Label Constrained Shortest Path Estimation

Ankita Likhyani
Indraprastha Institute of Information Technology
New Delhi, India
ankital@iiitd.ac.in

Srikanta Bedathur
Indraprastha Institute of Information Technology
New Delhi, India
bedathur@iiitd.ac.in

ABSTRACT

Shortest path querying is a fundamental graph problem which is computationally quite challenging when operating over massive scale graphs. Recent results have addressed the problem of computing either exact or good approximate shortest path distances efficiently. Some of these techniques also return the path corresponding to the estimated shortest path distance fast.

However, none of these techniques work very well when we have additional constraints on the labels associated with edges that constitute the path. In this paper, we develop SkIt index structure, which supports a wide range of label constraints on paths, and returns an accurate estimation of the shortest path that satisfies the constraints. We conduct experiments over graphs such as social networks, and knowledge graphs that contain millions of nodes/edges, and show that SkIt index is fast, accurate in the estimated distance and has a high recall for paths that satisfy the constraints.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures—*Graphs and Networks*; H.2.4 [**Database Management**]: Systems—*Query Processing*

Keywords

Graph Databases; Shortest Paths; Edge-label constraints

1. INTRODUCTION

Finding shortest paths between two given nodes in a graph is a problem of fundamental importance in computer science. It has a variety of applications ranging from network-routing [4] to its use as a data mining primitive [2]. Computing shortest paths in an online manner for each query pair over massive graphs is computationally challenging. This has given rise to recent research in efficiently estimating and computing the shortest path distances by preprocessing the graph. Some of these methods can be extended to generate corresponding shortest paths themselves efficiently.

As the modelling of networks gets richer, we have graphs that have certain properties associated with nodes and edges in the form of labels. For instance, in a large knowledge graph such as Yago, the relationship between two entities has one of a possible canonicalized relationship identifiers, such as "hasNeighbor", "happenedIn", "influences", etc. Even in social networks, it has become common to have edge-labels like "Friend", "Colleague", "Family", and so on, to distinguish the nature of relationship between people. If we consider general RDF databases that form the substrate for Semantic Web efforts, all relationships are *named* – i.e., contain labels.

As a natural consequence, many modern practical uses of shortest path computation demand certain constraints to be placed on the labels over edges that are involved in the path. Current solutions, which do not explicitly consider these labels during the distance/path computation, have to resort to enumerating of all paths in increasing order of their length until the path that satisfies the constraint is found, if at all it exists.

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2507818.

1.1 Contribution

In this paper, we consider how to answer these label constrained shortest path queries efficiently by augmenting the landmark-based path-sketches [5]. The resulting index, called SkIt (Sketches augmented with Inverted indexes), enables an efficient estimation algorithm for edge-label constrained shortest paths between two given nodes of an entirely disk-resident graph. In this poster, we demonstrate how SkIt can efficiently support the following forms of constraints on the edge-labels:

Label white-listing: The set of edge-labels on the qualifying paths should be a subset of the specified white-list of edge-labels.

Label black-listing: As opposed to above, user specifies the set of labels that must not appear on the qualifying paths.

However, it is worth noting that the proposed solution can also handle regular expression-like richer label constraints on paths.

In order to empirically establish the efficiency of SkIt, we implement it within RDF-3X database, the same framework that was used in path-sketches [5]. We compare the performance of SkIt against standard TreeSketch that approximately enumerates the paths in increasing order of their distance. We also compare with the path query performance in Neo4J [1] – a high-performance, industry-standard graph data management system. As a baseline, we also implement the standard Dijkstra's algorithm which can trivially support all forms of label constraints as it explores the graph. Our evaluation using multiple large-scale labeled graphs show that the use of SkIt makes complex label-constrained shortest path discovery highly scalable.

1.2 Problem Statement

Let $G = (V, E, \Sigma)$ denote a directed graph with vertex set V, edge set E, edge label set Σ. A path p from vertex u to v is an alternate-sequence of (distinct) vertices and edges i.e., $p = (u, e_1, v_1, ..., v_{i1}, e_i, v_i, ..., e_n, v)$.

