An Overview of Reachability Indexes on Graphs

Chao Zhang, Angela Bonifati, M. Tamer Özsu

An Overview of Reachability Indexes on Graphs

Chao Zhang
University of Waterloo
Waterloo, Canada
chao.zhang@uwaterloo.ca

Angela Bonifati
Lyon 1 University
Lyon, France
angela.bonifati@univ-lyon1.fr

M. Tamer Özsu
University of Waterloo
Waterloo, Canada
tamer.ozsu@uwaterloo.ca

ABSTRACT

Graphs have been the natural choice for modeling entities and the relationships among them. One of the most fundamental graph processing operators is a reachability query, which checks whether a path exists from the source to the target vertex in a plain graph, and additionally whether the path can satisfy a given path constraint based on the edge labels in an edge-labeled graph. Processing reachability queries requires potentially visiting a large portion of the graph due to the inherent transitivity of these queries. This makes it costly to evaluate them on large graphs. Thus, significant effort has been spent to design indexing techniques for reachability queries in the last three decades, building advanced data structures to efficiently compress the transitive closure of the graph so as to accelerate online query processing, aka reachability indexes. In this tutorial, we provide an in-depth technical review of the existing reachability indexes, ranging from those designed for plain graphs to ones for edge-labeled graphs. We conclude the tutorial by summarizing the open challenges for integrating these techniques into GDBMSs.

CCS CONCEPTS

• **Information systems** → **Graph-based database models**; **Data structures**; *Database query processing*.

KEYWORDS

graph database; reachability query; reachability index

ACM Reference Format:
Chao Zhang, Angela Bonifati, and M. Tamer Özsu. 2023. An Overview of Reachability Indexes on Graphs. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3555041.3589408

1 INTRODUCTION

Graphs are ubiquitous for modeling real-world data [32] where it is important to represent relationships as first-class objects – vertices represent entities and edges represent relationships. Examples can be found in various domains, *e.g.*, biological networks [28], financial networks [30], social networks, transportation networks [4], and knowledge graphs [16]. With graph-structured data, one of the most interesting queries is to check the existence of a directed path from one vertex to another, *i.e.*, the transitive relationship from one entity to another in the network, which is known as the *reachability query* (Q_r). Reachability queries are well-known fundamental graph data processing operators and have been widely used in practice [36, 37]. They are considered by some as the most interesting graph-oriented queries [38].

Tutorial outline. In this tutorial, we present an in-depth review of each type of reachability indexes. We start the tutorial with the presentation of the background on reachability queries on graphs (§2), followed by the review of plain reachability indexes (§3) and the review of path-constrained reachability indexes (§4). We end the tutorial by discussing the open challenges (§5). We share our vision towards having full-fledged indexes in modern GDBMSs for reachability queries.

2 BACKGROUND

We adopt the common definition of a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges, and $e \in E \subseteq V \times V$. In this paper, the edges are directed. This definition will be enhanced as the graph is enriched. Therefore, the issue is checking the existence of paths in G according to the arguments in Q_r.

2.1 Plain reachability

Plain graphs follow the above definition $G = (V, E)$. Figure 1(a) shows a plain graph. The reachability queries over plain graphs are referred to as *plain reachability queries*. A plain reachability query $Q_r(s, t)$ contains a pair of arguments, a source vertex $s \in V$ and a target vertex $t \in V$, and $Q_r(s, t)$ checks whether there exists a path from s to t in G, referred to as an s-t path, *e.g.*, in Figure 1(a), $Q_r(A, G) = true$ because of an s-t path (A, D, H, G).

2.2 Path-constrained reachability

Edge-labeled graphs can contain different kinds of relationships, where each edge is assigned a specific label, *e.g.*, RDF graphs [15]. In this case, the graph is defined as $G = (V, E, L)$, where L is a set of labels and each $e \in E$ is assigned a label $l \in L$. Figure 1(b) shows an edge-labeled graph, representing a social network with three kinds of relationships, *i.e.*, $L = \{$`friendOf`, `follows`, `worksFor`$\}$. Reachability queries over edge-labeled graphs can be expressed with additional constraints on paths using edge labels [21, 52], referred to as *path-constrained reachability queries*. A path-constrained reachability query $Q_r(s, t, \alpha)$ contains as arguments source (s) and target (t) vertices, and an expression α specifying the path constraint using L. $Q_r(s, t, \alpha)$ checks whether there exists an s-t path in G such that the path can satisfy the constraint enforced by α. In general, α is defined as a regular expression using edge labels $l \in L$ as literal characters and using concatenation '$\cdot$', alternation '$\cup$', and the Kleene operators (star '$*$' or plus '+') as

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06…$15.00
https://doi.org/10.1145/3555041.3589408

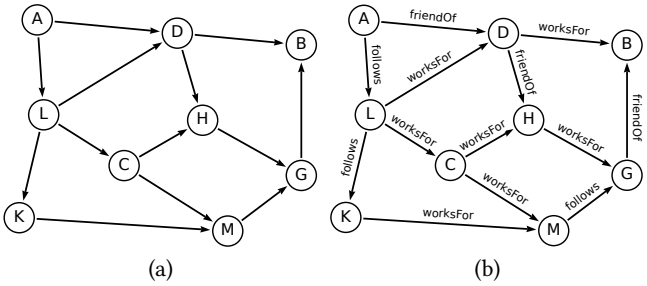

Figure 1: A plain graph (a) and an edge-labeled graph (b).

meta-characters, which is generated by the following grammar: $\alpha ::= l|\alpha \cdot \alpha|\alpha \cup \alpha|\alpha^+|\alpha^*$. The path constraint enforced by a regular expression α in $Q_r(s,t,\alpha)$ is that the sequence of the edge labels of an s-t path should form a word in the language of α. For instance, if $\alpha = (\texttt{friendOf} \cup \texttt{follows})^*$, then $Q_r(A,G,\alpha) = false$ in Figure 1(b) because every path from A to G includes `worksFor`. Path queries with regular expressions as path constraints are known as *regular path queries* (RPQ) [1, 3], and path-constrained reachability queries are a subclass of RPQs, which check the existence of an s-t path that can satisfy the path constraint defined by a regular expression. Path-constrained reachability queries have been widely used in graph analytics [12, 21, 33, 44, 52, 56], *e.g.*, social relationships analysis in social networks, analyzing interaction pathways of proteins in biological networks, money laundering detection in financial transaction networks, among others.

