_{v \to d}$ denote the paths from s to u in Ts and from v to d in Td respectively (Figure 3d). An equivalent procedure is executed for every vertex encountered during reverse BFS from d. I continue this procedure of BFS expansions and successor list checks, adding paths discovered along the way to the queue Q. Let lshortest denote the length of the shortest path in Q. The algorithm terminates if there is no further chance to find a path that is shorter than the current shortest path in Q. This is the case when the sum of depths of both BFS runs exceeds lshortest.

```
Algorithm TreeSketch(s, d)
Input: s, d ∈ V
Result: Q – priority queue of paths from s to d, ordered
by path length
1 begin
2 Load Sketch(s), Sketch(d) from disk
3 Ts ← tree of paths from s ⊲ taken from Sketch(s)
4 Td ← tree of paths to d ⊲ taken from Sketch(d)
5 Q ← ∅
6 lshortest ← ∞
7 VBfs ← ∅
8 VRBfs ← ∅
9 foreach u ∈ Bfs(Ts, s) and v ∈ RBfs(Td, d) do
10 VBfs ← VBfs ∪ {u}
11 $p_{v \to d}$ ← path from v to d in Td
12 foreach x ∈ VBfs do ⊲ iteration in order of visits
13 if v ∈ S(x) then ⊲ S(x) is set of successors of x in G
14 p ← $p_{s \to x} \circ (x, v) \circ p_{v \to d}$
15 Q ← Q ∪ {p}
16 lshortest ← min{lshortest, |p|}
17 VRBfs ← VRBfs ∪ {v}
18 $p_{s \to u}$ ← path from s to u in Ts
19 foreach x ∈ VRBfs do ⊲ iteration in order of visits
20 if x ∈ S(u) then ⊲ S(u) is set of successors of u 2 G
21 p ← $p_{s \to u} \circ (u, x) \circ p_{x \to d}$
22 Q ← Q ∪ {p}
23 lshortest ← min{lshortest, |p|}
24 if dist(s, u)+dist(v, d) ≥ lshortest then break
25 return Q
```

D.Methodology:

In order to evaluate the approximation quality and running times of our algorithms, I use a set of test triples of the form: (x, y, dist(x, y))
consisting of a pair of nodes x, y ∈ V and the actual distance dist(x, y) (length of shortest path) of these nodes in the graph. We generate these triples by uniformly sampling one hundred vertices and computing shortest path trees (forward and backward direction) for each vertex, using Dijkstra's algorithm. As output we obtain for every sampled vertex v one tree connecting the vertex to every other reachable node and one "reversed" tree, connecting every node for which a path to the sampled vertex exists to v.
As a result we obtain a set of triples of the structure shown in equation . Then, we group these triples into categories corresponding to the distance dist(x, y). From every such category, we sample at most 50 triples (tests) as our test set. The actual number of tests varies for every network because both the number of groups as well as the number of contained triples might be different.

1).Shortest Path Computation:

For each of the sampled seed sets Si and every node v ∈ V. I compute a shortest path
$p_{s_i} \to v$ that connects any member of the seed set to v (note that there could be more than one such path). I use breadth-first expansion from Si and build the complete shortest path tree. For every node v, we thus obtain the closest seed node, denoted by l1. Likewise, we compute a shortest path $p_v \to s_i$ that connects v to the seed set, using breadth-first expansion from Si on reversed edges, terminating at the first seed node, denoted by l2. The nodes l1, l2 are called *landmarks* of Si for the vertex v. This pre-computation routine – that is, seed set sampling and shortest path

computation – is repeated k times, thereby generating for each vertex v ∈ V a number of 2rk landmarks and paths (at costs of the same number of BFS expansions from the seed sets). Note that the set of selected landmarks might be different for every node. The data (sketch) gathered for node v, consisting of the 2rk landmarks and paths, is denoted by Sketch(v). As a result of the pre-computation step, we store the sketches of all nodes on disk.

2). Shortest Path Approximation:

In the second stage, the algorithm receives a pair of nodes, (s, d), as input. The goal is to compute, in real time, a path $p_{s\to d}$ from s to d that provides a good approximation of the shortest path, that is, a path with small error($p_{s\to d}$)The sketch algorithm, presented in Algorithm 1, performs the following steps to generate such a path:

1.Load the sketches of nodes s and d from disk

2. Let L be the set of common landmarks in the sketches. Note that for undirected graphs I can guarantee (for nodes in the same component) that there exists at least one such landmark, because the sketches of both s and d contain a path from (respectively to) the only member of the singleton seed set S0. In directed graphs this guarantee can only be given for nodes contained in the same strongly connected component.