DEFINITION 1 (LCSP). *Given an edge-labeled directed graph, $G = (V, E, \Sigma)$, a source vertex s, a target vertex t and an edge-label constraint set $C \subset \Sigma$, a label-constrained shortest path, $LCSP(s, t, C)$ is given by the shortest path p between s and t such that $L(p) \subseteq C$, where $L(p)$ are the edge labels involved in the path p. In case of black-listed label set $\bar{C}$, the edge-label constraint set is given simply by $C = \Sigma \setminus \bar{C}$.* □

From the definition above, it is clear that the edge-label white-listing and black-listing can be treated uniformly. It is also quite straightforward to see that, in practice, the value of $|C|$ is smaller in case of white-listing than in the edge-label black-listing setting.

1.3 Problem Difficulty

While the introduction of constraints on the edge-labels poses no problems to the Dijkstra's algorithm and its adaptations (such as its bidirectional variant and A*-search), it is well-known that these algorithms do not scale for graphs with millions of nodes and hundreds of millions of edges. At the same time, to the best of our knowledge, none of the existing fast shortest-path approximation techniques have a way of incorporating edge-labels within them. The simple alternative of considering all possible edge-label constraints and build separate shortest-path oracle is not suitable when we have a large number of edge-labels on account of $2^{|\Sigma|}$ combinations to deal with. Thus there is a clear need for a data structure that can support LCSP queries in a scalable manner, even on graphs that do not have favorable structural properties such as near-planarity.

The rest of the presentation is structured as follows: in the following section, we briefly describe the landmark-based shortest path estimation approach and the notion of path-sketches, which form the basis for our work. In Section 3 we describe how path-sketches are augmented with edge-labels to result in SkIt structure, and how to answer a LCSP query over it. Following this, in Section 4 we describe our experiment setup including datasets and queries, and present the results. Finally, we conclude in Section 5.

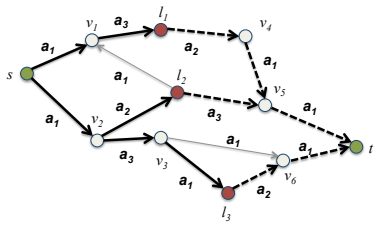
2. PATH-SKETCHES

One of the most popular shortest path estimation techniques over directed, unweighted graphs (such as those we consider here), is the landmark-based shortest path oracles [7]. Although there are many different variants, the underlying idea is as follows: We first select a set of special nodes, L, which we call *landmarks*, and compute shortest path distances to/from each of the landmarks to all other nodes in the graph. We associate a vector of these distances along with the corresponding landmark with every node, and call this a *shortest-path distance sketch* of the graph. After this pre-computation, we can estimate the shortest path between any two nodes u and v using triangle inequality as follows:

$$\hat{\delta}(u, v) = \min_{l \in L} \delta(u, l) + \delta(l, v),$$

where $\delta(x, y)$ denotes the directed shortest distance between x and y. One variant of this method [3], proposed to use set-based landmark selection, where we sample landmark sets of exponentially increasing sizes $S_1, \ldots, S_r$ with $r = \log(n)$. For each such set, every vertex v maintains two sketch entries: the distance to the *closest* landmark node in the set corresponding to the *forward* path from the node to a landmark set, and the distance from a landmark node from which v is at shortest distance corresponding to the *backward* path from a landmark to the node. The sketch for a vertex v is:

$$\begin{aligned} s_f(v) &= \{(f_i, \delta(v, f_i)) \mid i = 1 \ldots r\} \\ b_f(v) &= \{(b_i, \delta(b_i, v)) \mid i = 1 \ldots r\} \end{aligned}$$



node	b/f	l	δ	p
s	f	l_1	2	$\langle s, a_1, v_1, a_2, l_1 \rangle$
t	b	l_1	3	$\langle l_1, a_2, v_4, a_1, v_5, a_1, t \rangle$
..	..	..	..	..