2.3 Reachability processing

Plain reachability queries can be processed by performing online traversal [50], *e.g.*, breadth-first traversal (BFS), depth-first traversal (DFS), or bidirectional breadth-first traversal (BiBFS). However, these approaches visit a large portion of the graph, which is prohibitively inefficient for large graphs. Path-constrained reachability queries can be processed by guided graph traversal: a finite automata (FA) can be built according to the regular expression α in the query, and then the traversal is guided by the FA [3, 5]. However, a significant portion of the graph still needs to be visited because of the presence of the Kleene operator in α.

The naive approach to accelerate plain reachability queries is to build a *transitive closure* (TC) [2], which can be extended to path-constrained reachability queries, called *generalized transitive closure* (GTC) [21, 52]. TC computes and stores the existence of a path between every pair of vertices in the graph. Although query processing with TC requires only constant time, the high computation and storage costs make it infeasible in practice. GTC extends TC by adding additional information of edge labels to deal with path-constrained reachability queries. However, the computation of GTC is more challenging than the computation of TC because of the additional distinction of paths according to a large number of possible path constraints in graphs. Consequently, computing GTC is also infeasible in practice.

Over the past three decades, significant research has focused on designing advanced data structures to efficiently compute and store

Table 1: A review of plain reachability indexes.

Indexing Technique	Framework	Index Type	Input	Dynamic
Tree cover [2]	Tree cover	Complete	DAG	No
Tree+SSPI [9]	Tree cover	Partial	DAG	No
Dual labeling [17]	Tree cover	Complete	DAG	No
GRIPP [43]	Tree cover	Partial	General	No
Path-tree [24, 27]	Tree cover	Complete	DAG	Yes
GRAIL [50]	Tree cover	Partial	DAG	No
Ferrari [40]	Tree cover	Partial	DAG	No
DAGGER [51]	Tree cover	Partial	DAG	Yes
2-Hop [14]	2-Hop	Complete	General	No
Ralf et al. [39]	2-Hop	Complete	General	Yes
3-Hop [26]	2-Hop	Complete	DAG	No
U2-hop [7]	2-Hop	Complete	DAG	Yes
Path-hop [8]	2-Hop	Complete	DAG	No
TFL [13]	2-Hop	Complete	DAG	No
DL [25]	2-Hop	Complete	General	No
PLL [49]	2-Hop	Complete	General	No
TOL [55]	2-Hop	Complete	DAG	Yes
DBL [29]	2-Hop	Partial	General	Yes
O'Reach [18]	2-Hop	Partial	DAG	No
IP [46, 47]	Approximate TC	Partial	DAG	Yes
BFL [41]	Approximate TC	Partial	DAG	No
HL [25]	-	Complete	DAG	No
Feline [45]	-	Partial	DAG	No
Preach [31]	-	Partial	DAG	No

TCs or GTCs while providing efficient processing for (plain/path-constrained) reachability queries. These advanced data structures are known as *reachability indexes* [2, 7–11, 13, 14, 17–20, 25–27, 29, 31, 34, 35, 39–41, 43, 45, 46, 49–51, 55]. The general intuition in designing reachability indexes is to reduce the redundant recording of reachability information in building TCs or GTCs and to store succinct information among different reachable pairs of vertices. This makes it possible to derive the reachability for any pair of vertices in the graph. The derivation process should take much less time than the query processing using online graph traversal, which makes indexes interesting. There exist two different types of reachability indexes that mirror the reachability queries: *plain reachability indexes* [2, 7–9, 11, 13, 14, 17–20, 25–27, 29, 31, 34, 35, 39–41, 43, 45, 46, 49–51, 55] and *path-constrained reachability indexes* [10, 12, 21, 33, 44, 52, 56].

3 PLAIN REACHABILITY INDEXES

The existing plain reachability indexes can be grouped into three main classes according to the underlying frameworks with a few additional techniques: tree-cover [2], 2-hop labeling [14], and approximate TC [46]. The plain reachability indexes are summarized in Table 1. The Index Type column denotes whether an index is partial or complete. *Complete indexes* can process queries by using only index lookups, while *partial indexes* require additional graph traversal. The Input column indicates the type of graph that the index assumes as input: DAG for directed acyclic and General for general graphs. The Dynamic column indicates whether the index can support graph updates, *i.e.*, edge/vertex deletion and insertion.

3.1 Tree-Cover-Based Indexes

The foundation of the tree-cover index [2] is one or more intervals attached to each vertex, which are leveraged to process reachability

queries. Let $[a_v, b_v]$ denote the interval attached to vertex v. Tree-cover approach is defined on spanning trees of a general graph and then extended to deal with the edges that are not covered by the spanning trees. Thus, we can identify three basic structures: (1) trees; (2) DAGs; and (3) general graphs. The indexing techniques [9, 17, 24, 27, 40, 43, 50, 51] based on the tree-cover framework are listed in the the first block of Table 1. The intervals attached to vertices differ in each of these.

Interval labeling for trees. In this class of techniques, for each vertex v, b_v is v's post-order number obtained by the post-order traversal from the root of the tree, and a_v is the lowest post-order number of all the descendants of v in the tree. $Q_r(s,t)$ can be processed by checking if $b_t \in [a_s, b_s]$. The intuition is that $Q_r(s,t)$ can be processed by checking whether the sub-tree rooted at s in the spanning tree of G contains t, and the intervals assigned to vertices encode sufficient information for such a checking.

Reachability in DAGs. A major problem arises when using the interval labeling approach to deal with reachability in a DAG: if vertex t is reachable from vertex s *only* via paths containing non-tree edges, interval labeling cannot capture such paths. In the original tree-cover approach, the problem is solved by first computing vertex intervals based on spanning trees of the DAG and then inheriting vertex intervals. For a non-tree edge (u,v) with intervals $[a_u, b_u]$ and $[a_v, b_v]$ and $b_v \notin [a_u, b_u]$, in order to record that v is reachable from u, u inherits $[a_v, b_v]$ from v. Consequently, u has two intervals $[a_u, b_u]$ and $[a_v, b_v]$. Interval inheritance is also required for tree edges in the DAG due to the transitivity of reachability, *e.g.*, for a tree edge (w,u), v is reachable from w, such that w needs to inherit $[a_v, b_v]$ from u. Vertices in DAGs are examined in the reverse topological order to inherit intervals. In case intervals for a vertex happen to be adjacent, they can be merged for efficient storage, *e.g.*, intervals $[1,6]$ and $[7,8]$ can be merged to interval $[1,8]$.

