International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-4, November 2019

Implicit Landmark Path Indexing for Bounded Label Constrained Reachable Paths

Bhargavi B, K Swarupa Rani

Abstract:** In today's Big Data era, a graph is an essential tool that models the semi-structured or unstructured data. Graph reachability with vertex or edge constraints is one of the basic queries to extract useful information from the graph data. From the graph reachability with constraints, we obtained the information about the existence of a path between the given two vertices satisfying the vertex or edge constraints. The problem of Label Constraint Reachability (LCR) found the existence of a path between the two given vertices such that the edge-labels along the path are the subset of the given edge-label constraint. We extended the LCR queries by considering weighted directed graphs and proposed a novel technique of finding paths for LCR queries bounded by path weight. We termed these paths as bounded label constrained reachable paths (BLCRP). We extended the landmark path indexing technique [1] by incorporating the implicit paths which satisfy the user constraints but need not satisfy the minimality of edge label sets. We solved the BLCRP by using the extended landmark path indexing and BFS based query processing. We addressed the following challenges through our proposed technique of implicit landmark path indexing in the problem of BLCRP that included (1) the need to handle exponential number of edge label combinations with an additional total path weight constraint, and (2) the need to discover a technique that finds exact reachable paths between the given vertices. This problem could be applied to real network scenarios like road networks, social networks, and protein-protein interaction networks. Our experiments and statistical analysis revealed the accuracy and efficiency of the proposed approach tested on synthetic and real datasets.

***Keywords:** constraint reachability, label constraint, landmark vertex, weighted directed graph*

I. INTRODUCTION

In the Big data era, there are huge databases with relational and graphical information. The discovery of useful and unknown information in big data is challenging for research
and professional groups. The graph is one of the important tools that can represent the complex relationships between the objects of big data. Graph mining refers to mining data represented as a graph [2]. Some of the specific operations of mining graph data from real-world domains such as finding matching graph patterns, clustering graphs, and finding frequent subgraphs are the important contributions of deriving new knowledge.

Manuscript published on November 30, 2019.
* Correspondence Author

Bhargavi B, School of Computer and Information Sciences, University of Hyderabad, Hyderabad, India. Email: bhargavibbv@uohyd.ac.in
K Swarupa Rani, School of Computer and Information Sciences, University of Hyderabad, Hyderabad, India. Email: kswarupaprasad@gmail.com

One of the fundamental operations to manage graph data is to find the reachability from one vertex to another vertex in the graph. In real-time, the vertices and edges of a graph consists of attributes. These attributes give information about the type of vertices, type of relationship, and strength of the relationship between the vertices. For instance, in protein association networks [3], the graph attributes are edge weights that represent the reliability of an interaction between two proteins and edge labels that are enzymes that transform the proteins. In social networks, the vertex labels denote the name of the user and the organization of the user; the edge labels between the users represent the relationships like friends, relatives and colleagues. Thus, the *edge weight* denotes the weight assigned to each edge of the graph. The *edge label* denotes the label assigned to each edge, and *vertex label* denotes the label assigned to the vertex. The constrained reachability query finds the existence of reachability between the two given vertices that satisfies the given constraints. The Label-Constraint Reachability (LCR) query which was first formally defined by Ruoming Jin et al. [4] is for finding the existence and non-existence of the label-constrained reachable paths in an edge labeled directed graph. We extended the LCR query by including the bound for path weights, in an edge labeled weighted directed graph and formulated the problem of Bounded Label Constrained Reachable Paths formally defined in Section II.

A. Applications

For instance, in road networks, we can find the paths between two places X and Y bounded by the given distance d, which are connected through labeled roads. Fig. 1 illustrates a local road network with vertices denoting the locations and edges representing the existence of road between the adjacent locations.

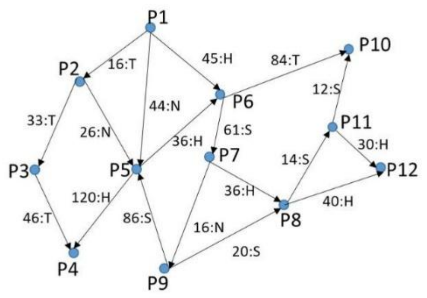

Fig. 1. Road Network, G1 with distances (Km.) and types of roads (S: Street, N: Narrow, H: High-way, T: Two-wheeler only) Each edge has a label that constitutes two parts W: Ty; W denotes the distance between the two adjacent locations and Ty denotes the type of roads between them.

For instance, the type of roads are assumed to be High-way (H), street-road (S), Narrow road (N) and Two-wheeler road (T) (accessible by only two-wheeler). Suppose the user query is to find the paths from "P1" to "P4" within distance of 100Km in a two-wheeler without using high-way. The query can be considered as the problem of finding bounded label constrained reachable paths in Fig. 1, from "P1" to "P4" with label set constraint {S, N, T} and the bound for path weight 100.

B. Techniques

Many reachability techniques proposed like 2-hop [5], 3-hop [6], and interval based cover [7] cannot solve LCR queries directly as they do not preserve the edge label information during index construction. Valstar et al. [8] proposed the technique of landmark based query processing to find the existence of reachability for LCR queries. We extend the LCR problem by appending edge weights and finding the actual paths that are bounded by path weight between the two given vertices for labeled weighted directed graphs. In addition to the reachable vertex and the path label, we include intermediate vertices and path weights during landmark index construction to find the bounded label constrained reachable paths.

C. Motivation

One of the significant challenges in finding the bounded label constrained reachable paths is that there can exist an exponential number of label combinations between the two given vertices. Another challenge is to compute the exact paths. But, in the literature, shortest path finding techniques [9] do not compute the exact paths. These observational studies persuade us towards the real network context that constitutes the categorical edge label constraints and real-valued edge weight constraints and thus we need to find the exact paths within the given maximum path-weight.

This paper is a revised and expanded version of our ICACDS paper [1]. In this paper, we have focused on edge labels and edge weights constraints for weighted labeled directed graphs. We address the following two challenges despite the slower index construction time: (1) The edge labels must satisfy the given membership constraint while finding graph reachability. (2) We find not only the reachability between the two given vertices but also the exact paths whose path weight is less than or equal to the given maximum path weight.

