Efficient Reachability Ratio Computation for 2-hop Labeling Scheme

Junfeng Zhou, Xian Tang, Ming Du

Abstract—As one of the fundamental graph operations, reachability queries processing has been extensively studied during the past decades. Many approaches followed the line of designing 2-hop labels to make acceleration. Considering that the index size cannot be bounded when using all nodes to construct 2-hop labels, researchers proposed to use a part of important nodes to construct 2-hop labels (partial 2-hop labels) to cover as much reachability information as possible. Then, we may achieve better query performance with limited index size and index construction time. However, partial 2-hop labels do not always perform well on different graphs.

In this paper, we focus on the problem of how to efficiently compute reachability ratio, such that to help users determine whether partial 2-hop labels should be used to answer reachability queries for the given graph. Intuitively, reachability ratio denotes the ratio of the number of reachable queries that can be answered by partial 2-hop labels over the total number of reachable queries involved in the given graph. We discuss the difficulties of reachability ratio computation, and propose an incremental-partition algorithm for reachability ratio computation. We show by rich experimental results that our algorithm can efficiently get the result of reachability ratio, and show how the overall query performance is affected by different partial 2-hop labels. Based on the experimental results, we give out our findings on whether partial 2-hop labels should be used to the given graph for reachability queries processing.

Index Terms—Reachability Queries Processing, 2-hop Labeling Scheme, Reachability Ratio

1 INTRODUCTION

Reachability queries processing is one of the fundamental graph operations and has been extensively studied in the literature [1]–[26]. Given a directed graph, a reachability query \( u \rightarrow v \) asks whether there exists a directed path from node \( u \) to \( v \). It can be used to Semantic Web (RDF), online social networks, biological networks, ontology, transportation networks, etc. To answer whether two nodes have a certain connection. It can also be used as a building brick in structured queries answering, such as XQuery or SPARQL.

To answer a given reachability query, researchers have proposed many efficient labeling schemes [1]–[26] to make acceleration, among which 2-hop labeling scheme has been widely adopted and was shown to be better than others in many cases [3], [4], [7], [9], [19], [20], [26]. Existing approaches that adopt 2-hop labels can be classified into two categories. The first kind of approaches [3], [4], [7], [19], [20] generate 2-hop labels based on all nodes, i.e., the 2-hop labels maintain the whole transitive closure (TC). For these approaches, a reachability query \( u \rightarrow v \) can be answered by comparing the 2-hop labels of \( u \) and \( v \) without graph traversal. However, the index size cannot be bounded w.r.t. the size of the input graph, and minimizing the size of 2-hop labels is NP-hard [4].

Different with [3], [4], [7], [19], [20], the second kind of approaches [9], [26] do not generate 2-hop labels based on all nodes, but based on a few nodes with large degree. We call these nodes as hop-nodes, and call 2-hop labels based on these hop-nodes as partial 2-hop labels. Compared with the first kind of approaches, the index size of partial 2-hop labels can be bounded, and is usually much smaller than that of the first kind of approaches. Even though partial 2-hop labels cannot answer all reachable queries in the whole TC, it was shown in [9], [26] that partial 2-hop labels can help improve the query performance significantly by answering most reachable queries for some graphs.

However, for some other graphs, the query performance may degenerate when using partial 2-hop labels [9], [26]. The reason lies in that the reachability ratio of partial 2-hop labels changes violently for different graphs. Here, reachability ratio means the ratio of the number of reachable queries that can be answered by partial 2-hop labels over the size of the TC. Figure 1 shows the reachability ratio of partial 2-hop labels on three graphs, from which we know that if we construct partial 2-hop labels using four hop-nodes with large degree, then the reachability ratio is greater than 90% on human and web-uk, meaning that the probability that a given reachable query \( q \) can be answered by partial 2-hop labels is greater than 90%, and is close to 0 on patent meaning that the probability that \( q \) can be answered by partial 2-hop labels is close to 0. In this case, using partial 2-hop labels brings us nothing but additional cost, which may degenerate the overall performance.

Therefore, before using partial 2-hop labels, a key problem that needs to be solved is: how to efficiently compute the reachability ratio of partial 2-hop labels w.r.t. the given graph? Because only if we know the reachability ratio, we can determine whether we should use it. For example, given the reachability ratio shown in Figure 1 we may decide to use partial 2-hop labels on human and web-uk, but not on patent, due to that using more hop-nodes on patent cannot increase the reachability ratio significantly. Furthermore, we can determine how many hop-nodes should be chosen to construct the partial 2-hop labels, if we decide to use partial 2-hop labels to accelerate the query performance. For example, according to the reachability ratio in Figure 1 we may decide to use four hop-nodes to
construct partial 2-hop labels on human, but for web-uk, reachability ratio changes little with the increase of hop-nodes and one hop-node is good enough, which means a larger reachability ratio and smaller index size.

To the best of our knowledge, this is the first work that addresses the problem of reachability ratio computation, which is not a trivial task and involves two operations. One is computing the size of TC, the other is computing the exact number of reachable queries that can be answered by the partial 2-hop labels, which we call as the coverage size. Considering that TC size computation can be efficiently solved by existing works [27], the difficulty of reachability ratio computation lies in how to efficiently compute the coverage size. The naive way is first generating partial 2-hop labels based on \( k \) selected hop-nodes, then getting the coverage size by checking all node pairs using the partial 2-hop labels. In this way, the cost of coverage size computation is \( O(k|V|^2) \) and cannot scale to large graphs, where \( V \) is the set of nodes in the input graph. Further, if the reachability ratio is too small to meet the requirement, we may need to increase \( k \)’s value and repeat the above operation, which makes reachability ratio computation more difficult to be solved.