From cyclic graphs to DAGs. General graphs with directed cycles can be transformed to a DAG using an efficient reduction method (Tarjan's algorithm [42]) in linear time. Specifically, all the strongly connected components (SCC), where each vertex is reachable from every other vertex, are identified, and each SCC is coarsened into a representative vertex. The output of the transformation will be a DAG. Then, $Q_r(s,t)$ can be processed by first checking whether s and t belong to the same SCC, followed by checking the reachability in the DAG in case they are in different SCCs using the DAG approach discussed above. Thus, most plain reachability indexes in literature assume DAGs as input since generalization is easy.

In a nutshell, the tree-cover index is the interval labeling approach with interval inheritance. The main drawback of the tree-cover approach is the potentially large number of intervals as each vertex may need to inherit intervals form other vertices due to the existence of non-tree edges in the graph. The index size is defined as the total number of intervals allocated to vertices, and a DAG can be covered by spanning trees of different shapes, which result in different index sizes. The optimal tree cover [2] can lead to the minimum index size. However, the complexity of computing the minimum size index based on the optimal tree cover is the same as that of computing TC [2], which is unfeasibly high in practice.

Several follow-up works [9, 40, 50] aim at reducing the number of intervals for each vertex. These works adopt two types of designs for reducing the number of intervals: (1) recording exactly k intervals (*e.g.*, GRAIL [50]) and (2) recording at most k intervals (*e.g.*, Ferrari [40]), where k is an input parameter. In GRAIL, the k intervals are computed by using k random spanning trees. In Ferrari, intervals are merged even if they are not adjacent so as to guarantee that the maximum number of intervals for each vertex is not larger than k, which leads to the existence of approximate intervals. Neither approach computes a complete index, and the query results using index lookups may contain false positives but no false negatives. Thus, these partial indexes can be used to guide online traversal to compute correct query results. Although partial indexes require additional graph traversal for query processing, their index building time and index size scale linearly with the input graph size, making them one of the first methods feasible for large graphs with millions of vertices. In addition, reachability processing using these indexes can be an order of magnitude faster than using only graph traversal. GRAIL is extended for dynamic graphs in DAGGER [51].

Several early works extend interval labeling, including dual-labeling [17], GRIPP [43], and path-tree [24, 27]. However, their sophisticated designs might not be able to deal with current real-world graphs. Specifically, dual-labeling and path-tree are designed for tree structures *e.g.*, XML databases, and their application to graphs works well only if the number of non-tree edges is very low. Although GRIPP is a partial index, it requires graph traversal if the partial index returns *false*, which is not competitive compared to the design of GRAIL and Ferrari that do not have false negatives in the query results returned by their partial indexes.

3.2 2-Hop-Based Indexes

The 2-hop labeling lies in between transitive closure materialization and online search (BFS, DFS, BiBFS). In the 2-hop index [14], each vertex $v \in V$ is labeled with two sets of vertices: $L_{in}(v)$ and $L_{out}(v)$. A vertex $u \in L_{in}(v)$ if v is reachable from u and similarly for $L_{out}(v)$. Construction of these sets is discussed below. $Q_r(s,t)$ is processed by checking whether one of the following cases can be satisfied: *(1)* $s \in L_{in}(t)$; *(2)* $t \in L_{out}(s)$; *(3)* $L_{in}(t) \cap L_{out}(s) \neq \emptyset$, *i.e.*, we can compute whether t is reachable from s by determining whether there exists at least a common hop u. The indexing techniques based on the 2-hop index [7, 8, 13, 18, 25, 26, 29, 39, 49, 55] are listed in the second block of Table 1. Notice that unlike the tree-cover index, the 2-hop index can be directly applied to general graphs.

The index size of the 2-hop index is $\sum_{i=1}^{n} |L_{out}(v_i)| + |L_{in}(v_i)|$ for a graph with n vertices. The minimum 2-hop index is the one with the minimum index size. Computing the minimum 2-hop index is NP-hard [14]. An approximation algorithm was proposed in the original work of the 2-hop index. However, the time complexity is $O(n^4)$, which is infeasible for large graphs.

Efficiently computing the 2-hop index while reducing the index size maximally has been a long-standing problem. Advanced heuristics have been proposed, including TFL [13], DL [25], PLL [49], and TOL [55]. TOL is a general approach for computing the 2-hop index with a total order of vertices as input, and TFL, DL, and PLL are instantiations of TOL, *i.e.*, an instantiation of a total order is used in each of them. Let o be a strict total order on V and $o(v)$ be the rank of v in o. TOL applies backward and forward BFSs from each vertex $v \in V$, and for each vertex u visited by the BFSs, TOL adds v to $L_{out}(u)$ and $L_{in}(u)$ respectively only if $u \in V$ satisfies $o(u) > o(v)$.

If a vertex w such that $o(w) < o(v)$ is visited by the backward and forward BFSs from v, then the BFSs immediately terminate, *i.e.*, the search space for computing the reachability paths to and from each v is pruned according to the total order o. An instantiation of o is the topological order of a DAG (obtained after coalescing all the SCCs), which is the strategy used in TFL. Another instantiation of o is based on vertex degree, which is adopted in DL and PLL. It has been proven that DL and PLL are equivalent [25].

Several works based on the 2-hop index also study how to accommodate dynamic graphs. TOL can handle both insertions and deletions by leveraging the total order of vertices. DBL [29] is a recent proposal based on the 2-hop index designed for insertion-only. The U2-hop [7] approach and the approach by Ralf et al. [39] study the maintenance of the 2-hop index on dynamic graphs, but it has been shown that they cannot scale to large graphs [55].

There are a few extensions of 2-hop indexing. Early works replace the intermediate vertices in the reachability path with graph structures, *i.e.*, chains in the 3-hop index [26] and trees in the path-hop index [8]. Although these approaches can further reduce the index size, they require more time to build the index compared to recent 2-hop indexes. O'Reach [18], one of the state-of-the-art indexes, builds a partial 2-hop index by selecting k vertices and processes queries with an extended topological order.