Figure 1: Labeled Path-sketch

where $f_i = \arg\min_{x \in S_i} \delta(v, x)$ and $b_i = \arg\min_{x \in S_i} \delta(x, v)$. Denoting the landmark nodes that appear in the forward and backward sketches of a node v as $L_f(v)$ and $L_b(v)$, respectively, we can write the shortest path distance estimator between two nodes u and v as,

$$\hat{\delta}(u, v) = \min_{x \in L_f(u) \cap L_b(w)} \delta(u, x) + \delta(x, v).$$

Note that these distance sketches maintain only the shortest distance between landmark nodes and regular nodes in the graph. Therefore, in order to reconstruct the actual path that corresponds to the estimated distance between the two nodes, additional accesses to the graph are required. This requirement is lifted by *path-sketches* [5] where the sketch also contains the shortest path between the landmark node and a regular node. That is, the path-sketch entries are of the form

$$\begin{aligned} s_f(v) &= \{(f_i, \delta(v, f_i), p(v, f_i)) \mid i = 1 \ldots r\} \\ b_f(v) &= \{(b_i, \delta(b_i, v), p(b_i, v)) \mid i = 1 \ldots r\} \end{aligned}$$

More significantly, based on this additional information, it is possible to improve the accuracy of the shortest path distance estimation significantly through the use of a series of improvements culminating in a bounded path computation algorithm called *TreeSketch* over the tree resulting from the union of paths stored in the sketches. For every node in the tree, *TreeSketch* performs a k-hop BFS (typically $k = 1$) to see if it is possible to reach the target with a shorter distance. Although TreeSketch requires additional accesses to the underlying graph, it has been shown to be relatively inexpensive even when the graph is entirely disk-resident [5], and can generate almost accurate distance estimates.

3. SKIT INDEX

As a first step towards extending path-sketches to support the edge-label constraints on the paths, we first augment the path information with the edge-label information. This fairly straightforward extension is illustrated using the example graph in Figure 1, along with the associated sketch. In this figure, the green nodes represent the query vertices s and t in the graph, and the red nodes indicate the landmarks that are common between the two query nodes. The thick solid edges are the paths stored in the forward path-sketches of s, while the thick dashed edges are the paths of the backward path-sketches of t.

With path-sketches thus augmented, we can proceed to estimate the LCSP as follows: load forward/backward sketches for s and t query nodes from disk, reject all sketches containing paths which do not satisfy the specified edge-label constraints, and similarly reject total distance estimations if the corresponding path violates

the constraints. Further, one observes that since the labels are stored in the order of their occurrence in the path, it is possible to include edge-label order constraints as well easily.

However, this simplistic approach may result in not finding any path between two nodes since we are pruning out candidate paths. For instance, consider the backward path-sketch starting from l_3 to t, whose edge-label set is $\{a_1, a_2\}$. If the user specified $C = \{a_1, a_3\}$ then rejecting the path-sketch entirely would result in not finding the LCSP between s and t, although TreeSketch allows for it. On the other hand, retaining all the sketches which violate the constraints also is not desirable as it adds needless exploration steps.

We tradeoff these two aspects by truncating the paths from sketches to a prefix (suffix for backward paths) that satisfies the edge-label constraints. For instance, considering the same example, we will truncate the sketch $\langle l_3, a_2, v_6, a_1, t\rangle$ to $\langle v_6, a_1, t\rangle$. Using TreeSketch, we can now find the LCSP between the query vertices.

Speeding up TreeSketch with Label Inverted Index.

During the execution of TreeSketch, for every node we perform a k-hop exploration of the graph in order to find a shorter connection between the forward and backward sketch trees. However, this may lead to many wasted accesses to the graph since they may violate label constraints.

We utilize a compact inverted list structure that, for each node, maintains the list of nodes directly reachable, as well as the list of nodes of that reach the current node, via an edge-label. In other words, each node will have a *forward edge-label* and a *backward edge-label* inverted list. This can be seen as an effective organization of the edge-labeled adjacency list of the graph so as to efficiently decide if a specific node can be expanded or not.