Based on the above observations, we formulate the problem and solve bounded paths for LCR [1] queries through landmark based path indexing that involves selecting a subset of vertices as landmark vertices. We extend the Landmark Index and Query algorithms [8] to find bounded label constrained reachable paths. In [1], for the landmark vertices, we constructed the landmark path index by extending the landmark based indexing approach proposed by Valstar et al. [8]. We extended by incorporating paths and path weights in the index in addition to destination vertex and labels. By including minimality of label sets [8], the landmark path index is constructed [1]. But, we observed that some of the implicit paths were not included in the index that satisfy the given constraints. In order to strengthen the efficiency of our proposed approach [1], in this paper, we modify the proposed LWPathIndex algorithm [1] and improve it by constructing an index that considers the implicit paths for the specific cases. Thus, the resultant path index does not necessarily satisfy the minimality of label sets for all the cases. To evaluate the accuracy of our proposed technique, we compute precision and recall for the resultant bounded label constrained reachable paths. Besides, we perform statistical analysis to infer the efficiency of our proposed approach on different datasets.

In summary, our main contributions in this work are:-

- Proposed LWPathIndexImplicit and BImplPath algorithms by including the implicit paths while indexing to find bounded label constrained reachable paths described in section IV.
- Theoretically proved the correctness of our proposed algorithms in section IV.
- Evaluated the accuracy by computing the precision and recall for all the queries of the real and synthetic datasets in section V.
- Statistically proved the efficiency of our proposed approach on datasets in section V.
- Presented extensive survey of constrained reachability techniques, which is given in Table I and conducted experiments on real and synthetic benchmark datasets to compute bounded label constrained reachable paths.

In section II, we describe and illustrate the preliminaries that are helpful in understanding our research. In section III, we describe the techniques in the literature related to reachability, constrained reachability, and finding paths. In section IV, we describe our contributions, which constitutes the improved landmark path indexing by including implicit paths and query processing algorithms. Section V describes the index construction, evaluation of results based on precision and recall measures, and statistical analysis on datasets. In section VI, we conclude and describe the scope for further research.

II. PRELIMINARIES

An edge-labeled directed graph is denoted by G (V, E, T_l, λ), where V is the set of vertices, E is the set of edges, T_l is the set of edge labels, and λ is the function that assigns each edge e∈ E, a label $\lambda(e) \in T_l$. We describe the path p from vertex 'vs' to 'vd' in the edge labeled directed graph G as a vertex sequence, i.e., $p=(vs, v_{m1}, \ldots, v_{mi}, \ldots vd)$. v_{mi} indicates the ith intermediate vertex along the path from 'vs' to 'vd'. We use path label $L(p)$ to denote the set of all edge labels in the path p, i.e., $L(p) = \{ \lambda(e_1) \cup \lambda(e_2) \cup \ldots \lambda(e_n)\}$.

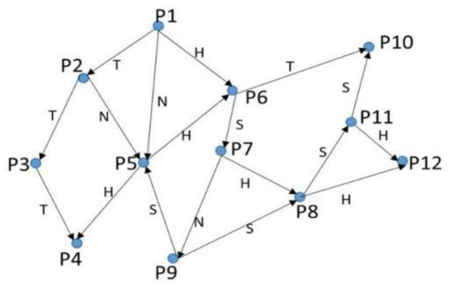

Fig. 2. Road Network with types of roads (S: Street, N: Narrow, H:High-way, T:Two-wheeler only)

DEFINITION 1. (Label-Constraint Reachability) *"Given two vertices, 'vs' and 'vd' in the edge labeled directed graph G, and a label set A, where 'vs', 'vd' $\in$ V, and A $\subseteq$ T_l, if there is a path p from vertex 'vs' to 'vd' whose path label L(p) is a subset of A, i.e., L(p)$\subseteq$A, then we say 'vs' can reach 'vd' with label-constraint A. We also refer to path p as an A-path from 'vs' to 'vd'."*[4]

LCR Query: "Given two vertices 'vs' and 'vd', and a label set A, the label-constraint reachability (LCR) query asks if there exists an A-path from 'vs' to 'vd'."[4]

We demonstrate the LCR query for road network in Fig. 2 through the following cases:

Case 1: For instance, the LCR query (*P1, P4, {H, N}*) returns true as there exists a path {P1, P5, P4}.

Case 2: For the LCR query (*P1, P7, {T, N}*), it returns false because there doesn't exist any path from *P1* to *P7* satisfying the given label-set constraint.

Case 3: There can exist a reachability query (*P1, P7,{T, N}*) for which there exists a path *p* = {P1, P6, P7} for which L(*p*) is not a subset of the given label set, i.e. L(*p*)={H, S}$\not\subseteq$ {T,N}.

Case 4: There can exist a reachability query (P3, P1, {T, N}) for which there does not exist any path in the entire graph even without satisfying the given label set constraint.

In this paper, we handle LCR queries of type case 1 and case 2. The type of reachability queries in the remaining cases are beyond the scope of the paper.

DEFINITION 2. (Landmark Vertex) The landmark vertices (denoted by V_L) are defined as the subset of vertices for the given graph G(*V*, *E*), i.e., $V_L \subseteq V$. These are selected based on criteria such as the top 'k' vertices of highest degree or any of the centrality measures of the graph. For instance, in Fig. 2, the top 4 vertices of highest total degree which can be considered as landmark vertices are {P5, P6, P8, P1}.

Landmark based query processing [8]: This technique involves selection of subset of vertices based on highest degree criterion. It involves full index construction for the landmark vertices by considering minimality of label sets [8]. The index for the landmark vertex includes all possible reachable vertices along with edge labels. For each non-landmark vertex, a similar index is constructed that considers up to 'b' reachable landmark nodes. Using these indexes and BFS, the LCR query is checked for the existence of reachability.

A. Problem Definition of BLCRP

We denote an edge labeled weighted directed graph as $G^{'}(V, E, T_l, \lambda, w)$ where *V* represent vertex set, *E* denotes the edge set, T_l is the total set of different labels in $G^{'}$ and for every edge e $\in E$, w(e) $\in R^+$ and λ (e) $\in T_l$. The path weight or path cost is computed by adding edge weights ($w(e_i)$) along the path (*p*). Thus, the path weight for the path p denoted by C(p) is $\Sigma(w(e_i))$.

We proposed the Bounded Label Constrained Reachable Paths problem [1] by extending the LCR definition proposed by Ruoming Jin et al. [4] as follows:

DEFINITION 3. (Bounded Label Constrained Reachable Paths) "*Given two vertices 'vs' and 'vd', the label set A $\in T_l$ and bound for the path-weight $\delta \in R^+$ in an edge-labeled weighted directed graph $G^{'}$, if there is an A-path lp_i from 'vs' to 'vd' such that the path weight $C(lp_i) \leq \delta$, then we say 'vs' can reach 'vd' with label-constraint A and the path weight bound δ.*"[1]

In other words, it can also be referred as follows: Given two vertices 'vs' and 'vd', a label set A and bound δ, the bounded label constrained reachable paths are the A-paths lp_i, between 'vs' and 'vd' that satisfy the bounded path weight constraint $C(lp_i) \leq \delta$.