3.3 Approximate Transitive Closure

Let $Out(v)$ be the set of all the vertices that are reachable from v in the graph (note that this is different from $L_{out}(v)$ defined earlier). Then, the computation of the TC for a graph is basically computing $Out(v)$ for every vertex v in the graph, which is unrealistic in practice. The intuition of an approximate TC is computing an approximate version of $Out(v)$ efficiently. We use $AP()$ to denote the approximating function. Note that $|AP(Out(v))| \ll |Out(v)|$. The $AP()$ function needs to be carefully designed so that it provides correct query results instead of random ones. The design of $AP()$ can be guided by the following observation. If it is known that vertex t is reachable from vertex s in the graph, then $Out(t) \subseteq Out(s)$. If $In(v)$ is defined as the dual, namely it contains all the vertices that reach v, a similar observation can be obtained, *i.e.*, if t is reachable from s, then $In(s) \subseteq In(t)$. Although the observation does not make sense since it contradicts processing $Q_r(s,t)$, it turns out that the contra-positive condition can be leveraged: if $Out(t)$ is not a subset of $Out(s)$, then t is not reachable from s in the graph. Therefore, as long as the $AP()$ function preserves the contra-positive conditions – if $AP(Out(t))$ is not a subset of $AP(Out(s))$, then $Out(t)$ is not a subset of $Out(s)$ – the approximate TC built using the $AP()$ function will not provide false negatives.

The first design of the $AP()$ function uses a k-min-wise independent permutation, leading to the IP index [46, 47], followed by using a Bloom filter in BFL [41], which is one of the state-of-the-art techniques for plain reachability indexing. Query processing using IP or BFL does not have false negatives since the corresponding $AP()$ function can preserve the contra-positive condition. However, false positives might exist, which calls for additional graph traversal. Similar to the case of the partial indexing techniques based on the tree cover framework, the graph traversal can be pruned by recursively querying the index, *i.e.*, if all the neighbors of v (the vertex currently visited by the traversal from the source vertex) do not reach the target vertex, then v can be skipped in the traversal.

3.4 Other Techniques

A few graph reduction techniques are proposed to accelerate reachability indexing, including SCARAB [23], ER [54], and RCN [53]. These reduction techniques are orthogonal to the indexing techniques. Several other techniques that do not use the three indexing frameworks discussed earlier also exist in literature, including HL [25], Feline [45], and Preach [31]. In general, these indexes have their own designs to deal with reachability queries. A few early works study the maintenance of transitive closure or SCCs on dynamic graphs, including [19, 34, 35], but these approaches cannot scale to large graphs [55].

4 PATH-CONSTRAINED REACHABILITY INDEXES

The reachability indexes that support path-constrained reachability queries $Q_r(s,t,\alpha)$ are generally designed for a specific type of path-constrained expressions α. Based on the specification of α there are two types of path-constrained reachability queries: *alternation-based* queries, where $\alpha = (l_1 \cup l_2 \cup ...)^*, \forall l_i \in L$ [12, 21, 33, 44, 56], and *concatenation-based* queries, where $\alpha = (l_1 \cdot l_2 \cdot ...)^*, \forall l_i \in L$[52].

Unfortunately, there is currently no index that can support both query classes; indexes have been proposed for either one or the other and these are shown in Table 2. As indicated in the table, there are three index classes that support alternation-based queries: (1) tree-based, (2) 2-hop, and (3) GTC. The first two are extensions of their counterparts for the plain graphs, while GTC is an extended TC enriched by edge label information specific for this query type. There is only one index that has been proposed for concatenation. In the remainder, we discuss each of these index classes.

Table 2: A review of path-constrained reachability indexes.

Indexing Technique	Framework	Path Constraint	Index type	Input	Dynamic
Jin et al. [21]	Tree cover	Alternation	Complete	General	No
Chen et al. [12]	Tree cover	Alternation	Complete	General	No
Zou et al. [48, 56]	GTC	Alternation	Complete	General	Yes
Landmark index [44]	GTC	Alternation	Partial	General	No
P2H+ [33]	2-Hop	Alternation	Complete	General	No
DLCR [10]	2-Hop	Alternation	Complete	General	Yes
RLC index [52]	2-Hop	Concatenation	Complete	General	No

4.1 Indexes for Alternation-Based Queries

An alternation of edge labels defines a set of labels $L' \subseteq L$, and the path constraint in $Q_r(s,t,\alpha)$ enforces that an s-t path should only contain edges with labels that are in L', *e.g.*, if $\alpha = (\texttt{friendOf} \cup \texttt{follows})^*$, then $L' = \{\texttt{friendOf}, \texttt{follows}\}$, and $Q_r(A,G,\alpha) = false$ in Figure 1(b). An alternation-based reachability query is also known as *label-constrained reachability* (LCR) query.

Sufficient path-label sets. The problem of indexing alternation-based reachability queries is first studied by Jin et al. [21], where two foundations on edge label information for indexing such queries are proposed. The first foundation is that if there are two s-t paths with edge-label sets S_1 and S_2 and $S_1 \subseteq S_2$, then S_2 is redundant, which means recording S_1 in the index is sufficient. Edge-label sets such

as S_1 are denoted as *sufficient path-label sets* (SPLS). For instance, in Figure 1(b), vertex M is reachable from vertex L via two paths: p_1 = (L, worksFor, C, worksFor, M), and p_2 = (L, follows, K, worksFor, M), and the label set of p_1 is a subset of the label set of p_2, such that the former is the SPLS from L to M. The second foundation is that the SPLSs of paths from vertex s to vertex t can be obtained by computing the cross product of the ones from vertex s to vertex u and the ones from vertex u to vertex t (transitivity of SPLSs). For instance, in Figure 1(b), the SPLS from A to M is {follows, worksFor}, which can be computed by using the SPLS from A to L, *i.e.*, {follows}, and the SPLS from L to M, *i.e.*, {worksFor}.