We propose to compute the coverage size incrementally, such that when the value of \( k \) changes, we can avoid the costly coverage size recomputation, such that to support efficient reachability ratio computation. The basic idea is, given the coverage size w.r.t. \( k \) nodes, when we decide to compute the coverage size w.r.t. \( k + 1 \) hop-nodes, we do not compute the coverage size from scratch, but only compute the increased coverage size. However, the increased coverage size cannot be easily computed. To know the increased coverage size w.r.t. the \((k + 1)\)th hop-node \( u \), we need to firstly traverse from \( u \) to get a set of nodes \( D_u \) that \( u \) can reach, then traverse from \( u \) backwardly to get the second set of nodes \( A_u \) that can reach \( u \). Given \( A_u \) and \( D_u \), we need to check for each pair of nodes \((a, d)\), whether \( a \) can reach \( d \) can be determined by the current partial 2-hop labels without \( u \), where \( a \in A_u, d \in D_u \). If the answer is YES, then we know that \( a \) can reach \( d \) can be answered by the partial 2-hop labels without \( u \), and should not be considered when computing the increased coverage size w.r.t. \( u \). The cost of processing one hop-node \( u \) is as high as \( O(k|A_u||D_u|) \). Obviously, with the increase of the number of hop-nodes for partial 2-hop labels construction, the cost could be unaffordable. To this problem, we propose to divide both \( A_u \) and \( D_u \) into a set of disjoint subsets based on equivalence relationship (defined later), such that for each pair of subsets \( A_1 \subseteq A_u \) and \( D_1 \subseteq D_u \), we only need to test one reachability query, rather than \(|A_1| \times |D_1| \) queries. The cost of reachability ratio computation is, therefore, reduced significantly even when processing large graphs. We make the following contributions.

1) To the best of our knowledge, this is the first work to address the problem of reachability ratio computation.

2) We propose a set of algorithms for reachability ratio computation. We show that according to the properties of 2-hop labels, the two sets of nodes that can reach and be reached by a certain hop-node can be divided into a set of disjoint subsets, such that the computation cost can be reduced significantly. We prove the correctness and efficiency of our approach.

3) We conduct rich experiments on real datasets. The experimental results show that compared with the baseline approach, our algorithm works much more efficiently on reachability ratio computation. We also show how the overall query performance is affected by partial 2-hop labels with different number of hop-nodes, based on which we give out our findings on whether partial 2-hop labels should be used to the given graph for reachability queries processing.

The remainder of the paper is organized as follows. We discuss the preliminaries and the related work in Section 2. In Section 3, we give out the baseline algorithm for reachability ratio computation, and propose the first incremental algorithm in Section 4. After that, we propose the optimized incremental algorithm in Section 5. We report our experimental studies in Section 6, and conclude our paper in Section 7.

## 2 Background and Related Work

### 2.1 Preliminaries

Given a directed graph \( G \), we can construct a directed acyclic graph (DAG) \( \bar{G} \) from \( G \) in linear time [28] by coalescing each strongly connected component (SCC) of \( G \) into a node in \( \bar{G} \). Then, the reachability query on \( \bar{G} \) can be answered equivalently on \( G \). We follow the tradition and assume that the input graph is a DAG.

Given a DAG \( G = (V,E) \), where \( V \) is the set of nodes and \( E \) the set of edges. We define \( \text{in}(u) = \{v | (v, u) \in E\} \) as the set of in-neighbor nodes of \( u \) in \( G \), and \( \text{out}(u) = \{v | (u, v) \in E\} \) the set of out-neighbor nodes of \( u \). Similarly, we use \( \text{in}^*(u) \) to denote the set of nodes in \( G \) that can reach \( u \), and \( \text{out}^*(u) \) the set of nodes in \( G \) that \( u \) can reach. We say \( u \) can reach \( v \) (\( u \rightsquigarrow v \)), if \( v \in \text{out}^*(u) \).

The transitive closure \((TC)\) of \( G \) is \( G^* = (V, E^*) \), where \( E^* = \{(u, v) | u, v \in V, v \in \text{out}^*(u), v \neq u\} \). We define \( TC(u) = \text{out}^*(u) \setminus \{u\} \) as the transitive closure of \( u \), and define \( TC^{-1}(u) = \text{in}^*(u) \setminus \{u\} \) as the reverse TC of \( u \). The TC size of \( G \) is denoted as \( TC(G) = \sum_{u \in V} |TC(u)| \). In [27], the authors proposed an efficient algorithm for TC size computation with time complexity \( O(|r||E|) \), where \( r \) is the number of distinct paths decomposed from the input graph. In this paper, we assume that the TC size is given in advance, which can be got by executing the algorithm in [27] as an offline activity. Note that TC size computation is different with TC computation. The former computes \( |TC(v)| \) for all nodes, while the latter computes \( TC(v) \) for all nodes with time complexity \( O(|V| \times |E|) \).

Given a set of \( k \) nodes \( S_k \subseteq V \), we use \( L_k \) to denote 2-hop labels constructed based on nodes of \( S_k \), where each node in \( S_k \) is called a hop-node. If \( v \in TC(u) \) and \( L_k \) can correctly tell that \( u \rightsquigarrow v \), then we say \( L_k \) (or \( S_k \)) can cover the reachable query \( u \rightsquigarrow v \). Let \( N_k \) be the number of distinct reachable queries that can be covered by \( L_k \), the reachability ratio of \( S_k \) is defined as Equation 1.

Equation 1: \( N_k / TC(G) \)

### Problem Statement

Given a DAG \( G = (V, E) \), its TC size and a hop-node set \( S_k \subseteq V \), return the reachability ratio of \( S_k \).
TABLE 1: Notations

| Notation | Description |
|----------|-------------|
| $G = (V, E)$ | a DAG with a node set $V$ and an edge set $E$ |
| in$(v)$(out$(v)$) | the set of in-neighbors (out-neighbors) of $v$ |
| $\tau^*(v)$(out$(v)$) | the set of nodes that can reach (be reached by) $v$ |
| $TC^*(G)$ | the $TC$ size of $G$ |
| $S_k$ | a set of $k$ hop nodes |
| $L_{out}(v)$ | the 2-hop out (in) label of $v$ w.r.t. $S_k$ |
| $L_k$ | the partial 2-hop labels w.r.t. $S_k$ |
| $N_k$ | the number of reachable queries covered by $L_k$ |
| $A_k$ | ancestor set containing nodes that can reach $v_k$ |
| $D_k$ | descendant set containing nodes that $v_k$ can reach |

2.2 Related Work

As no existing works has addressed reachability ratio computation, we only discuss existing works on reachability queries processing. We discuss these approaches according to whether they use 2-hop labels to answer reachability queries.