In this paper, we referred and termed the Bounded Label Constrained Reachable Paths as BLCRP.

For instance, for Fig. 1, we need to find the path for BLCRP query q(*vs, vd, A, δ*) with *vs*="P1", *vd*="P4", A={*S, N, T*} and δ =100. The resultant bounded A-path *lp* is {P1, P2, P3, P4} with the path cost of 95.

In the process of solving BLCRP, we derived the novel concept of implicit paths and defined it as follows:

DEFINITION 4. (Implicit Paths) Implicit paths are defined as the paths in the given graph that implicitly satisfy the given reachability constraints but need not satisfy the minimality of label sets.

For instance, in Fig. 1, there exist two paths from P1 to P5, i.e., lp_1= {P1, P5} with path label {N}, path weight 44, and lp_2= {P1, P2, P5} with path label {N, T}, and path weight 42. For the BLCRP query q(*P1, P5, {T, N}, 44*), the resultant bounded paths satisfying the given constraints can be both $lp_1$and lp_2. The path lp_2 although violating minimality of label sets property [8], satisfies the given reachability constraints. Hence, lp_2 is an implicit path.

Table -I: Survey of Constrained Reachability Techniques from [2010-2018

S.No.	Constraint	Technique (Authors,Year)	Advantages
1	edge label set member	Maximal Spanning Tree based query processing (Jin et al., 2010 [4])	Compress the transitive closure of the graph to more than two orders of magnitude
2	edge label set member	Bidirectional Labeled BFS (Fan et al., 2011 [18])	Used as base technique for SplitMatch and JoinMatch algorithms to find matching graph patterns in large graphs
3	every edge weight bound	Memory based and Disk-based Index and query processing algorithms (Miao Qiao et al., 2013 [19])	Highly scalable disk based algorithm and faster query processing

4	path distance and exact edge label match in uncertain graphs	Spanning Tree based BPFilter algorithm (Minghan Chen et al., 2014 [20])	Generate subpath query and find approximate reachability in uncertain graphs
5	edge label set member	Augmented Transitive closure technique and partition approach (Zou et al., 2014 [21])	Scalability is addressed using sample based solution for good partition
6	multidimensional vertex and edge attributes	Hashing and graph-attribute clustering based solution (Duncan Yung et al., 2016 [22])	finds reachability with multidimensional attribute constraints
7	edge label set member	Landmark Index and Query algorithms (Valstar et al., 2017 [8])	solves LCR with significant speed-ups in query processing
8	edge label set member and bound to path-weight	LWPathIndex and QBPath algorithms (Bhargavi B. et al., 2018 [1])	finds reachability within the given edge label constraints and given maximum path weight in weighted graphs

III. RELATED WORK

In this section, we reviewed various techniques that find the reachability, constrained reachability, shortest paths, and constrained paths. Also, we surveyed on finding path compression techniques and efficient ways to index labels and paths.

A. Reachability Techniques

Graph reachability finds the existence of path between the vertices in a graph. In [10], [7] detailed survey of reachability techniques was specified. H. Wei et al. [11] classified the reachability techniques into two categories; Label-only approaches and Label+G approaches. Label-only approaches directly find the existence of reachability between the two vertices from the constructed index. Label-only approaches include 2-hop [5], 3-hop [6], Chain-cover [12], Path-Tree Cover [13], Tree-Cover [14], and Dual Labeling [15]. Label+G approaches use index labels and BFS/DFS to find the reachability between two vertices. Tree+SSPI [16] and GRIPP [17] are Label+G approaches which have linear index size and query time in terms of the number of vertices of the graph. H. Wei et al. [11] also developed an independent permutation labeling approach with two additional labels to find the reachability between two vertices.

Different variants of reachability are formed by incorporating constraints to vertices and edges of the graph. The vertex/edge constraints can be a specific bound for each edge weight, specific vertex labels, specific edge labels, membership of specific edge labels, membership of specific vertex labels, and edge/vertex labels in a specified order. Table I describes the different constraints, their corresponding reachability techniques and its advantages.

Valstar et al. [8] technique of landmark based query processing is the latest technique to find the existence of reachability for LCR queries. The landmark vertices are selected based on criteria such as the top ‘k’ highest degree or any of the centrality measures of the graph. In [8], while constructing the landmark index for the landmark vertices, the edge label sets minimality is considered and landmark vertices are selected based on highest total degree criterion. The index is constructed for landmark vertices using BFS and forward propagation of indexed landmark vertices. For the remaining vertices, reachability of up to ‘b’ landmark vertices are indexed [8]. This index along with BFS is used to solve LCR queries.

B. Path Finding Techniques

Chris Barrett et al. [23] defined edge label and edge weight constraints in formal language and computed the constrained shortest paths through dynamic programming. The constrained TreeSketch algorithm of Ankita et al. [9], considered edge label constraint (A) and finds approximate A-paths between the given vertices. In [24], the landmark-based indexes are used to compute approximate shortest path distance. Minghan Chen et al. [20] worked on uncertain graphs with their new sampling techniques to find approximate shortest paths constrained through distance parameter.

In [25], [26], a B+ tree index based solution was used to solve reachability queries for graphs. In [27], a landmark-based approach with efficient pruning was developed to solve queries to find the exact shortest path distance between two vertices. In [28], a scalable solution based on 2-hop labels was used for solving the distance queries in large networks. In [29], the partition replication method, workload prediction method, and workload balancing method addressed the data locality and workload balancing while finding reachability in large attributed graphs.

From the literature review, we observe that there is a scope for finding efficient reachability techniques that have lesser index construction time, lesser index size, and faster query processing. The existing reachability techniques cannot be applied directly to constrained reachability queries as the attributes of vertices/edges are not stored while computing the index. Many approximate shortest path techniques do not compute exact paths. Hence, there is a need to find techniques that compute the exact reachable paths satisfying the given vertex/edge attribute constraints. Hashing is found to be an efficient data structure for secondary storage access than the B+tree [30]. Duncan Yung et al. [22] used BFS with hashing techniques to compute vertex-labeled reachability on big attributed graphs.