Constrained TreeSketch Algorithm.

Next we briefly describe the Edge-label Constrained TreeSketch algorithm, presented in Algorithm 1, as an extension of TreeSketch [5]. In its first step, the algorithm loads all the path-sketches for the two given pair of vertices s and t, retaining the prefix (or the suffix) that satisfies the edge-label constraint. This results in a edge-label constrained sketch tree, which we denote as T_s^C and T_t^C. Then, a bidirectional BFS is initiated from s and t. As each node is visited, its neighbors which satisfy the label constraints are loaded using its inverted index list. This local BFS step can proceed upto k-hops – in the algorithm listing we present the situation when $k = 2$. Note that if we perform 2-hop BFS from nodes in T_s^C and T_t^C we can discover shortcuts upto 3-hops.

Algorithm 1: Edge-label Constrained TreeSketch (s, t, C)

Input: $s, t \in V, C \subset \Sigma$
Result: Q, priority queue of paths from s to t respecting C

$CT_s \leftarrow$ tree of paths from s that respect C
$CT_t \leftarrow$ tree of paths to t that respect C
$Q \leftarrow \emptyset$
$\mu \leftarrow \infty$
$NBFS \leftarrow \emptyset, NRBFS \leftarrow \emptyset$
foreach $u \in BFS(CT_s, s)$ ***and*** $v \in BFS(CT_t, t)$ **do**
  $NBFS \leftarrow NBFS \cup \{u\}$
  $p_{v\to t} \leftarrow$ path from v to $t \in CT_t$
  foreach $x \in NBFS$ **do**
    if $v \in$ *upto2HopSuccessors*(x, C) **then**
      $p \leftarrow p_{s\to x} \circ p_{x\to v} \circ p_{v\to t}$
      $Q \leftarrow Q \cup \{p\}$
      $\mu \leftarrow min\{\mu, |\ p\ |\}$
  $NRBFS \leftarrow NRBFS \cup \{v\}$
  $p_{s\to u} \leftarrow$ path from s to $u \in CT_s$
  foreach $x \in NRBFS$ **do**
    if $x \in$ *upto2HopSuccessors*(u, C) **then**
      $p \leftarrow p_{s\to u} \circ p_{u\to x} \circ p_{x\to t}$
      $Q \leftarrow Q \cup \{p\}$
      $\mu \leftarrow min\{\mu, |\ p\ |\}$
  if $dist(s, u) + dist(v, d) \geq \mu$ **then**
    return

DataSet	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\Sigma\rvert$	$\overline{\lvert L_f\rvert}$	$\overline{\lvert L_b\rvert}$
Orkut	3,072,441	117,185,083	324	39.4	31.4
socLive	4,847,571	68,993,773	315	40.1	38.5
Yago	14,395,591	30,717,443	96	1.47	5.92

Table 1: Dataset Characteristics

4. EXPERIMENTAL EVALUATION

We implemented all methods within the original RDF-3X-based implementation made available to us by the authors of path-sketches [5]. Due to lack of space, we will not give details of the RDF-3X graph database system, instead direct the interested reader to [6]. Using the highly compressed triple storage within RDF-3X, we store graphs edgewise with each edge represented as a triple $\langle s, label, t\rangle$. Path-sketches are also stored in triple format in a separate RDF-3X database as follows: $\langle v_i\rangle\langle t\rangle\langle l_{ij} : p_{ij}\rangle$ for forward sketches and $\langle v_i\rangle\langle f\rangle\langle l_{ij} : p_{ij}\rangle$ for backward sketch, where v_i is the source node, l_{ij} is the landmark for the node from landmark set S_j, and p_{ij} is the path between v_i and l_{ij} augmented with edge-labels.

Experimental Setup.