2-hop based Approaches: Cohen et al. proposed to use 2-hop label [4] to answer reachability queries, wherein each node $u$ is assigned two labels, one is in-label $L_{in}(u)$, and the other is out-label $L_{out}(u)$. $L_{in}(u)(L_{out}(u))$ consists of a set of nodes $v$ that can reach (be reached by) $u$. Given the 2-hop label, the answering of a reachability query $u \leadsto v$ can be done by a set intersection operation on two labels, as indicated by Formula (2):

$$u \leadsto v = \begin{cases} \text{TRUE,} & L_{out}(u) \cap L_{in}(v) \neq \emptyset, \\ \text{FALSE,} & \text{otherwise} \end{cases}$$ (2)

Existing works involving 2-hop labeling scheme can be classified into two categories. The approaches in the first category construct 2-hop labels based on all nodes [3, 4, 7, 19, 20]. Considering that minimizing 2-hop label size is NP-hard [4], Cohen et al. proposed a ($\log |V|$)-approximate solution. However, the index construction cost is $O(|V||E| \log(|V|^2/|E|))$, which makes it difficult to scale to large graphs. Motivated by this, the following works [3, 7, 19, 20] have to discard the approximation guarantee and focused on finding better ordering strategy to rank nodes, such that to improve the efficiency of 2-hop label construction. Even though, the index size still cannot be bounded w.r.t. the size of the input graph.

Different with the above approaches, approaches in the second category [9, 20] generate partial 2-hop labels based on a hop-nodes to cover as more reachability relationships as possible. It was shown in [9, 20] that partial 2-hop labels can work very efficiently in answering reachability queries for some graphs, due to that the partial 2-hop labels can cover most reachability relationships for these graphs. Moreover, the index size of partial 2-hop labels can be bounded, and is usually much smaller than that of the first kind of approaches in practice. However, for some other graphs, they cannot work efficiently [9, 20]. The query performance may degenerate due to small reachability ratio for these graphs, as shown by Fig. 1. Therefore when considering partial 2-hop labels for reachability queries processing, its necessary that we can quickly know what is the reachability ratio w.r.t. a set of hop-nodes for the underlying graphs, such that we can correctly decide whether we should use partial 2-hop labels, and further, we can decide how many hop-nodes should be chosen to construct partial 2-hop labels.

Other Reachability Approaches: Besides approaches that use (partial) 2-hop labels, researchers also proposed other approaches that do not involve 2-hop labels, including [9, 12, 14–16, 18]. [21]. These approaches assign each node $u$ a label that maintains partial TC. For a given reachability query $u \leadsto v$, we may need to conduct depth-first search (DFS) or breadth-first search (BFS) from $u$ to check whether $v$ can reach $v$, if we cannot get the result by comparing labels of $u$ and $v$.

3 THE BASELINE ALGORITHM

To get the reachability ratio of $S_k$, we need to solve two tasks. One is constructing partial 2-hop labels, the other is computing the reachability ratio. In this section, we first analyze the construction of 2-hop labels and the computation of reachability ratio, then give out the baseline algorithm for reachability ratio computation.

Step-1: 2-hop Labels Construction. To construct 2-hop labels, existing approaches need to sort all nodes based on a certain rank value, such as degree [7, 20] or closeness [29]. The result of the sorting operation is $v_1, v_2, ..., v_{|V|}$, where the first (last) node has the largest (smallest) rank value. Based on the sorting result, we select the first $k$ nodes as hop-nodes to get the hop-node set $S_k = \{v_1, v_2, ..., v_k\}$. We have the following result w.r.t. the hop-node sets (Equations 3 and 4).

$$\emptyset = S_0 \subset S_1 \subset S_2 \subset ... \subset S_{|V|} = V$$ (3)

$$S_i \setminus S_{i-1} = \{v_i\}, \text{where } 0 < i \leq |V|$$ (4)

Given a set $S_i$ of $i$ hop-nodes, its 2-hop labels $L_i$ can be generated by processing $v_i$ based on $L_{i-1}$ according to Equations 3 and 4. Specifically, we first perform forward BFS from $v_i$ to get a set $D_i$ of nodes that $v_i$ can reach. Second, we perform backward BFS from $v_i$ to get a set $A_i$ of nodes that can reach $v_i$, as denoted by Figure 3(a). We call $A_i$ the ancestor set of $v_i$ and $D_i$ the descendant set of $v_i$. For each node $a \in A_i$, we add $v_i$ to $a$’s out-label, i.e., $L_{out}(a) = L_{out}^{-1}(a) \cup \{v_i\}$, denoting that $a$ can reach $v_i$. For each node $d \in D_i$, we add $v_i$ to $d$’s in-label, i.e., $L_{in}(d) = L_{in}^{-1}(d) \cup \{v_i\}$, denoting that $v_i$ can reach $d$. After processing $v_i$, we get 2-hop labels $L_i$. The superscript $i$ in $L_{out}(a)$/$L_{in}(a)$ denotes that both 2-hop labels $L_{out}(a)$ and $L_{in}(a)$ w.r.t. node $a$ are subsets of $S_i$, i.e., they contain only nodes of $S_i$. When $i = |V|$, then $S_i = V$, the 2-hop labels of $a$ are subsets of $V$. In this case, all nodes are hop-nodes and we do not use superscript in 2-hop labels for simplicity, i.e., $L_{out}(a) = L_{out}(a)$ and $L_{in}(a) = L_{in}(a)$.

It is worth noting that we can use 2-hop labels $L_{i-1}$ to reduce the size of both $A_i$ and $D_i$ by checking whether we can terminate the BFS traversal from $v_i$ in advance. For example, consider Figure 3(c), where both $v_{i-1}$ and $v_i$ are hop-nodes and $v_i$ is processed after $v_{i-1}$. After processing $v_{i-1}$, we have $L_{i-1}$. When processing $v_i$, the backward BFS traversal from $v_i$ can be terminated at $v_{i-1}$, due to that $v_{i-1}$ can reach $v_i$ can be answered by $L_{i-1}$, and $\forall a \in TC^{-1}(v_{i-1})$ that can reach $v_i$ through $v_{i-1}$ can also be answered by $L_{i-1}$. Therefore in practice, $A_i \subseteq in^*(v_i)$ and $D_i \subseteq out^*(v_i)$.