IV. NOVEL TECHNIQUE OF IMPLICIT PATH INDEXING AND QUERY PROCESSING

We extended the landmark based technique for LCR queries [8] and proposed an improved path index algorithm to solve the proposed problem of BLCRP formally defined in section II. We also use the proposed novel concept of implicit paths defined in section II. The extended work includes

- **Appending intermediate vertices and path weights:** By including the paths and total path weight in the path index,we can find the paths and check the bounds of paths faster during query processing.

- **Incorporating implicit paths in the path index:** By including the implicit paths in the proposed path index, we achieve higher recall than LWPathIndex algorithm (referred to as LM2) [1]. These extensions are referred to as LandMark path extensions (LM3) in section V.

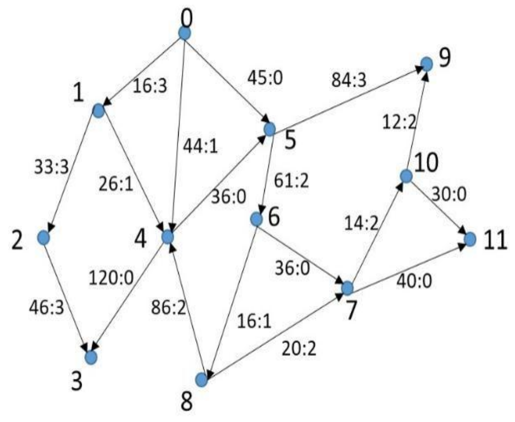

Fig. 3. Pre-processed directed graph G' of the road network G1

Table –II: Implicit Path Landmark Index for Fig. 3

Index	Tuples
IPL[4]	<5, 1, -, 36>, <6, 5, 5, 97>, <8, 7, 5-6, 113>, <3, 1, -, 120>, <7, 5, 5-6, 133>, <9, 9, 5, 120>
IPL[5]	<6, 4, -, 61>, <9, 8, -, 84>, <7, 5, 6,97>, <7, 3, 6-8, 97>, <4, 3, 6-8, 163>, <10, 5, 6-7, 111>, <11, 5, 6-7, 137>
IPL[7]	<10, 4, -, 14>, <9, 4, 10, 26>, <11, 1, -, 40>
IPL[0]	<1, 8, -, 16>, <4,10, 1, 42>, <4, 2, -, 44>, <5, 1, -, 45>, <2, 8, -, 49>, <3, 8, 1-2, 95>, <6, 5, 5, 106>, <8, 7, 5-6, 122>, <7, 5, 5-6, 142>, <3, 3, 4, 164>, <9, 9, 5, 129>

Table –III: Path Non-Landmark Index of for Fig. 3

Index	Tuples
PNL[1]	<4, 2, -, 26>, <5, 3, 4, 62>
PNL[6]	<7, 1, -, 36>, <4, 6, 8, 102>
PNL[8]	<4, 4, -, 86>, <5, 5, 4, 122>, <7,4, -, 20>
…	…

Algorithm 1: LWPathIndexImplicit

```
  Input   : G'
  Output  : transL, IPL, PNL
1 IPL[vi]←ImplPathperLV (vi)// i ∈ 1:k
2 PNL[vj ]←ImplPathIndNV (vj )// j ∈ remaining (n-k) vertices
3 procedure ImplPathperLV (v) // Enqueue (v,,,0)in priority queue q
4 while (!isempty(q)) do
5     Add (u, PL, iv, wt) IPL[s] through AddIPath ()
6     Append u to transL[s][S] if PL=S, else insert (u, PL) to transL[s]
7     if u is indexed then
8         ImplExpand (); continue
9     for (w, P Lt , wtt) ∈ outneighbors(u) do
10        iv←join(iv,w)
11        PLtt ← PL∪ PLt
12        wttt ← wt+wtt
13        if PathLength(s~w)≤ ⌈(0.5 * diameter)⌉ then
14            Add (w, PLtt, iv, wttt) to q

15 procedure AddIPath (s, v, PL, intv, C)
16 for (v, PLt , intvt , Ct ) PInd[s] do
      // Case 7 and case 8
17    if (PLt⊆ PL and Ct≤ C)) then
18        return false
      // Case 5 and case 6
19    if (PL⊆ PLt and Ct ≥C)) then
20        Delete entry (v, PLt, intvt, Ct) from PInd[s]
21        IE ← (v, PL, intv, C)
22 Insert IE into PInd[s].
23 return true
```

During index construction, we select ‘k’ vertices sorted in descending order of total degree as landmark vertices. The ‘k’ value is considered to be $\lceil sqrt(n) \rceil$ which is derived from theoretical observation of upper bound on the burning number of graph [31], [8].

For each landmark vertex(v), we construct a reachable Implicit Path Landmark index (IPL) that includes reachable vertex (from v), its label , path and weight. For non-landmark vertices, upto ‘b’ reachable landmark vertices are indexed in Path Non-Landmark (PNL). While inserting the same vertex
with different labels or weights, we consider the minimality of label sets [8] and Dijkstra’s relaxation property based on Table V.

Heuristic: We assume that paths of path length less than or equal to half the diameter of the graph exist between the vertex and most of its reachable vertices. This heuristic is considered while indexing to reduce the index construction size and the index construction time. For instance, in Fig. 3, with heuristic, (i) the number of intermediate paths stored for vertex 4 are six (as shown in table II). (ii) the path length of the stored paths is less than or equal to three (which is the upper bound of half of the diameter of the given graph). But, without heuristic, it includes all possible paths from the vertex 4 to its reachable vertices (including vertices 10, 11). Hence, we index the paths of length less than or equal to half the diameter of the graph for the nodes for faster indexing to handle large graphs.

Table II shows the path landmark index for landmark vertices 4, 5, 7 and 0 of pre-processed directed graph G' of Fig. 3. In pre-processing, vertices are numbered from 0 to n-1.

The labels are also numbered from 0 to $|T_l|$-1. The corresponding bits of numbered labels are set during path index construction.

1) *Handling Label and Path-weight constraints:* Table IV shows the different possible cases during path indexing for which minimality and Dijkstra’s relaxation properties are handled. We denote the path label to be indexed by PL and path label already indexed by PL'. We denote v as the reachable vertex and iv and iv' denote the intermediate paths of the new tuple (to be indexed) and indexed tuple respectively. We denote the path weight to be indexed by C and indexed path weight by C'.

For the cases 1 and 2 of Table IV, both minimality of label sets and Dijkstra’s relaxation property are not preserved; yet the resultant paths may implicitly satisfy the given label constraints and bounded path constraints. Hence, both the existing tuple <v, PL', iv', C'> and new tuple <v, PL, iv, C> are retained. For instance, in Table II, for IPL[0], both tuples <4, 10, 1, 42> and <4, 2, -, 44> are retained although minimality of label sets and Dijkstra’s relaxation property constraints are violated. Thus, implicit paths are included in the constructed index through case 1 and case 2 of table IV.