We made use of two systems in our experiments: a server-class machine, termed SYSTEM-S, with Intel Xeon CPU E5-2640 @ 2.50GHz with 64GB RAM running OpenSUSE Linux, and a desktop, termed SYSTEM-D, with Intel i3 CPU 550 @ 3.2GHz with 4GB RAM running Linux Mint. The algorithms are evaluated on three datasets: (a) Orkut[1], (b) socLive[2], and (c) Yago[3]. Both Orkut and socLive are social networks recorded without any edge-label information. We synthetically labeled the edges on these two networks with edge-labels following exponential distribution with exponent 0.5. On the other hand, Yago is a large entity-relationship network with every edge labeled using one of 96 labels. The relevant statistics of these three networks are given in Table 1. The table also lists average number of forward and backward landmark sets, denoted as $\overline{|L_f|}$ and $\overline{|L_b|}$ respectively, for each node that are stored in the path-sketches. These numbers show that the total number of sketches for each node we need to read, is fairly small for all datasets.

In addition to RDF-3X based implementations, we also evaluated the performance of constrained shortest path queries over Neo4J [1] a state-of-the-art graph data management system.

We have performed experiments on two types of queries:
(1) **Positive label restrictions**: Given a set of edge-labels $C \subset \Sigma$, the path label set $L(p) \subset C$, this query set is constructed as follows: first we select paths from a tree that is a union of BFS trees started simultaneously with 100 nodes, chosen at random with degree more than average-degree, as root nodes. We repeated this process 3 times, and from from the resulting collection of queries, for each path length at most 50 queries are selected, with different source and target pairs;
(2) **Negative label restrictions**: Given a set of edge-labels $C \subset \Sigma$, the path label set $L(p) \nsubseteq C$, this query set is constructed by selecting a source node, a target node and constraint set uniformly

[1]http://snap.stanford.edu/data/com-Orkut.html
[2]http://snap.stanford.edu/data/soc-LiveJournal1.html
[3]http://www.mpi-inf.mpg.de/yago-naga/yago/

Dataset	Path-sketch	SkIt	Neo4J (load)
Orkut	17323	28650	8765
socLive	21647	29568	5236
Yago	11529	23148	104

Table 2: Index Construction Time (in seconds)

Dataset	Query	SkIt-1hop		SkIt-2hop		TreeSketch	
		ϵ	τ	ϵ	τ	ϵ	τ
Orkut	+ve	0.0073	0.163	0.0045	0.1227	0.0112	0.088
	-ve	0.1197	0.147	0.0437	0.0568	0.1325	0.136
socLive	+ve	0.00	0.082	0.00	0.056	0.0076	0.024
	-ve	0.0390	0.0042	0.0051	0	0.0392	0.0084
Yago	+ve	0.0108	0.0759	0.0106	0.0629	0.0111	0.0724

Table 3: Effectiveness of SkIt in Estimating LCSP

at random, with constraint set size varying from 1 to 3. For each constraint set size we generated 100 queries.

Evaluation Metrics.

- **Approximation error (ϵ):** If the shortest path that satisfies the given constraints has length l^c and the estimated constrained path length has length $\hat{l^c}$, then we report $\frac{|l^c - \hat{l^c}|}{l^c}$.
- **False negative ratio (τ):** The fraction of queries which fail to return any path that satisfies the given constraint, although at least one such path exists.
- **Query execution time:** We compare the performance of our SKIT-based LCSP estimation technique against the performance of traditional Dijkstra's algorithm and standard TreeSketch algorithm. Note that for TreeSketch algorithm, we need to traverse through the heap of paths until a path that satisfies the given constraints is found.

Finally, we also compare query execution time against the constrained shortest path function available in Neo4J (`http://www.neo4j.org`) – an open-source, enterprise-grade, high-performance graph management system.

4.1 Results

We computed all indexes over the SERVER-S machine, and the time taken to construct them are reported in Table 2. The construction time reported for SkIt index are much higher than the time taken for path-sketches since we include the time taken to compute the edge-label inverted indexes in a separate round. In comparison, Neo4J loads the data extremely fast – most likely because it stages data load entirely within memory and schedules disk-writes lazily. On the other hand, all the results we report are over databases and indexes stored entirely on disk.