Example 1. Consider $G$ in Figure 3. Assume that we want to construct partial 2-hop labels $L_2$. The first thing we need to do is to sort all nodes by a certain rank value. In this paper, we follow the tradition [7, 20] and take $(out(v)) + 1 \times (in(v)) + 1$ as $v$’s rank value for sorting. The sorting result is $v_1, v_2, v_3, ..., v_{15}$. To get $L_2$, we first process $v_1$ by performing both forward and backward BFS from $v_1$ to get $A_1 = \{v_1, v_4, v_6, v_11\}$ and $D_1 = \{v_1, v_2, v_7, v_9, v_{10}, v_{13}, v_{15}\}$. After that, we add 1 to out label of nodes in $A_1$ and in label of nodes in $D_1$. Then, we get $L_1$. 
Step-2: Reachability Ratio Computation. Given 2-hop labels $L_k$ w.r.t. $S_k$, the baseline approach computes the reachability ratio of $S_k$ as follow. First, it computes the set of nodes that can reach either one of the set of hop-nodes, as shown by Equation 5. Second, it computes the set of nodes that can be reached by either one of the set of hop-nodes, as shown by Equation 6. It computes the number of reachable queries that can be answered by $L_k$, as shown by Equation 7. At last, we return the reachability ratio of 2-hop labels w.r.t. $S_k$ based on Equation 8.

\[ A = \bigcup_{i \in [1,k]} A_i \]  
\[ D = \bigcup_{i \in [1,k]} D_i \]  
\[ N_k = \left| \{(a,d) | a \in A, d \in D, a \neq d \right| \]  
\[ L_{out}^k(a) \cap L_{in}^k(d) \neq \emptyset \]  

Example 2. Continue Example 1. To compute the reachability ratio of $S_2 = \{v_1, v_2\}$, we first compute $A = A_1 \cup A_2 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, $D = D_1 \cup D_2 = \{v_1, v_2, v_7, v_9, v_{10}, v_{13}, v_{15}\}$ according to Equations 5 and 6, respectively. At last, we check for each pair of nodes $a \in A$ and $d \in D(a \neq d)$, whether $a$ can reach $d$ can be answered by $L_2$. And compute the number of answered reachable queries according to Equation 7 which is 42 for $G$ in Figure 3 and $L_2$ in Table 2. Given $TC(G) = 70$, we know that the reachability ratio of $S_2$ is $42/70 = 60\%$.

\[ TC(G) = \sum_{i=1}^{k} TC_i \]

The Algorithm: The baseline algorithm to compute reachability ratio is shown in Algorithm 1 which works in two steps. Step-1 (lines 1-17) constructs 2-hop labels $L_k$ of $k$ hop-nodes and gets the two set of nodes $A$ and $D$. Specifically, it first sorts all nodes in certain order in line 2, then selects $k$ hop-nodes in line 3. In lines 4-15, it performs forward and backward BFS from each hop-node $v_i$ to construct 2-hop labels $L_1$. During the processing, only if the reachability relationship between $v_i$ and the visited node $v$ cannot be answered by 2-hop labels $L_{i-1}$, it adds $v_i$ to $v$’s in-label (line 8) or out-label (line 13), and adds $v$ to $D_i$ (line 9) or $A_i$ (line 14); otherwise, it terminates the processing due to that the reachability relationship has already been covered by $L_{i-1}$. In lines 16-17, it gets the two sets $A$ and $D$ according to Equations 5 and 6. Step-2 (lines 18-20) computes the number of covered reachable queries by $L_k$ according to Equation 7. Finally, it computes and returns the reachability ratio in line 21.
Algorithm 1: brrR(G = (V, E), k, TC(G))

1. $N_k \leftarrow 0, S_k \leftarrow \emptyset, A \leftarrow \emptyset, D \leftarrow \emptyset$
2. rank nodes in $G$ in a certain order
3. put the first $k$ nodes into $S_k$ as hop-nodes
4. foreach $(v_i \in S_k)$ do
   5. $A_i \leftarrow \emptyset, D_i \leftarrow \emptyset$
   6. perform forward BFS from $v_i$, and for each visited $v$
      7. if $(L_i^{-1}(v) \cap L_i^{-1}(v)) = \emptyset$ then /*$L_i$*/
         8. $L_i^{-1}(v) \leftarrow L_i^{-1}(v) \cup \{v_i\}$ /*compute $L_i$*/
         9. $D_i \leftarrow D_i \cup \{v\}^1$
      10. else stop expansion from $v$
      11. perform backward BFS from $v_i$, and for each visited $v$
      12. if $(L_i^{-1}(v) \cap L_i^{-1}(v)) = \emptyset$ then /*$L_i$*/
         13. $L_i^{-1}(v) \leftarrow L_i^{-1}(v) \cup \{v_i\}$ /*compute $L_i$*/
         14. $A_i \leftarrow A_i \cup \{v\}$
      15. else stop expansion from $v$
      16. $A = \bigcup_{i \in [1,k]} A_i$
      17. $D = \bigcup_{i \in [1,k]} D_i$
5. foreach $(a \in A, d \in D, a \neq d)$ do
   18. if $(L_i^{-1}(a) \cap L_i^{-1}(d)) = \emptyset$ then /*$L_i$*/
      19. $N_k \leftarrow N_k + 1$
5. return $\alpha \leftarrow N_k/TC(G)$ as reachability ratio of $S_k$

Analysis: For Step-1 (lines 1-17), the time cost of line 2 is $O(|V|)$ by counting sort. The time cost of performing BFS from each-hop-node $v_i$ is $O(|V| + |E|)$ (lines 5-15). During the two BFS traversals, the time cost of processing every visited node $v$ is $O(k)$ (lines 7 and 12). Thus the time cost of 2-hop labels construction for each-hop-node is $O(k(|V| + |E|))$, and the time cost of processing $k$ hop-nodes, i.e., the time cost of Step-1 is $O(k^2(|V| + |E|))$.