3. For each common landmark l ∈ L, let $p_{s\to l}$ be the path from s to l and $p_{l\to d}$ the path from l to d. Construct (by concatenation) the path $p_{s\to d}$:= $p_{s\to l}$ ∘ $p_{l\to d}$from s to d through l, and add it to a priority queue (sorted in ascending order by path length).

4. Return the priority queue of paths obtained in step 3.

3) Cycle Elimination :

The paths returned by Algorithm 1 approximate the shortest path for the two query nodes and thus might be suboptimal, i.e. longer than the true shortest path. Some of the returned paths can however be easily improved, because they contain cycles. The modified sketch algorithm, named SketchCE, is described in Algorithm

4) Shortcutting :

The second modification I propose is path shortcutting: Suppose the queue Q returned by the algorithm contains a path $p_{s\to l\to d}$ from s to d via a landmark l. Two nodes u, v in the path might actually have a closer connection than the one contained in the respective subpath of $p_{s\to l\to d}$. Consider the example depicted below: While the nodes u and v are connected by a subpath of $p_{s\to l\to d}$ of length 3, the original graph contains the edge (u, v). I can then easily substitute this subpath by the single edge (u, v). Note that in all cases that allow for this shortcutting optimization, the landmark l will be located on the subpath from u to v. In order to find out whether any two nodes in a given path $p_{s\to l\to d}$ = $p_{s\to l}$ ∘ $p_{l\to d}$ are neighbors, I start at the first vertex, s, and load the list S(s) of its successors in the original graph. Then I check if any of the successors of s is contained in the path $p_{l\to d}$ from the landmark l to the destination vertex d. If this is the case, I can substitute the subpath from s to this node by the single edge. If no successor of s is contained in the path from l to d, I proceed to the second node in the path $p_{s\to l}$ load the set of its successors and repeat the procedure. I can terminate this routine if I either find a shortcut or arrive at the last vertex of the path $p_{s\to l}$, the landmark node l. In this case, the original path $p_{s\to l\to d}$ cannot be improved by shortcutting, because the path $p_{l\to d}$ from l to d is already guaranteed to be a shortest path (obtained using BFS in the preprocessing step). The complete algorithm (cycle elimination + shortcutting), called SketchCESC, is depicted in algorithm .

V. EXPECTED RESULTS

A. Approximation Quality

In order to assess the approximation quality of the paths generated by the different algorithms, I run for every triple s, d, dist(s, d) contained in the test set a shortest path query for all 3 proposed algorithms: Sketch, SketchCE, AND SketchCESC. For every shortest path query (s, d), I compare the length $l_{shortest}$ of the shortest path $P_{s\to d}$ in the returned queue with the true node distance dist(s, d) specified in the test triple. Then, I obtain the approximation error, error $P_{s\to d}$, for the path: error $P_{s\to d}$ = $l_{shortest}$ − dist(s, d) dist(s, d) ∈ [0,∞]. For every algorithm I record the average approximation error over all the test triples.

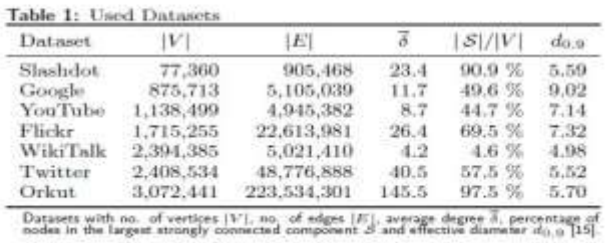


Table 1: Used Datasets

Dataset	$\|V\|$	$\|E\|$	$\bar{\delta}$	$\|\mathcal{S}\|/\|V\|$	$d_{0.9}$
Slashdot	77,360	905,468	23.4	90.9 %	5.59
Google	875,713	5,105,039	11.7	49.6 %	9.02
YouTube	1,138,499	4,945,382	8.7	44.7 %	7.14
Flickr	1,715,255	22,613,981	26.4	69.5 %	7.32
WikiTalk	2,394,385	5,021,410	4.2	4.6 %	4.98
Twitter	2,408,534	48,776,888	40.5	57.5 %	5.52
Orkut	3,072,441	223,534,301	145.5	97.5 %	5.70

Datasets with no. of vertices $|V|$, no. of edges $|E|$, average degree $\bar{\delta}$, percentage of nodes in the largest strongly connected component $\mathcal{S}$ and effective diameter $d_{0.9}$ [15].

B. Path Diversity

In many application settings it is not only important to quickly generate an accurate approximation of the shortest path, but also crucial to return as many candidate paths as possible. Our algorithms are designed in such a way that this goal can be satisfied. They return an ordered queue of paths which can – for example – be used to filter out candidates based on some user-specified constraints. The average number of generated shortest paths is given.

C. *Query Execution Time*

The second important evaluation category we assessing is query execution time. I compare the results of our methods, averaged over the test triples, to the average running time of Dijkstra's algorithm.