4.1.1 ***Tree-Based Indexes.*** The first indexing approach is proposed by Jin et al. [21], which consists of a tree-based index enriched with SPLSs and a partial GTC for paths with non-tree edges. The general idea is to characterize paths in the graph into 2 cases: *case (1)* the first or the last edge in the path is a tree edge, and *case (2)* neither the first nor the last edge is a tree edge. A partial GTC containing SPLSs is pre-computed to deal with the reachability caused by the paths in *case (2)*. Reachability due to the paths in *case (1)* is processed by combining spanning trees and the partial GTC, *i.e.*, visiting the successors of the source and the predecessors of the target in a spanning tree and then for each possible successor-predecessor pair checking the partial GTC. Two optimization techniques are proposed to speed up query processing. The first optimization is based on interval labeling to efficiently find the successors and the predecessors in a spanning tree. The second optimization is recording the occurrences of individual edge labels in SPLSs of the paths from the root r of a spanning tree to each vertex in the spanning tree. Then, the SPLS of an s-t path in the spanning tree can be computed by subtracting the SPLS of the r-s path from the SPLS of the r-t path in the tree.

One of the state-of-the-art indexes is proposed by Chen et al. [12]. The approach uses spanning trees to classify edges and distinguishes the reachability caused by different classes. Specifically, after computing a spanning tree (or forest) T in G, directed edges (u, v) are grouped into four disjoint classes: (1) tree edge if (u, v) is an edge in T; (2) forward edge if v is a descendant of u in T; (3) back edge if v is an ancestor of u in T; and (4) cross edge if v is neither an ancestor nor descendant of u in T. Tree and forward edges are grouped together to form a tree-like structure, denoted as $\mathcal{T}$. The reachability information in $\mathcal{T}$ can be efficiently encoded by the interval labeling approach enriched with SPLSs. More precisely, as there is already a path in T for each forward edge in $\mathcal{T}$, the interval labeling built for T is able to encode the plain reachability information in $\mathcal{T}$. In order to efficiently record the SPLSs in $\mathcal{T}$, the second optimization technique in the approach discussed earlier [21] is adopted, *i.e.*, recording the SPLSs of paths from the root of $\mathcal{T}$ to each v in $\mathcal{T}$ and the occurrence of each individual edge label in each SPLS. Then, G is compressed by keeping only the vertices and edges that can transfer the reachability information caused by cross edges, which leads to a summary G_c. Thus, G is decomposed into $(\mathcal{T}, G_c)$. G_c is then taken as input, denoted as G^1, and further decomposed into $(\mathcal{T}^1, G_c^1)$, which are the tree-like structure and the graph summary of G_1, respectively. The approach applies a recursive decomposition method until the decomposition of G_i only contains a tree-like structure $\mathcal{T}^i$. The series of the decomposition results are used as the index of the approach. In each decomposition result $(\mathcal{T}^i, G_c^i)$, the reachability information caused by back edges in G_i are taken into account by performing online search with a optimization of chaining back edges, *e.g.*, for back edges (u, v) and (u', v'), if u' is a descendant of v in $\mathcal{T}^i$, then they are chained into a *back path* (u, v'). Finally, $Q_r(s, t, \alpha)$ is recursively decomposed into sub-queries that are evaluated on the series $(\mathcal{T}, \mathcal{T}^1, ...)$ with online search for back paths, and if an s-t satisfying α can be found by using $\mathcal{T}^i$, then query result *true* will be returned.

4.1.2 ***GTC-Based Indexes.*** The problem of efficiently computing a GTC for alternation-based reachability queries is extensively studied by Zou et al. [48, 56]. The fundamental step is to compute a single-source GTC, *e.g.*, computing all the vertices that are reachable from a source vertex and the corresponding SPLSs. A Dijkstra-like algorithm is proposed by using the number of distinct labels in path-label sets to simulate distances between vertices. Consider the computation of the single-source GTC from vertex L in Figure 1(b). H is reachable from L via two paths: p_3 = (L, worksFor, C, worksFor, H) and p_4 =(L, worksFor, D, friendOf, H). Path p_3 is 'shorter' than p_4 since p_3 has only 1 distinct label while p_4 has 2. Thus, p_3 is expanded to compute SPLS from L to G and p_4 is ignored. An input graph is first transformed into a DAG, and then the computation is done by following the topological order of the DAG so as to share the single-source GTC of vertices in a bottom-up manner. The transformation from a general graph to a DAG for alternation-based reachability is more complicated than the one for plain reachability discussed in Section 3, because paths in SCCs are not equivalent due to different SPLSs. Each SCC is replaced by a bipartite graph with *in-portal* and *out-portal* vertices. A vertex v in an SCC is in-portal (or out-portal) iff v has at least one incoming (or outgoing) edge from vertices out of the SCC. Then, the SPLSs from in-portal to out-portal vertices are computed and recorded in the index. In computing the reachability information within each SCC, the Dijkstra-like algorithm is further optimized by starting the computation from vertices of high degree; the single-source GTC of such vertices will be leveraged to prune search space for the subsequent computations. The index maintenance problem on dynamic graphs is also discussed.

A partial index with online BFS has been proposed [44]. The partial index is a set of single-source GTCs for a subset of vertices in the graph. The subset of vertices is selected as those in the top-k degree ranking, referred to as *landmark vertices*. $Q_r(s, t, \alpha)$ is processed by performing BFS with acceleration through leveraging the partial index when landmark vertices are hit during the traversal from s. If the single-source GTC of a landmark vertex v that is hit by the BFS from s contains t with a SPLS that can satisfy α, then the approach returns *true*. Otherwise, all the vertices that are reachable from the landmark v under the constraint α can be pruned in the subsequent search. In addition, the querying process is further improved by computing the reachability and SPLSs of paths from non-landmark vertices to landmark vertices, where the number of indexed paths is controlled by a predefined parameter.

4.1.3 ***2-Hop-Based Indexes.*** P2H+ [33] is a 2-hop-based reachability index for alternation-based reachability queries. The index utilizes the 2-hop index as the framework and adds SPLSs to deal with path constraints. The 2-hop indexing algorithm is extended

to build the P2H+ index by leveraging the transitivity of 2-hop reachability and SPLSs. The indexing algorithm of P2H+ performs backward and forward BFSs from vertices sorted in a total order according to vertex degree (from high degree to low degree), during which pruning rules are applied to skip redundant traversal. In addition, the indexing algorithm can guarantee that the built index does not contain any redundancy, which is achieved by prioritizing the visiting of edges with labels that are already contained in the SPLSs of the paths from v to current frontier vertices during the BFS from each vertex v.