For Step-2, the time cost is $O(|k||A||D|)$. Therefore, the time complexity of Algorithm 1 is $O(k^2(|V| + |E|) + |k||A||D|)$.

During the processing, we do not need to actually maintain every $A_i$ and $D_i$, instead, we only need to maintain $A$ and $D$. Further, we need to maintain the 2-hop labels w.r.t. $k$ hop-nodes, the space cost is $O(k|V|)$. As $S_k$, $A$ and $D$ are bounded by $V$, the space complexity of Algorithm 1 is $O(k|V|)$.

In practice, if the reachability ratio is too small to meet the requirement, we may need to use more hop-nodes, and therefore Algorithm 1 will be called once more to compute the new reachability ratio, for which all reachability relationships tested for $S_k$ will be tested again for the new hop-node set.

Example 3. Continue Example 2 After getting $A = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{13}, v_{15}\}$ and $D = \{v_1, v_2, v_7, v_9, v_{10}, v_{13}, v_{15}\}$ during constructing partial 2-hop labels, in lines 18-20, we need to test 56 reachability queries, due to $|A| = 8$ and $|D| = 7$. By line 21, we know that the reachability ratio is 60%. If we set the threshold of the reachability ratio to be equal or greater than 80%, then we need to enlarge the hop-node set and recompute the reachability ratio from scratch. As a result, the 56 queries tested for $S_2$ will be tested again for the new hop-node set.

4 The Incremental Approach

Considering that when the hop-node set is enlarged, Algorithm 1 will be called once more, and the set of reachability relationships tested for the first call will be tested again for the second call, a natural question is: can we compute the reachability ratio incrementally? That is, given the reachability ratio w.r.t. $S_{i-1}$, when we decide to compute the reachability ratio w.r.t. $i$ hop-nodes, i.e., $S_i = S_{i-1} \cup \{v_i\}$, we do not compute the number of covered reachable queries from scratch, instead, we only compute the number of increased reachable queries that cannot be covered by $L_{i-1}$, but can be covered by $L_i$.

However, the increased reachability ratio cannot be easily computed. On one hand, by constructing 2-hop labels using hop-node $v_i$, we capture three kinds of reachability relationships: (1) $v_j$ can reach every node in $D_i \setminus \{v_i\}$ can be determined by 2-hop labels w.r.t. $v_i$; and the number of covered reachable queries is $|D_i| - 1$; (2) every node in $A_i \setminus \{v_i\}$ can reach $v_i$ can be determined by 2-hop labels w.r.t. $v_i$, and the number of covered reachable queries is $|A_i| - 1$; and (3) each node in $A_i \setminus \{v_i\}$ can reach every node in $D_i \setminus \{v_i\}$ can be determined by 2-hop labels w.r.t. $v_i$, and the number of covered reachable queries is $|A_i| - 1 \times |D_i| - 1$. Thus the number of covered reachable queries by 2-hop labels w.r.t. $v_i$ can be computed as $|A_i| - 1 \times |D_i| - 1 + (|A_i| - 1) + (|D_i| - 1) = |A_i| \times |D_i| - 1$.

On the other hand, 2-hop labels w.r.t. different hop-nodes may cover the same reachable queries. For example, consider Figure 2(b), where $v_{i-1}$ and $v_i$ are two hop-nodes, and $v_i$ is processed after $v_{i-1}$. After processing $v_{i-1}$, every node $a_1 \in A_{i-1}$ can reach every node $d_1 \in D_{i-1}$ can be covered by 2-hop labels w.r.t. $v_{i-1}$, due to that $v_{i-1} \in L_{i-1}^{-1}(a_1) \cap L_{i-1}^{-1}(d_1)$. After processing $v_i$, we also know that $a_1$ can reach $d_1$ can also be covered by 2-hop labels w.r.t. $v_i$, due to that $v_i \in L_i^{-1}(a_1) \cap L_i^{-1}(d_1)$. Therefore, the increased number of reachable queries w.r.t. $v_i$ can be computed as Equations 8 and 9 and the total number of reachable queries $N_k$ covered by $L_k$ can be computed as Equation 10.

$$n_i = |A_i| \times |D_i| - 1 - \lambda$$

$$\lambda = \{|(a,d)|a \in A_i, d \in D_i, a \neq d, L_i^{-1}(a) \cap L_i^{-1}(d) = \emptyset\|$$

$$N_k = \sum_{i \in [1,k]} n_i$$

Therefore, to compute the number of reachable queries that cannot be covered by $L_{i-1}$ but can be covered by $L_i$, the intuitive way is firstly getting the two sets of nodes $A_i$ and $D_i$, then testing for each pair of nodes $a \in A_i$ and $d \in D_i$, whether $a$ can reach $d$ can be answered by $L_{i-1}$. If $a$ can reach $d$ can be answered by $L_{i-1}$, it means that $a \rightarrow d$ has already been covered by $S_{i-1}$; otherwise, it is a new covered reachable query and needs to be counted in, as shown by Algorithm 2.

In Algorithm 2 we compute reachability ratio for each $S_i$ when $v_i(i \in [1,k])$ is added into $S_{i-1}$. For each processed $v_i$ (lines 4-24), we first perform forward and backward BFS from $v_i$ to get the two set of nodes $A_i$ and $D_i$ (lines 5-13). In lines 14-17, we compute the number of reachable queries that can be covered by $L_{i-1}$. After that, we get the increased number of reachable queries that can be covered by $L_i$ but cannot be covered by $L_{i-1}$ in line 18 according to Equation 8. In line 19, we get the total number of reachable queries covered by $L_i$, and get the reachability ratio of $S_i$ in line 20. We compute $L_i$ based on $L_{i-1}$ in lines 21-24. At last, we return the reachability ratio of $S_k$ in line 25.