When $PL \neq PL'$, irrespective of $C > C'$ or $C \leq C'$, the tuple $<v, PL, iv, C>$ can be added to path index for cases 3 and 4 of Table IV. For case 5 and case 6, $PL \subseteq PL'$ and $C < C'$, both minimality of label sets and Dijkstra's relaxation property constraints are satisfied. Hence, the existing index tuple tuple $<v, PL', iv', C'>$ is removed and the new tuple $<v, PL, iv, C>$ is added.

Table -IV: Label and path-weight constraints while indexing with implicit paths

Case No.	Label set condn.	Path-weight condn.	(PL', C') removed?	(PL, C) add?	Dijkstra Property preserve?	Minimality preserve?
1	$PL' \subset PL$	$C < C'$	No	Yes	No	No
2	$PL \subset PL'$	$C > C'$	No	Yes	No	No
3	$PL \subset PL'$ and $PL' \not\subset PL$	$C < C'$	No	Yes	No	Yes
4	$PL \subset PL'$ and $PL' \not\subset PL$	$C > C'$	No	Yes	No	Yes
5	$PL \subset PL'$	$C \leq C'$	Yes	Yes	Yes	Yes
6	$PL = PL'$	$C < C'$	Yes	Yes	Yes	Yes
7	$PL = PL'$	$C \geq C'$	No	No	Yes	Yes
8	$PL' \subset PL$	$C \geq C'$	No	No	Yes	Yes

Algorithm 2: BImplPath

```
Input : vs, vd, A, maxcost
Output : Bunded Paths p[i], i∈[1, maxp] where maxp is maximum
number of paths allowed
1  if s∈ VL then
2      pi=QPathLM (vs, vd, A, maxcost)
3      return p
4  for (v, PLt , intvt , Ct ) ∈ PNL[vs] do
5      if ( PLt ⊆ and (v= vd)) then
           // when target vertex is a PLandmark vertex
6          if C(vs v)≤ maxcost then
7              Insert vs ~v to p
8              break

9      if ( PLt ⊆ A and QMark (v, t, A, marked, maxcost)=true) then
10         Insert vs~v~pi into p

11 Enqueue vs
12 while (!isempty(q)) do
13     Dequeue vertex v
14     if (v=vd) then
15         pt ← s~vd
16         if (pcost(pt ) ≤ maxcost) then
17             Add path pt to p
18             break

19     if v ∈ VL and QMark (v, vd, A, marked, maxcost)=true then
20         pt ← vs~v~pi
           // pi is intermediate path from QMark()
21
22         if (pcost(pt ) ≤ maxcost) then
23             Add path pt to p

24     for (w, PLt ) ∈ outneighbors(v) do
25         if (marked(w)=false & PLt ⊆ A)) then
26             Enqueue w

27 if (pi is not empty and pcost(pi)≤maxcost) then
28     return pi
29 procedure QPathLM (s, t, PL, bound)
30 for ((u, PLt , ivt , wtt ) in IPL[s]) do
31     if (u=t) & (PLt ⊆ PL) then
32         pi =s ~ivt ~ t
33         if (wtt ≤ bound) then
34             Add pi to p

35 procedure QMark (s, t, PL, marked, bound)
36 if (QPathLM (s, t, PL, bound)=true) then
37     return true
38 for (St , PLt ) ∈ transL[s] do
39     if (PLt ⊂ PL) then
40         marked= St ∪ marked

41 return false
```

For case 7 and case 8, $PL' \subseteq PL$ and $C > C'$, both minimality of label sets and Dijkstra's relaxation property constraints are satisfied. Hence, the existing index tuple $<v, PL', iv', C'>$ is retained and the new tuple $<v, PL, iv, C>$ is not added.

The proposed Algorithm 1 (LWPathIndexImplicit) describes landmark path index construction for the edge-labeled weighted directed graph. LWPathIndexImplicit algorithm first generates IPL index for landmark vertices by invoking call to ImplPathperLV() in Step 1 for each landmark vertex. Landmark vertices ($V_L \subseteq V$) are obtained by sorting vertices in decreasing order of their total degree and selecting the first 'k' vertices from the sorted vertices. Step 2 invokes ImplPathIndNV() procedure that generates PNL index.

2) Implicit Path Landmark Index (IPL): For each landmark vertex v_s, its descendent vertices are traversed as shown in steps 9-14 and pushed into a priority queue q. In priority queue, vertices are prioritized based on increasing path length from the source vertex (v_s). Each descendant vertex (v), its label (PL), intermediate path (iv) and total path weight (wt) are stored as tuple $<v, PL, iv, wt>$ in IPL[v_s] through AddIPath() of Step 5 in Algorithm 1. IPL is the path landmark index which stores the tuples in a list for each landmark vertex. AddIPath() adds the tuples by considering minimality of labelsets and Dijkstra's relaxation property constraints as shown in Table IV.

If v_s encounters an already indexed descendant vertex v', all the tuples of v' are joined to v_s using ImplExpand() as shown in step 8 of Algorithm 1. The reachable paths with path length less than or equal to half of the diameter of the graph are enqueued as shown in Step 13 of Algorithm 1. For instance, in Table II, for vertex 0 of graph G' (Fig. 3), IPL[0] stores all its reachable vertices with their path information.

*3)Path Non-Landmark Index(PNL):*ImplPathIndNV() stores upto 'b' reachable landmark vertices for each nonlandmark vertex in Algorithm 1. In ImplPathIndNV(), for each non-landmark vertex v_n, its descendent vertices are pushed into the priority queue. If dequeued vertex v' is a landmark vertex, the vertex (v'), its label (PL), intermediate path (iv) and total pathweight (wt) are stored as tuple $< v', PL, iv, wt>$ in PNL[v_n] by invoking AddIPath(). If v' is already indexed, its tuples are joined and stored in PNL[v_n] using ImplExpandNM(). In ImplExpandNM(), if vertex passed is non-landmark vertex, PNL[v'] is used, otherwise IPL[v'] is used to join the tuples. Thus, up to 'b' reachable landmark vertices are stored in PNL for each non-landmark vertex.

For instance, for non-landmark vertex 8 of graph G', Table III shows its reachable landmark vertices 4, 5, and 7 and their path information.