DLCR [10], one of the state-of-the-art indexes, extends P2H+ to support graph updates, including edge insertions and deletions. When an edge is updated, DLCR will identify a set of nodes that need to perform backward and forward BFSs to reflect the update into the index. The BFS performed for edge updates only needs to traverse paths that contain the updated edges. During the BFS traversal, DLCR further identifies affected vertices that require additional index updates to either remove redundancy (in the case of edge insertion) or keep the index correct (in the case of edge deletion). In general, in the case of edge insertion, the newly inserted edge could make existing index entries redundant, which needs to be deleted. For each index entry *IE* in DLCR, there might be index entries that are redundant because of *IE* so that the set of such redundant index entries, referred to as *RIE*, is not included in DLCR. However, if the updates for reflecting the changes caused by a deleted edge require deleting *IE*, index entries in *RIE* might become non-redundant so that they need to be inserted into DLCR to keep the index correct.

4.2 Indexes for Concatenation-Based Queries

A concatenation of edge labels defines a sequence, and the corresponding path constraint in $Q_r(s, t, \alpha)$ enforces that the edge-label sequence of an s-t path must be an arbitrary number of repeats of the sequence of concatenated edge labels defined under the Kleene operator in α, *e.g.*, $Q_r(L, B, \texttt{worksFor} \cdot \texttt{friendOf})^*) = true$ in Figure 1(b). A concatenation-based reachability query is also known as *recursive label-concatenated* (RLC) query.

The problem of indexing concatenation-based reachability queries is first studied by Zhang et al. [52] which proposes the RLC index. To index such queries, the *minimum repeats* (MR) of edge-label sequence are recorded, *e.g.*, the MR of the path (L, `worksFor`, D, `friendOf`, H, `worksFor`, G, `friendOf`, B) is (`worksFor`, `friendOf`), and the MR is helpful in processing $Q_r(L, B, \texttt{worksFor} \cdot \texttt{friendOf})^*)$. The RLC index leverages the 2-hop index framework and records, additionally, MRs of edge-label sequences. Recording the MRs might be prohibitive due to the presence of infinite MRs of s-t paths as a result of directed cycles on the paths. In order to deal with this challenge, the concatenation length under the Kleene operator in α is leveraged to guide the computation of MRs. The second challenge is that MRs do not necessarily have the transitive property, *e.g.*, it is unfeasible to derive the MR of an s-t path given that the MRs of an s-u path and a u-t path are (l_1) and (l_2), respectively. Thus, the indexing algorithm of the RLC index separates the index building process into two phases: the first phase computes all the possible MRs for any (s, t), and the second phase selects the transitive MRs to build the index. Pruning rules are proposed to avoid redundant computation and recording in the index.

5 OPEN CHALLENGES

The state-of-the-art plain reachability indexes can be built for a large graph efficiently, *e.g.*, BFL can be built in a few seconds on graphs with millions of vertices, with an index size of only a few hundred megabytes. The main challenge is that these advanced indexing techniques, *i.e.*, partial indexes with guided graph traversal, are mainly designed for static graphs. The only approaches that can deal with graph updates are DAGGER [51], IP [47], and DBL [29]. However, the dynamic approach adopted in IP is based on DAGGER that could be no faster than BFS [55], and DBL can only support insertion-only graphs. Having partial indexes, which can process queries and deal with graph updates of insertions and deletions efficiently, is highly interesting.

A partial index only records partially reachability information in an input graph, so that they can scale to large graphs. $Q_r(s, t)$ can be processed by performing online traversal that can be guided by using index lookups in partial indexes. Let v be a current frontier vertex during the online traversal from s. In a partial index without false positives, if the index lookup for evaluating the reachability from v to t returns *true*, the online traversal can immediately terminate. In the case of a partial index without false negatives, the online traversal does not need to visit the outgoing neighbours of v if the index lookup for evaluating the reachability from v to t returns *false*. From the evolution of plain reachability indexing techniques over the last three decades, we observe that partial indexes play an important role in making reachability indexes scale to large graphs, *i.e.*, obtaining efficient query processing with realistic indexing cost. More importantly, partial indexes that do not have false negatives in query results using only index lookups (*e.g.*, GRAIL, Ferrari, IP, and BFL) have demonstrable advantages. That is because if a vertex t is not reachable from s, $Q_r(s, t)$ can stop immediately, and in real-world graphs there will be many such vertices s.

According to Table 2, the only partial index for path-constrained reachability queries is the landmark index. However, landmark index is a partial index without false positives, which means that if the index returns *false*, the online traversal for query processing has to continue. As noted above, in real-world graphs, it is likely that there will be many vertices that might not be reachable from a given source vertex, and, therefore, it would be interesting to have a partial index without false negatives for path-constrained reachability queries. Equally importantly, we observe that the index construction cost of path-constrained reachability indexes is high and takes hours of indexing for graphs with millions of vertices [10, 12, 33, 52]. This also calls for the design of partial indexes without false negatives. Analogously, practical issues of updates on dynamic graphs should be also taken into account for the integration into GDBMSs.

Another important challenge for path-constrained reachability indexes is that the existing solutions can only deal with a specific type of path constraint α, as shown in Table 2. However, real-world query log analysis [6] has shown that practical path constraints have many more types. It will be of great interest to have one indexing technique for general path constraints and thus the entire fragment of regular path queries [5].

Finally, the parallel computation of indexes (*e.g.*, parallel 2-hop indexing [22]) is also worth exploring.