It is worth noting that for Algorithm 2 when processing the first hop-node $v_1$, we do not need to actually execute lines 15-17, due
Algorithm 2: incRR(G = (V, E), k, TC(G))
1: \(N_0 \leftarrow 0\)
2: rank nodes in \(G\) in a certain order
3: put the first \(k\) nodes into \(S_k\) as hop-nodes
4: \(\textbf{foreach} (v_i \in S_k) \textbf{do}\)
5: \(A_i \leftarrow \emptyset, D_i \leftarrow \emptyset\)
6: perform forward BFS from \(v_i\), and for each visited \(v\)
7: if \((L_{out}^{-1}(v_i) \cap L_{in}^{-1}(v)) = \emptyset\) then \(*/L_{i-1}*/\)
8: \(D_i \leftarrow D_i \cup \{v\}\)
9: else stop expansion from \(v\)
10: perform backward BFS from \(v_i\), and for each visited \(v\)
11: if \((L_{out}^{-1}(v) \cap L_{in}^{-1}(v_i)) = \emptyset\) then \(*/L_{i-1}*/\)
12: \(A_i \leftarrow A_i \cup \{v\}\)
13: else stop expansion from \(v\)
14: \(\lambda \leftarrow 0\)
15: \(\textbf{foreach} (a \in A_i, d \in D_i, a \neq d) \textbf{do}\)
16: if \((L_{out}^{-1}(a) \cap L_{in}^{-1}(d)) \neq \emptyset\) then \(*/L_{i-1}*/\)
17: \(\lambda \leftarrow \lambda + 1\)
18: \(n_i \leftarrow |A_i| \times |D_i| - 1 - \lambda\)
19: \(N_i \leftarrow N_{i-1} + n_i\)
20: \(\alpha \leftarrow N_i/TC(G)\)
21: \(\textbf{foreach} (a \in A_i) \textbf{do}\)
22: \(L_{out}^{-1}(a) \leftarrow L_{out}^{-1}(a) \cup \{v_i\}\)
23: \(\textbf{foreach} (d \in D_i) \textbf{do}\)
24: \(L_{in}^{-1}(d) \leftarrow L_{in}^{-1}(d) \cup \{v_i\}\)
25: return \(\alpha\) as reachability ratio of \(S_k\)

to that for \(v_1, A_1 = in^*(v_1)\) and \(D_1 = out^*(v_1)\), we can directly get \(n_1 = |A_1| \times |D_1| - 1\) and the corresponding reachability ratio.

Analysis: Different with Algorithm [1], Algorithm 2 performs Step-1 by first computing the two sets \(A_i\) and \(D_i\) in lines 1-13, then computing \(L_i\) in lines 21-24. The overall cost is same as that of Algorithm [1], i.e., \(O(k^2(|V| + |E|))\).

The difference between Algorithm [1] and Algorithm 2 lies in Step-2 (lines 14-20), i.e., how to compute the increased number of reachable queries that cannot be covered by \(L_{i-1}\) but can be covered by \(L_i\) based on Equation 8. The cost of Step-2 for each hop-node is \(O(|A_i||D_i|)\). For \(k\)-hop-node, the cost is therefore \(O(\sum_{i\in[1,k]} |A_i||D_i|)\).

Therefore, the time complexity of Algorithm 2 is \(O(k^2(|V| + |E|)) + \sum_{i\in[1,k]} |A_i||D_i|\).

Similar to Algorithm [1], we need to maintain the 2-hop labels w.r.t. at most \(k\) hop-nodes during the processing. As \(A_i\) and \(D_i\) are bounded by \(V\), and \((A_i(D_i))\) can be used to store nodes of \(A_{i+1}(D_{i+1})\), the space complexity of Algorithm 2 is \(O(k|V|)\).

Example 4. Consider \(G\) in Figure 3. Assume that we want to construct partial 2-hop labels \(L_3\).

The first node to be processed is \(v_1\), and the partial 2-hop labels are shown in Table 2 as \(A_1 = \{v_1, v_4, v_5, v_{11}\}\), \(D_1 = \{v_1, v_2, v_7, v_9, v_{10}, v_{13}, v_{15}\}\), thus we know that \(N_1 = n_1 = |A_1| \times |D_1| - 1 = 27\). The second processed node is \(v_2\) and the partial 2-hop labels are shown in Table 2. By lines 5-13, we have that \(A_2 = \{v_2, v_3, v_5, v_{12}\}\), \(D_2 = \{v_2, v_{10}, v_{13}, v_{14}\}\). Then, in lines 15-17, we need to test \(|A_2| \times |D_2| = 16\) reachability queries. The result is \(\lambda = 0\), thus \(n_2 = |A_2| \times |D_2| - 1 - 0 = 15\), and \(N_2 = n_1 + n_2 = 27 + 15 = 42\). The third processed node is \(v_3\). By lines 5-13, we have that \(A_3 = \{v_3, v_4, v_5, v_6, v_{11}\}\), \(D_3 = \{v_3, v_7, v_8, v_9, v_{14}\}\). Then, in lines 15-17, we need to test \(|A_3| \times |D_3| = 25\) reachability queries. The result is \(\lambda = 6\), thus \(n_3 = |A_3| \times |D_3| - 1 - 6 = 18\), and \(N_3 = n_2 + n_3 = 42 + 18 = 60\). After processing \(v_1, v_2, v_3, v_4\), we have \(L_3\) shown in Table 2 and the reachability ratio is \(60/70 = 85.7\%\) by testing 16 + 25 = 41 reachability queries for Algorithm 2.

As a comparison, when using Algorithm [1], \(|A_3| = 8, |D_3| = 10\), and we need to test 80 reachability queries to get the reachability ratio.

Note that, since a reachable query \(a \rightarrow d\) may be covered by 2-hop labels w.r.t. different hop-nodes, compared with Algorithm 1, Algorithm 2 may test the reachability relationship between \(a\) and \(d\) in line 16 more than once. However, it is still valuable due to that (1) only a part of reachable queries, rather than all, need to be tested more than once, and (2) we can terminate the computation whenever we find that reachability ratio of \(S_i\) is good enough in line 20. As a comparison, Algorithm [1] may be called more than once before getting a reachability ratio meeting the requirement. When it is called again due to the enlarged hop-node set, all previously tested queries will be tested once more.

5 THE INCREMENTAL-PARTITION APPROACH

5.1 The Equivalence Relationship

By comparing Algorithm 1 and Algorithm 2, we know that the key factor that affects the overall performance is the total number of tested reachability queries, which dominates the cost of Step-2, as indicated by their time complexities. Even though Algorithm 2 does not need to compute reachability ratio from scratch when the hop-node set becomes large by adding one more hop-node \(v_i\), it still needs to test \(|A_i| \times |D_i|\) reachability queries in line 16 with cost \(O(|i|A_i||D_i|)\). Given a large hop-node set, the cost could be unbearable.