A. Query Processing

During query processing, the first step is to check if the source vertex is a landmark vertex or not. The proposed Algorithm 2 (BImplPath) describes the query processing of BLCRP query q(*vs*, *vd*, *A*, *maxcost*). If '*vs'* is a landmark vertex, QPathLM() is invoked as shown in step 2 of Algorithm 2.

QPathLM() uses IPL[*vs*] as shown in step 29 of Algorithm 2 to find the target vertex. If the target vertex is reached and label and bound constraints are satisfied, then the correspondingpaths are added to *p*. For instance, for BLCRP Query q(0, 7, 5, 180), the resultant path {0, 5, 6, 7} is returned using IPL[0] through QPathLM() of Algorithm 2.

If source vertex is non-landmark vertex and target vertex is landmark vertex, the target vertex can be reached via PNL as shown in step 4-8 of Algorithm 2. If target vertex is non-landmark vertex, source vertex may reach the target vertex via an intermediate landmark vertex by invoking QMark() as shown in step 9 of Algorithm 2. These cases of source vertex and target vertex are clearly described in the proof of theorem 1. For instance, BLCRP query q(8, 10, 4, 60) returns the resultant path {8, 7, 11, 10} via PNL[8] and IPL[7] using step 4-8 of Algorithm 2. From step 12, descendant vertices are traversed till target vertex is reached satisfying the label constraints. If an intermediate landmark vertex is reached, its reachable vertices are checked for the target vertex using QMark() as shown in step 19 of Algorithm 2.

B. Correctness Proof for LWPathIndexImplicit Algorithm and BImplPath Algorithm

The following lemmas and theorem 1 prove the correctness for the proposed LWPathIndexImplicit and BImplPath algorithms.

Lemma 1: Let G'(V, E, L, w) be the graph. Let $\{IPL[v_i]\}_{i\in[k]}$ *be index constructed by LWPathIndexImplicit(G') where* v_1, v_2, v_3,..v_k *are landmark vertices. Then, for every i* $\in$ *[1, k] and k* $\in$ *N, IPL[*v_i*] is sound but not complete.*

Proof: The paths indexed in IPL[v_i] are valid from v_i to any reachable vertex *v* as we traverse through the descendant vertices of v_i using steps 9-14 of Algorithm 1 (LWPathIndex-Implicit).We consider indexing only those paths whose path length is less than or equal to half of the diameter of the graph in Step 13 of Algorithm 1. Hence, IPL[v_i] is not complete. We include the labels by union of labels along the path as shown in step 11 of Algorithm 1, and we also include the intermediate vertices and path weight for each record in the path index. Thus, all possible reachable vertices are traversed and indexed with their path information. Hence, we can say that IPL[v_i] is sound.

Lemma 2: Let G'(V, E, L, w) be the graph. Let $\{PNL[v_i]\}_{i\in[k+1,\ n]}$ *be index constructed by LWPathIndexImplicit(G') where* v_{k+1}, v_{k+2}, v_{k+3},..v_n *are non-landmark vertices. Then, for every i* $\in$ *[k+1, n] and k* $\in$ *N, PNL[*v_i*] is sound..*

Proof: For each non-landmark vertex v_i, its descendant vertices are traversed by invoking ImplicitPathIndNV() from of algorithm 1. Only landmark vertices are added to PNL[v_i]. It is not complete because upto 'b' landmark reachable vertices are only indexed. Thus, every reachable landmark vertex and its path information is included in PNL[v_i] up to 'b' records. Hence, PNL[v_i] is sound.

Theorem 1. *Bounded path for BLCRP query, q= (vs, vd, A, δ) is a true query if BImplPath(vs, vd, A, δ) using the path index from LWPathIndexImplicit algorithm returns at-least one bounded path.*

Proof: We can prove the above theorem in the following cases:

Case 1: If *'vs'* is a landmark vertex, BImplPath*(vs, vd, A, δ)* invokes call to QPathLM() which finds the target vertex information from IPL[vs] in step 29 of Algorithm 2 and returns
the resultant path if it is within the bound (*δ*).

Case 2: If *'vs'* is not landmark vertex, its reachable landmark
vertices are traversed through PNL[*vs*] using step 4 of Algorithm 2. From each of the landmark vertex, v_l reachable vertices are checked for target vertex through QMark() via IPL[v_l] using step 9 of Algorithm 2. Thus, query returns the resultant bounded path when target vertex is reached.

Case 3: If *'vs'* is not landmark vertex, and the target vertex '*vd'* is not reached via PNL[*vs*], then breadth first traversal of 'vs' is carried out till landmark vertex or target vertex is reached from step 12 of Algorithm 2. If an intermediate landmark vertex (v_l) is reached, the existence of target vertex is checked in IPL[v_l]. If target vertex is reached, query returns the resultant bounded path. If no paths are found from all the above cases, then BImplPath algorithm for the query returns zero paths.

C. Time Complexity

To compute time complexity of LWPathIndexImplicit algorithm, it involves computing time complexity for the following significant steps:

- Step 1 of Algorithm 1 invokes call to ImplPathperLV() k times. It takes O(kx) time where x is time complexityof ImplPathperLV().
- Step 2 of Algorithm 1 takes O((n-k)y) time where y is the time complexity of ImplPathIndNV().
- Step 3 takes O(x) time. It takes atmost $2^{|T|}$ labels for each landmark vertex in the worst case where *T* is total number of labels. Each push into priority queue requires O(logn) time. Thus, O(x)=O(n*(logn+$2^{|T|}$)).
- Step 6 of Algorithm 1 requires atmost $2^{|T|}$ time to store all possible labels in transL.

Let us assume ImplicitPathIndNV() method has O(y) time complexity. Since, upto b landmark vertices are considered, O(y)=O((n*(logn+b)+m) $2^{|T|}$). Thus, the proposed path indexing requires O(k(((n(logn+$2^{|T|}$)m)*$2^{|T|}$)))+O((n-k)(n(logn+b)+m)*$2^{|T|}$)time. The query processing time complexity is computed through following steps:

- QPathLM() of BImplPath algorithm requires O($2^{|T|}$ +logn) as finding any specific w in IPL needs O(logn) time.
- QMark() requires to invoke QPathLM() and setting n bits of marked for atmost $2^{|T|}$ | labels. Thus, the worst case running time of QMark() is O(n+$2^{|T|}$).

- The remaining graph exploration part of Algorithm 2 takes at-most O(n+m) time.

Thus, the total query processing requires $O(m+k(2^{|T|}+n))$ time.