REFERENCES

[1] Serge Abiteboul and Victor Vianu. 1997. Regular Path Queries with Constraints. In *Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*. 122–133. https://doi.org/10.1145/263661.263676

[2] R. Agrawal, A. Borgida, and H. V. Jagadish. 1989. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 253–262. https://doi.org/10.1145/67544.66950

[3] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5, Article 68 (2017), 40 pages. https://doi.org/10.1145/3104031

[4] Marc Barthélemy. 2011. Spatial networks. *Physics Reports* 499, 1 (2011), 1–101. https://doi.org/10.1016/j.physrep.2010.11.002

[5] Angela Bonifati, George Fletcher, Hannes Voigt, Nikolay Yakovets, and H. V. Jagadish. 2018. *Querying Graphs*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00873ED1V01Y201808DTM051

[6] Angela Bonifati, Wim Martens, and Thomas Timm. 2019. Navigating the Maze of Wikidata Query Logs. In *Proc. 28th Int. World Wide Web Conf.* 127–138. https://doi.org/10.1145/3308558.3313472

[7] Ramadhana Bramandia, Byron Choi, and Wee Keong Ng. 2010. Incremental Maintenance of 2-Hop Labeling of Large Graphs. *IEEE Trans. Knowl. and Data Eng.* 22, 5 (2010), 682–698. https://doi.org/10.1109/TKDE.2009.117

[8] Jing Cai and Chung Keung Poon. 2010. Path-Hop: Efficiently Indexing Large Graphs for Reachability Queries. In *Proc. 19th ACM Int. Conf. on Information and Knowledge Management*. 119–128. https://doi.org/10.1145/1871437.1871457

[9] Li Chen, Amarnath Gupta, and M. Erdem Kurul. 2005. Stack-Based Algorithms for Pattern Matching on DAGs. In *Proc. 31st Int. Conf. on Very Large Data Bases*. 493–504. https://dl.acm.org/doi/10.5555/1083592.1083651

[10] Xin Chen, You Peng, Sibo Wang, and Jeffrey Xu Yu. 2022. DLCR: Efficient Indexing for Label-Constrained Reachability Queries on Large Dynamic Graphs. *Proc. VLDB Endowment* 15, 8 (2022), 1645–1657. https://doi.org/10.14778/3529337.3529348

[11] Y. Chen and Y. Chen. 2008. An Efficient Algorithm for Answering Graph Reachability Queries. In *Proc. 24th Int. Conf. on Data Engineering*. 893–902. https://doi.org/10.1109/ICDE.2008.4497498

[12] Yangjun Chen and Gagandeep Singh. 2021. Graph Indexing for Efficient Evaluation of Label-constrained Reachability Queries. *ACM Trans. Database Syst.* 46, 2, Article 8 (2021), 50 pages. https://doi.org/10.1145/3451159

[13] James Cheng, Silu Huang, Huanhuan Wu, and Ada Wai-Chee Fu. 2013. TF-Label: A Topological-Folding Labeling Scheme for Reachability Querying in a Large Graph. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 193–204. https://doi.org/10.1145/2463676.2465286

[14] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. on Comput.* 32, 5 (2003), 1338–1355. https://doi.org/10.1137/S0097539702403098

[15] Klyne Graham, J. Carroll Jeremy, and McBride Brian. 2014. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation. (25 Feb. 2014). https://www.w3.org/TR/rdf11-concepts.

[16] Claudio Gutierrez and Juan F. Sequeda. 2021. Knowledge Graphs. *Commun. ACM* 64, 3 (2021), 96–104. https://doi.org/10.1145/3418294

[17] Haixun Wang, Hao He, Jun Yang, P. S. Yu, and J. X. Yu. 2006. Dual Labeling: Answering Graph Reachability Queries in Constant Time. In *Proc. 22nd Int. Conf. on Data Engineering*. 75–75. https://doi.org/10.1109/ICDE.2006.53

[18] Kathrin Hanauer, Christian Schulz, and Jonathan Trummer. 2022. O'Reach: Even Faster Reachability in Large Graphs. *ACM J. Exp. Algorithmics* 27, Article 4.2 (2022), 27 pages. https://doi.org/10.1145/3556540

[19] M.R. Henzinger and V. King. 1995. Fully dynamic biconnectivity and transitive closure. In *Proc. 36th Annual Symp. on Foundations of Computer Science*. 664–672. https://doi.org/10.1109/SFCS.1995.492668

[20] H. V. Jagadish. 1990. A Compression Technique to Materialize Transitive Closure. *ACM Trans. Database Syst.* 15, 4 (1990), 558–598. https://doi.org/10.1145/99935.99944

[21] Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. 2010. Computing Label-Constraint Reachability in Graph Databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 123–134. https://doi.org/10.1145/1807167.1807183

[22] Ruoming Jin, Zhen Peng, Wendell Wu, Feodor Dragan, Gagan Agrawal, and Bin Ren. 2020. Parallelizing Pruned Landmark Labeling: Dealing with Dependencies in Graph Algorithms. In *Proc. 34th Annual Int. Conf. on Supercomputing*. Article 11, 13 pages. https://doi.org/10.1145/3392717.3392745

[23] Ruoming Jin, Ning Ruan, Saikat Dey, and Jeffrey Yu Xu. 2012. SCARAB: Scaling Reachability Computation on Large Graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 169–180. https://doi.org/10.1145/2213836.2213856

[24] Ruoming Jin, Ning Ruan, Yang Xiang, and Haixun Wang. 2011. Path-Tree: An Efficient Reachability Indexing Scheme for Large Directed Graphs. *ACM Trans. Database Syst.* 36, 1, Article 7 (2011), 44 pages. https://doi.org/10.1145/1929934.1929941

[25] Ruoming Jin and Guan Wang. 2013. Simple, Fast, and Scalable Reachability Oracle. *Proc. VLDB Endowment* 6, 14 (2013), 1978–1989. https://doi.org/10.14778/2556549.2556578

[26] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 2009. 3-HOP: A High-Compression Indexing Scheme for Reachability Query. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 813–826. https://doi.org/10.1145/1559845.1559930

[27] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. 2008. Efficiently Answering Reachability Queries on Very Large Directed Graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 595–608. https://doi.org/10.1145/1376616.1376677

[28] Mikaela Koutrouli, Evangelos Karatzas, David Paez-Espino, and Georgios A Pavlopoulos. 2020. A Guide to Conquer the Biological Network Era Using Graph Theory. *Front. Bioeng. Biotechnol.* 8 (2020), 34. https://doi.org/10.3389/fbioe.2020.00034

[29] Qiuyi Lyu, Yuchen Li, Bingsheng He, and Bin Gong. 2021. DBL: Efficient Reachability Queries on Dynamic Graphs. In *Proc. 26th Int. Conf. on Database Systems for Advanced Applications*. 761–777. https://doi.org/10.1007/978-3-030-73197-7_52

[30] Nav Mathur. 2021. White Paper: Neo4j for Financial Services. https://neo4j.com/whitepapers/financial-services-neo4j.