Definition 1. [Equivalence Relationship] Given a hop-node \(v_i\), its ancestor set \(A_i\) and descendant set \(D_i\). We say two nodes \(a_1, a_2 (a_1 \neq a_2)\) of \(A_i\) are forward equivalent to each other, denoted as \(a_1 \equiv_F a_2\), if they have the same out-label, i.e., \(L_{out}(a_1) = L_{out}(a_2)\). Similarly, we say two nodes \(d_1, d_2 (d_1 \neq d_2)\) of \(D_i\) are backward equivalent to each other, denoted as \(d_1 \equiv_B d_2\), if they have the same in-label, i.e., \(L_{in}(d_1) = L_{in}(d_2)\).

By Definition 1, we can get, for \(A_i\), a partition \(A(i) = \{A_{i1}, A_{i2}, ..., A_{im}\}\), which consists of a set of \(m\) disjoint subsets satisfying that (1) \(\forall i, j, l \in [1, m], l \neq j, A_{il} \cap A_{ij} = \emptyset\) and \(\bigcup_{i\in[1,m]} A_{id} = A_i;\) and (2) \(\forall a_1, a_j\) belonging to the same subset, \(a_1 \equiv_F a_j\).

Theorem 1. Let \(a_1\) and \(a_2\) be two nodes satisfying that \(a_1\) and \(a_2\) are forward equivalent to each other (\(a_1 \equiv_F a_2\)). For \(\forall d \in V\), we have that \(L_{out}(a_1) \cap L_{in}(d) = L_{out}(a_2) \cap L_{in}(d)\).

Proof: The correctness is obvious, due to that \(a_1\) and \(a_2\) are forward equivalent to each other, which means that they have the same out-label.

Based on this result, for each subset \(A_{ij} \in A(i)\), to know the reachability relationship from all nodes of \(A_{ij}\) to \(\forall d \in V\), we do not need to test \(|A_{ij}|\) reachability queries, instead, we only need to
test one reachability query, due to that all nodes of \( A_{ij} \) are forward equivalent to each other.

For \( D_i \), we also have a partition \( D(i) = \{ D_{i1}, D_{i2}, \ldots, D_{in} \} \) satisfying that (1) \( \forall i, j \in [1, n], i \neq j, D_{il} \cap D_{lj} = \emptyset \) and \( \bigcup_{j=1}^{n} D_{ij} = D_i \); and (2) \( d_{il}, d_{lj} \) belonging to the same subset, \( d_{il} \equiv_B d_{lj} \). And similarly, for each subset \( D_{ij} \in D(i) \), to know the reachability relationship from any node to all nodes of \( D_{ij} \), we do not need to test \( |D_{ij}| \) reachability queries, instead, the number of tested reachability queries can be reduced to one, due to that all nodes of \( D_{ij} \) are backward equivalent to each other.

**Theorem 2.** Given a hop-node \( v_i \), its ancestor set \( A_i \) and descendant set \( D_i \), the number of tested reachability queries for reachability ratio computation is \( |A(i)| \times |D(i)| \), which is bounded by \( \min\{|A_i| \times |D_i|, 4^{-1}\} \).

**Proof:** Let \( A(i)(D(i)) \) be the partition of \( A_i(D_i) \) based on the equivalence relationship, \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) the partition of \( V \) w.r.t. hop-node set \( S_i \) and forward (backward) equivalence relationship, i.e., all nodes in each subset have the same out-label (in-label), which is a subset of \( S_i \). Initially, \( A(1) = \{A_i\} \) \( D(1) = \{D_i\} \), \( \mathcal{P}_A(1) = \{A_i, V \setminus A_i\} \) \( \mathcal{P}_D(1) = \{D_i, V \setminus D_i\} \).

On one hand, according to Theorem 1 for each subset \( A_{il} \in A(i) \), the result of testing all the reachability relationships from nodes of \( A_{il} \) to any other node is same to each other, thus we only need to randomly pick a node and take it as the representative node of \( A_{il} \) to perform the testing of reachability relationship. Similarly, for each subset \( D_{ij} \in D(i) \) based on backward equivalence relationship, we can also randomly pick a node and take it as the representative node of \( D_{ij} \) to test the reachability relationships from any node to all nodes of \( D_{ij} \). As a result, the number of tested reachability queries from nodes of \( A_i \) to nodes of \( D_i \) is \( |A(i)| \times |D(i)| \). Since \( A(i)(D(i)) \) is the partition of \( A_i(D_i) \), we know that \( |A(i)| \times |D(i)| \leq |A_i| \times |D_i| \).

On the other hand, given the partition \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) of \( V \), the size of \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) is at most twice bigger than that of \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \). The reason lies in that all nodes in each subset of \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) can be further divided into at most two disjoint subsets. One consists of nodes that can reach (be reached by) \( v_i \), and the other contains nodes that cannot reach (be reached by) \( v_i \). Then, the size of \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) is bounded by \( 2^2 \), and the size of \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) is bounded by \( 2^{2^{-1}} \), thus the number of tested reachability queries is bounded by \( 2^{2^{-1}} \times 2^{2^{-1}} = 4^{-1} \).

In summary, we know that the number of tested reachability queries for reachability ratio computation w.r.t. hop-node \( v_i \) is bounded by \( \min\{|A_i| \times |D_i|, 4^{-1}\} \).

**5.2 Partitions Computation**

To get the partitions of both \( A_i \) and \( D_i \), we need to compare the labels of nodes in \( A_i \) and \( D_i \), such that nodes with same labels can be clustered together. The naive way to do this is based on pairwise comparing node labels, which is expensive in practice. It is worth noting that the out-label and in-label of a node are sorted in advance, this actually can be done without additional cost, due to that these labels are used to check whether their set-intersection is empty, which means that we can store the processing order of hop-nodes, rather than their IDs, in these labels. In this way, the integers in both out-label and in-label of any node are naturally sorted, as shown by Table 2. With this result, we can sort all nodes in \( A_i(D_i) \) by comparing their out-labels (in-labels) in lexicographic order. After the sorting operation, all equivalent nodes are clustered together. As the size of each label is bounded by \( i \), the cost of computing the partition \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) of \( A_i(D_i) \) is \( O(i \times |A_i| \times \log |A_i|)(O(i \times |D_i| \times \log |D_i|)) \).