V. EXPERIMENTAL EVALUATION

We conducted experiments in C++ programming on Intel 2.6GHz 32-core server with CentOS Linux operating system and 64GB RAM on real and synthetic data sets shown in table V. The following subsection describes the datasets. While constructing landmark path index, we considered b=20 and $k = \lceil sqrt(n) \rceil$ based on the parameter values set in [8] for the proposed approaches. Table VI shows the resultant path index construction size and the index construction time for the datasets described in Table V respectively. From table VI, we observe that our proposed approach in this paper (LM3) has a faster index construction time and lesser index size than the previously proposed landmark based path index approach (LM2) [1], which is because we do not consider the minimality of label sets for all the cases.

A. Dataset description

We describe the significance and generation of the real and synthetic datasets of Table V. The edge weights are assigned values from {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120} randomly to all datasets.

1) Erdos-Renyi Graph: Erdos-Renyi (E-R) graphs are the basic random graphs [34] with near uniform degree distribution. E-R graph is generated using SNAP [33] with possible edges for each vertex set to 2 and n=1000. We assign 8 labels randomly to the edges.

2) Preferential-Attachment Graph: Preferential-attachment (P-A) graphs are synthetic graphs with the property that nodes with higher degree are more likely to have edges to be added in the future. These graphs have skewed degree distribution. P-A Graphs are scale-free networks [34] and hence mimic the real world networks. P-A graph is generated using SNAP [33] with n=1000 and m=2000. The 8 labels are randomly distributed to the edges.

3) Robots: Robots is a real trust network [32] with 4 edge labels; M-master, A-apprentice, Journeyer-J and Observer-O. The edge labels denote the level of trust interaction between the users.

Table- V: Dataset Repository

S.No.	Dataset	*n*	*m*	*|T|*	real/synthetic
1	Robots	1724	3596	4	real
2	Erdos-Renyi Graph	1000	2000	8	synthetic
3	Preferential-Attachment Graph	1000	1997	8	synthetic

Table- VI: Index construction time and IPL index size for LM3 and LM2 approaches

S.No.	Dataset	LM2		LM3	
		Time(s)	Size (KB)	Time (s)	Size (KB)
1	Robots	8575.48	1642	1373.12	1291
2	Erdos-Renyi Graph	527.98	713	137.26	572
3	Preferential-Attachment Graph	16579.8	1148	1154.27	1061

Table- VII: Run-time of LM3 approach

S.No.	Dataset	*|A|*	*Average Execution Time(s)*	τ
1	Robots	2	0.0003	0.0736
2	Erdos-Renyi Graph	4	0.0080	0.0404
3	Preferential-Attachment Graph	4	0.0005	0.0300

Table- VIII: Recall Analysis of LM3 approach

S.No.	Dataset	*|A|*	Min. recall	Max. recall	Average recall
1	Robots	2	0.1666	1	0.5939
2	Erdos-Renyi Graph	4	0.3	1	0.8097
3	Preferential-Attachment Graph	4	0.1666	1	0.5673

Table- IX: Statistical Analysis of LM3 vs LM2

S.No.	Dataset	*|A|*	t-statistic	p value
1	Robots	2	3.1555	0.001076
2	Erdos-Renyi Graph	4	4.328	0.00001
3	Preferential-Attachment Graph	4	7.228	0.000005

B. Query Generation and Evaluation

We generated 100 true queries and 100 false queries based on BFS query generation [8] with random path weights and tested with different labels for different datasets. During query generation, we randomly generated the bound for all queries with their values in range [10*diameter, 120*diameter]. We have considered this range for bound, as paths, in general, have path length near to the diameter of the graph.

For synthetic datasets, we generated queries with labels, |A|=4, 6 out of the total eight labels, and for Robots dataset, we generated queries with |A|=2 out of the 4 labels. Table VII shows the average execution time and the false negative ratios (τ) for different datasets. The false negative ratios are due to the inclusion of the condition path length $\leq$ (0.5*diameter) of the graph. It is found to be considerably very small.

We computed accuracy measures of Precision and Recall for the proposed landmark based path indexing algorithm. We computed precision based on (1) by checking if the resultant retrieved paths are present in the graph and if they satisfy the given label constraints and bounded weights.

$$Precision = \frac{Number\ of\ retrieved\ paths}{Number\ of\ retrieved\ and\ relevant\ paths} \quad (1)$$

Recall is computed based on (2) by considering minimality of label sets and Dijkstra's relaxation property for the LM3 approach to find whether all possible relevant paths are retrieved from the graph database. The total number of possible paths is computed based on DFS satisfying the given label constraints and bound constraints.

$$Recall = \frac{Number\ of\ retrieved\ paths}{Total\ number\ of\ relevant\ paths} \quad (2)$$

The precision percentage is computed on the real and synthetic datasets for the proposed LM3 and found to be 100% for all the datasets.

The recall percentage is computed for verification and is found to be between 16% and 100% for LM3 tested on real and synthetic benchmark datasets for the true queries. Table VIII shows the minimum, maximum, and average recall computed for 100 true queries on different datasets which reveals that landmark based path indexing solves BLCRP queries efficiently. We performed the statistical analysis of recall for LM3. Paired t-test is used to find the statistical significance of LM3 over LM2 [1]. The sample data values need to be in the normal distribution to use the paired t-test. Also, the sample size must be at least 30 for non-normal data[35]. We have tested recall values for 100 random true queries with |A|=4 for synthetic datasets and |A|=2 for real dataset to find the statistical significance of our proposed approach (LM3) compared to previous approach (LM2).

The following are the parameters set to compute paired t-test [35] :

- The null hypothesis is set to difference of recall values to 0.
- Number of samples, n=100.
- The t-statistic is computed and the probabilistic value p is calculated using statistical function tdist() as p=tdist(t, df, 1) where df is the degrees of freedom (df=n-1).

Table IX shows that our proposed approach is statistically significant than previous approach with respect to recall for real and synthetic datasets. From Table IX, we observe that LM3 is statistically significant than LM2.

VI. CONCLUSION AND FUTURE SCOPE

We proposed an efficient solution to the problem of finding BLCRP by extending the landmark based indexing and query processing. We obtained the reachable paths that satisfy given label constraints and bounded by weight. We adopted landmark based indexing and extended to store path information in path index. The bounded path based LCR query is processed by using IPL and PNL indices to get the resultant paths. If the target vertex is not still obtained, then BFS with IPL index is used in query processing to find the existence of paths and the resultant reachable paths are retrieved. We evaluated accuracy of our path landmark technique by using precision and recall measures on the resultant paths. We observed that our approach is statistically significant than our previous proposed approach with respect to recall tested on synthetic and real datasets.