[31] Florian Merz and Peter Sanders. 2014. PReaCH: A fast lightweight reachability index using pruning and contraction hierarchies. In *In Proc. 22th European Symp. on Algorithms*. 701–712. https://doi.org/10.1007/978-3-662-44777-2_58

[32] Mark Newman. 2010. *Networks: An Introduction*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199206650.001.0001

[33] You Peng, Ying Zhang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2020. Answering Billion-Scale Label-Constrained Reachability Queries within Microsecond. *Proc. VLDB Endowment* 13, 6 (2020), 812–825. https://doi.org/10.14778/3380750.3380753

[34] Liam Roditty. 2013. Decremental Maintenance of Strongly Connected Components. In *Proc. 24th Annual ACM-SIAM Symp. on Discrete Algorithms*. 1143–1150. https://doi.org/10.1137/1.9781611973105.82

[35] Liam Roditty and Uri Zwick. 2004. A Fully Dynamic Reachability Algorithm for Directed Graphs with an Almost Linear Update Time. In *Proc. 36th Annual ACM Symp. on Theory of Computing*. 184–191. https://doi.org/10.1145/1007352.1007387

[36] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2017. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *Proc. VLDB Endowment* 11, 4 (2017), 420–431. https://doi.org/10.1145/3186728.3164139

[37] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. 2020. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.* 29, 2 (2020), 595–618. https://doi.org/10.1007/s00778-019-00548-x

[38] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid G. Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. 2021. The future is big graphs: A community view on graph processing systems. *Commun. ACM* 64, 9 (2021), 62–71. https://doi.org/10.1145/3434642

[39] R. Schenkel, A. Theobald, and G. Weikum. 2005. Efficient creation and incremental maintenance of the HOPI index for complex XML document collections. In *Proc. 21st Int. Conf. on Data Engineering*. 360–371. https://doi.org/10.1109/ICDE.2005.57

[40] S. Seufert, A. Anand, S. Bedathur, and G. Weikum. 2013. FERRARI: Flexible and efficient reachability range assignment for graph indexing. In *Proc. 29th Int. Conf. on Data Engineering*. 1009–1020. https://doi.org/10.1109/ICDE.2013.6544893

[41] J. Su, Q. Zhu, H. Wei, and J. X. Yu. 2017. Reachability Querying: Can It Be Even Faster? *IEEE Trans. Knowl. and Data Eng.* 29, 3 (2017), 683–697. https://doi.org/10.1109/TKDE.2016.2631160

[42] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM J. on Comput.* 1, 2 (1972), 146–160. https://doi.org/10.1137/0201010

[43] Silke Trißl and Ulf Leser. 2007. Fast and Practical Indexing and Querying of Very Large Graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 845–856. https://doi.org/10.1145/1247480.1247573

[44] Lucien D.J. Valstar, George H.L. Fletcher, and Yuichi Yoshida. 2017. Landmark Indexing for Evaluation of Label-Constrained Reachability Queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 345–358. https://doi.org/10.1145/3035918.3035955

[45] Renê Rodrigues Veloso, Loïc Cerf, Wagner Meira Jr., and Mohammed J. Zaki. 2014. Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach. In *Proc. 17th Int. Conf. on Extending Database Technology*. 511–522. https://openproceedings.org/EDBT/2014/paper_166.pdf

[46] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. 2014. Reachability Querying: An Independent Permutation Labeling Approach. *Proc. VLDB Endowment* 7, 12 (2014), 1191–1202. https://doi.org/10.14778/2732977.2732992

[47] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. 2018. Reachability Querying: An Independent Permutation Labeling Approach. *VLDB J.* 27, 1 (2018), 1–26. https://doi.org/10.1007/s00778-017-0468-3

[48] Kun Xu, Lei Zou, Jeffery Xu Yu, Lei Chen, Yanghua Xiao, and Dongyan Zhao. 2011. Answering Label-Constraint Reachability in Large Graphs. In *Proc. 20th ACM Int. Conf. on Information and Knowledge Management*. 1595–1600. https://doi.org/10.1145/2063576.2063807

[49] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast and Scalable Reachability Queries on Graphs by Pruned Labeling with Landmarks and Paths. In *Proc. 22nd ACM Int. Conf. on Information and Knowledge Management*. 1601–1606. https://doi.org/10.1145/2505515.2505724

[50] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. 2010. GRAIL: Scalable Reachability Index for Large Graphs. *Proc. VLDB Endowment* 3, 1–2 (2010), 276–284. https://doi.org/10.14778/1920841.1920879

[51] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. 2013. DAGGER: A Scalable Index for Reachability Queries in Large Dynamic Graphs. arXiv:1301.0977

[52] Chao Zhang, Angela Bonifati, Hugo Kapp, Vlad Ioan Haprian, and Jean-Pierre Lozi. 2022. A Reachability Index for Recursive Label-Concatenated Graph Queries. arXiv:2203.08606

[53] Junfeng Zhou, Jeffrey Xu Yu, Yaxian Qiu, Xian Tang, Ziyang Chen, and Ming Du. 2023. Fast Reachability Queries Answering Based on RCNRCN Reduction. *IEEE Trans. Knowl. and Data Eng.* 35, 3 (2023), 2590–2609. https://doi.org/10.1109/TKDE.2021.3108433

[54] Junfeng Zhou, Shijie Zhou, Jeffrey Xu Yu, Hao Wei, Ziyang Chen, and Xian Tang. 2017. DAG Reduction: Fast Answering Reachability Queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 375–390. https://doi.org/10.1145/3035918.3035927

[55] Andy Diwen Zhu, Wenqing Lin, Sibo Wang, and Xiaokui Xiao. 2014. Reachability Queries on Large Dynamic Graphs: A Total Order Approach. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. 1323–1334. https://doi.org/10.1145/2588555.2612181

[56] Lei Zou, Kun Xu, Jeffrey Xu Yu, Lei Chen, Yanghua Xiao, and Dongyan Zhao. 2014. Efficient Processing of Label-Constraint Reachability Queries in Large Graphs. *Inf. Syst.* 40 (March 2014), 47–66. https://doi.org/10.1016/j.is.2013.10.003