Average approximation error and false negative ratio are reported in Table 3. The average approximation error for SkIt-1hop and SkIt-2hop is always less than that for TreeSketch. This is particularly noticeable in social network datasets where we see an order of improvement in approximation error of LCSP over TreeSketch. The false-negative ratio of SkIt is slightly inferior to TreeSketch, due to relatively aggressive pruning of sketches. We have reported only average running for negative queries over Yago, since even Dijkstra's algorithm failed to return any paths for many choices of negative label restrictions we tried.

Figure 2 shows average query execution time over SYSTEM-D as well as SYSTEM-S in logarithmic scale. Neo4j, being largely an in-

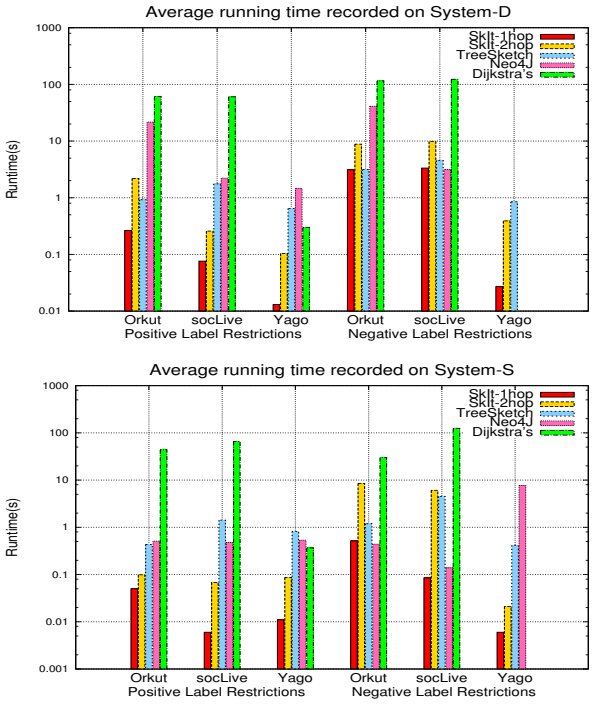


Figure 2: Running Time recorded on different machines

memory graph database performs significantly better on SYSTEM-S as compared to SYSTEM-D. Moreover, it ran out of memory for negative queries over Yago on System-D. Nevertheless, SkIt-1hop outperforms it by almost an order of magnitude over large graphs over both SYSTEM-S as well as SYSTEM-D. This is particularly surprising considering that the SkIt index entirely resides on disk with minimal memory footprint.

5. CONCLUSION

In this paper, we have presented SkIt index for effective and efficient estimation of edge-label constrained shortest paths. Through experiments on large-scale graphs we demonstrate that SkIt outperforms its competitors for both white-listing and black-listing of edge-labels. In continuation of this work, we plan to support even richer set of label constraints, and also reduce the size of SkIt index.

Acknowledgements This work is supported by the Max Planck Society (MPG) and the Dept. of Science and Technology (DST), India under the joint cooperation scheme of Max Planck Partner Group for Large-scale Graph Mining.

6. REFERENCES

[1] Neo4J, the Graph Database. `http://www.neo4j.org`, 2013. [Online; accessed 31-May-2013].

[2] B. V. Cherkassky, L. Georgiadis, A. V. Goldberg, R. E. Tarjan, and R. F. F. Werneck. Shortest-path feasibility algorithms: An experimental evaluation. *ACM Journal of Experimental Algorithmics*, 14, 2009.

[3] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In WSDM, 2010.

[4] D. Delling, A. V. Goldberg, and R. F. F. Werneck. Shortest paths in road networks: From practice to theory and back. *IT - Information Technology*, 53(6):294–301, 2011.

[5] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In CIKM, 2010.

[6] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1):91–113, 2010.

[7] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.