A. Future Scope

In the future, we can optimize by storing the path results in efficient logs and use the logs to solve the repeated queries. We can further optimize the proposed approach using path compression techniques to compress the intermediate paths. Besides, we can find efficient techniques to handle multidimensional vertex and edge attributes for graph reachability queries. As described in section II, we can further explore and study the case 3 and case 4 type of reachability queries. We can extend our research by applying our proposed techniques to find paths for all LCR queries in an edge-labeled directed graph.

REFERENCES

1. B. Bhargavi and K. Swarupa Rani. Bounded paths for LCR queries in labeled weighted directed graphs. , in Advances in Computing and Data Sciences, pages 124–133, Singapore, 2018. Springer Singapore.
2. Diane J. Cook and Lawrence B. Holder. Mining Graph Data. John Wiley &#38; Sons, Inc., USA, 2006.
3. STRING Database. https://string-db.org/, 2019. [Online; accessed 27-June-2019].
4. Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang, Computing label-constraint reachability in graph databases. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD ’10, pages 123–134, New York, NY, USA, 2010. ACM.
5. Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. In Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’02, pages 937–946, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.
6. Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 3-hop: A high compression indexing scheme for reachability query. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD ’09, pages 813–826, New York, NY, USA, 2009. ACM.
7. Charu C. Aggarwal and Haixun Wang. Graph Data Management and Mining: A Survey of Algorithms and Applications. Springer US, 2010.
8. Lucien D. J. Valstar, George H. L. Fletcher, and Yuichi Yoshida. Landmark indexing for evaluation of label-constrained reachability queries. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, pages 345–358, 2017.
9. Ankita Likhyani and Srikanta Bedathur. Label constrained shortest path estimation. In Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management, CIKM ’13, pages 1177–1180,New York, NY, USA, 2013. ACM.
10. Cheng J. Yu J.X. Graph Reachability Queries: A Survey, volume 40.Springer, Boston, MA, 2010.
11. Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. Reachability querying: An independent permutation labeling approach. The VLDB Journal, 27(1):1–26, February 2018.
12. Yangjun Chen and Yibin Chen. An efficient algorithm for answering graph reachability queries. In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, ICDE ’08, pages 893–902, Washington, DC, USA, 2008. IEEE Computer Society.
13. Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. Efficiently answering reachability queries on very large directed graphs. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD ’08, pages 595–608, New York, NY, USA, 2008. ACM.
14. R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, SIGMOD ’89, pages 253–262, New York, NY, USA, 1989. ACM.
15. Haixun Wang, Hao He2, Jun Yang, Philip S. Yu, Jeffrey Xu Yu, and Jeffrey Xu Yu. Dual labeling: Answering graph reachability queries in constant time. In Proceedings of the 22Nd International Conference on Data Engineering, ICDE ’06, pages 75–, Washington, DC, USA, 2006. IEEE Computer Society.
16. Li Chen, Amarnath Gupta, and M. Erdem Kurul. Stack-based algorithms for pattern matching on dags. In Proceedings of the 31st International Conference on Very Large Data Bases, VLDB ’05, pages 493–504. VLDB Endowment, 2005.
17. Silke Triand Ulf Leser. Fast and practical indexing and querying of very large graphs. In Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD ’07, pages 845–856, New York, NY, USA, 2007. ACM.
18. Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. Adding regular expressions to graph reachability and pattern queries. IEEE, pages 39–50, 2011.
19. Miao Qiao, Hong Cheng, Lu Qin, Jeffrey Xu Yu, Philip S. Yu, and Lijun Chang. Computing weight constraint reachability in large networks.VLDB J., 22(3):275–294, 2013.
20. Minghan Chen, Yu Gu, Yubin Bao, and Ge Yu. Label and distance constraint reachability queries in uncertain graphs. In Database Systems for Advanced Applications, pages 188–202, Cham, 2014. Springer International Publishing.

21. Lei Zou, Kun Xu, Jeffrey Xu Yu, Lei Chen, Yanghua Xiao, and Dongyan Zhao. Efficient processing of label-constraint reachability queries in large graphs. Inf. Syst., 40:47–66, March 2014.
22. Duncan Yung and Shi-Kuo Chang. Fast reachability query computation on big attributed graphs. In Big Data (Big Data), 2016 IEEE International Conference on, pages 3370–3380. IEEE, 2016.
23. Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-languageconstrained path problems. SIAM J. Comput., 30(3):809–837, May 2000.
24. Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Antti Ukkonen.Distance oracles in edge-labeled graphs. In Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014., pages 547–558, 2014.
25. Jonathan Sumrall. Path indexing for efficient path query processing in graph databases, 2015.
26. Jonathan M Sumrall, George HL Fletcher, Alexandra Poulovassilis, Johan Svensson, Magnus Vejlstrup, Chris Vest, and Jim Webber. Investigations on path indexing for graph databases. In European Conference on Parallel Processing, pages 532–544. Springer, 2016.
27. Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pages 349–360. ACM, 2013.
28. Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Robust distance queries on massive networks. In European Symposium on Algorithms, pages 321–333. Springer, 2014.
29. Li-Yung Ho, Jan-Jan Wu, and Pangfeng Liu. Workload prediction and balance for distributed reachability processing for large-scale attribute graphs. Concurrency and Computation: Practice and Experience, 30(6):e4344, 2018.
30. Michael J. Folk, Greg Riccardi, and Bill Zoellick. File Structures: An Object-Oriented Approach with C++. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997.
31. Max R. Land and Linyuan Lu. An upper bound on the burning number of graphs. In WAW, volume 10088 of Lecture Notes in Computer Science, pages 1–8, 2016.
32. Robots. http://tinyurl.com/gnexfoy/, 2017.
33. Jure Lescovec. SNAP: A general purpose network analysis and graph mining library in C++. . http://snap.stanford.edu/snap.
34. Réka Albert and Albert lászló Barabási. Statistical mechanics of complex networks. Rev. Mod. Phys, pages 47–94, 2002.
35. Henry H Su and P Eter A. L Achenbruch. Paired t-test. In Wiley Encyclopedia of Clinical Trials. John Wiley & Sons Inc., 2008.

AUTHORS FROFILE

Bhargavi B, is a Research Scholar from School of Computer and Information Sciences, University of Hyderabad. Her research areas include Graph theory, Social Networks and Graph Mining. She is student IEEE member and student ACM member.

Dr K. Swarupa Rani, is an Associate Professor at School of Computer and Information Sciences, University of Hyderabad. She is working in the area of Artificial Intelligence, Machine Learning, and Data Mining. She is part of Smart India Hackathon event group at University of Hyderabad.