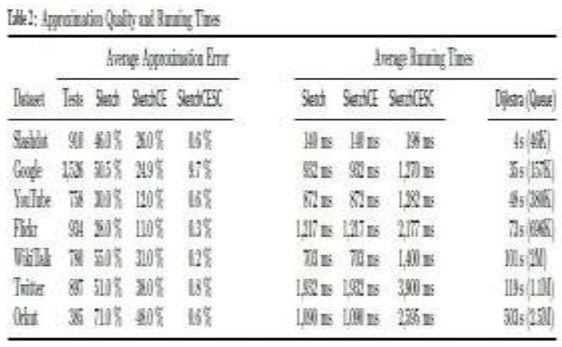


Table 2: Approximation Quality and Running Times

	Average Approximation Error				Average Running Times			
Dataset	Tests	Sketch	SketchCE	SketchCESC	Sketch	SketchCE	SketchCESC	Dijkstra (Queue)
Slashdot	918	40.0 %	20.0 %	0.6 %	140 ms	146 ms	198 ms	4 s (40K)
Google	1,536	30.5 %	24.9 %	0.7 %	932 ms	932 ms	1,270 ms	35 s (157K)
YouTube	758	30.0 %	12.0 %	0.6 %	672 ms	672 ms	1,282 ms	49 s (380K)
Flickr	934	20.0 %	11.0 %	0.3 %	1,217 ms	1,217 ms	2,177 ms	73 s (696K)
WikiTalk	798	55.0 %	31.0 %	0.2 %	700 ms	700 ms	1,400 ms	101 s (2M)
Twitter	897	51.0 %	30.0 %	0.8 %	1,932 ms	1,932 ms	3,900 ms	119 s (1.1M)
Orkut	385	71.0 %	40.0 %	0.6 %	1,090 ms	1,090 ms	2,385 ms	503 s (2.5M)

D. Pre-computation Time.

We measure the running time of the preprocessing stage for all datasets. It provides an overview over the required times for k = 2 preprocessing iterations. Note that the required time increases linearly in k. For small datasets like Slashdot, the sketches can be obtained within five minutes, while the largest dataset requires about 11 hours of preprocessing.

VI. CONCLUSION

Here we have discussed about Sketch-based Index structure. Beat the classical Dijkstra's shortest path algorithm on the same pre-computed data yield paths with excellent quality. Practicability of proposed algorithms in terms of speed, quality, and diversity. We significantly reduce the distance estimation error, compared with global landmark embedding. The results also confirmed the effectiveness and efficiency of our solution.

REFERENCES

[1] R. C. Prim. "Shortest Connection Networks and some Generalizations," Bell System Technology Journal, 36:1389–1401, 1957.

[2] P.E. Hart, N.J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Trans. Systems Science and Cybernetics, vol. SSC-4, no. 2, pp. 100-107, July1968.

[3] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guideto the Theory of NP-Completeness," W.H. Freeman, 1979.

[4] F. B. Zhan and C. E. Noon. "Shortest Path Algorithms: An Evaluation using Real Road Networks," Transportation Science, 32(1):65–73, 1998.

[5] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," IEEE/ACM Trans. Networking, vol. 9, no. 5, pp. 525-540, Oct. 2001.

[6] C. Shahabi, M. Kolahdouzan, and M. Sharifzadeh, "A RoadNetwork Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases," Proc. 10th ACM Int'l Symp. Advances in Geographic Information Systems (GIS '02), pp. 94-100, 2002.

[7] A.V. Goldberg and C. Harrelson, "Computing the Shortest Path: A* Search Meets Graph Theory," Proc. 16th Ann. ACM-SIAMSymp. Discrete Algorithms (SODA '05), pp. 156-165, 2005.

[8] A.V. Goldberg, H. Kaplan, and R.F. Werneck, "Reach for A*:Efficient Point-to-Point Shortest Path Algorithms," Proc. SIAMWorkshop Algorithms Eng. and Experimentation, pp. 129-143, 2006.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. "Introduction to Algorithms," MIT Press, 3rdedition, 2009.

[10] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum, "Fast and Accurate Estimation of Shortest Paths in Large Graphs," Proc. 19thACM Int'l Conf. Information and Knowledge Management (CIKM '10),pp. 499-508, 2010.

[11] Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu,"Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme," IEEE Transaction on Knowledge and Data Engineering, Vol. 26, NO. 1, January 2014.

[12] R. C. Prim. Shortest Connection Networks and some Generalizations. Bell System Technology Journal, 36:1389–1401, 1957.

[13] M. Thorup and U. Zwick. Approximate Distance Oracles. In STOC'01: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, pages 183–192. ACM, 2001