Let \( \mathcal{P}_A(i) \) be the partition of \( V \) w.r.t. hop-node set \( S_i \) and forward equivalence relationship, i.e., all nodes in each subset have the same out-label, which is a subset of \( S_i \), \( \mathcal{P}_D(i) \) the partition of \( V \) w.r.t. \( S_i \) and backward equivalence relationship, i.e., all nodes in each subset have the same in-label, which is also a subset of \( S_i \). We have the following result.

**Theorem 3.** Given the hop-node \( v_i \) and its ancestor (descendant) set \( A_i(D_i) \), for \( \forall v_1, v_2 \in A_i(D_i) \), \( v_1 \equiv_B v_2(\equiv_B v_2) \), iff they belong to the same subset of \( \mathcal{P}_A(i-1)(\mathcal{P}_D(i-1)) \).

**Proof:** We prove this result from two aspects. First, we prove the correctness when both \( v_1 \) and \( v_2 \) belong to \( A_i \) (Case-1), then we prove the correctness when both \( v_1 \) and \( v_2 \) belong to \( D_i \) (Case-2).

**Case-1** where \( v_1, v_2 \in A_i \) and \( v_i \in L_{out}^i(v_1) \cap L_{out}^i(v_2) \). On one hand, if \( v_1 \equiv_B v_2 \), which means that \( L_{out}^i(v_1) = L_{out}^i(v_2) \) according to Definition 1. Hence, \( L_{out}^i(v_1) \setminus \{v_i\} = L_{out}^i(v_2) \setminus \{v_i\} \), i.e., they belong to the same subset of \( \mathcal{P}_A(i-1) \).

On the other hand, if both \( v_1 \) and \( v_2 \) belong to the same subset of \( \mathcal{P}_A(i-1) \), it means that before processing hop-node \( v_i \), \( L_{out}^i(v_1) = L_{out}^i(v_2) \) according to the definition of \( \mathcal{P}_A(i-1) \). As \( v_1, v_2 \in A_i \), we know that after processing \( v_i \), \( v_1 \in L_{out}^i(v_1) \cap L_{out}^i(v_2) \) and \( L_{out}^i(v_1) = L_{out}^i(v_2) \) still holds. According to Definition 1 \( v_1 \equiv_B v_2 \).

Therefore we have that \( v_1 \equiv_B v_2 \), iff they belong to the same subset of \( \mathcal{P}_A(i-1) \).

**Case-2** where \( v_1, v_2 \in D_i \) and \( v_i \in L_{in}^i(v_1) \cap L_{in}^i(v_2) \). Similar to the proof of Case-1, we know that \( v_1 \equiv_B v_2 \), iff they belong to the same subset of \( \mathcal{P}_D(i-1) \).

According to Theorem 3 we assign each node \( v \) two set IDs, denoted as \( id_A(v) \) and \( id_D(v) \), which are used to check which subset it belongs to in \( \mathcal{P}_A(i) \) and \( \mathcal{P}_D(i) \), respectively. Then, given the ancestor (descendant) set \( A_i(D_i) \) of \( v_i \), we only need to scan all nodes of \( A_i(D_i) \) once, and know immediately that for two nodes \( v_1 \) and \( v_2 \), if \( id_A(v_1) = id_A(v_2) = id_A(v_1) = id_A(v_2) \) in \( \mathcal{P}_A(i-1)(\mathcal{P}_D(i-1)) \), then \( v_1 \equiv_B v_2(\equiv_B v_2) \) and will definitely belong to the same subset of \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \). Therefore, \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) is a refinement of \( \mathcal{P}_A(i-1)(\mathcal{P}_D(i-1)) \), i.e., each element of \( \mathcal{P}_A(i)(\mathcal{P}_D(i)) \) is a subset of a unique element of \( \mathcal{P}_A(i-1)(\mathcal{P}_D(i-1)) \).
Recall that when processing the hop-node $v_i$, we first have its ancestor (descendant) set $A_i(D_i)$, then get the partition $A(i)\{D(i)\}$ of $A_i(D_i)$ based on equivalence relationship. Since $\mathcal{P}_A(i)\{D_P(i)\}$ is the partition of $V$ w.r.t. equivalence relationship, we know that $A(i) \subseteq \mathcal{P}_A(i)\{D(i) \subseteq \mathcal{P}_D(i)\}$, and the relationship between $\mathcal{P}_A(i)(\mathcal{P}_D(i))$, $\mathcal{P}_A(i)\{(\mathcal{P}_D(i) - 1)\}$ and $A(i)(\mathcal{D}(i))$ are shown as Equations (12) and (13).

$$\mathcal{P}_A(i) = \{P \setminus A_i|P \in \mathcal{P}_A(i - 1)\} \cup A(i)$$ \hspace{1cm} (12)

$$\mathcal{P}_D(i) = \{P \setminus D_i|P \in \mathcal{P}_D(i - 1)\} \cup \mathcal{D}(i)$$ \hspace{1cm} (13)

When processing hop-node $v_i$, since we only need to check the reachability relationships from nodes of $A_i$ to $D_i$, we choose to maintain the information of both $\mathcal{P}_A(i)(\mathcal{P}_D(i))$ and $A(i)(\mathcal{D}(i))$ using the set ID of each node to facilitate partitions computation. Specifically, we use a hash table $H_A(D_A)$ to help achieve linearity complexity. Each element of $H_A(D_A)$ is a tuple $(i_d, e_n)$ denoting a subset $A(i_d)$ of $\mathcal{A}(\mathcal{D}(i))$, where $i_d$ is, for all nodes of $A(i_d)$, their set ID in $\mathcal{P}_A(i - 1)(\mathcal{P}_D(i - 1))$, $e_n = (i_d, v_n, s)$ is a triple denoting the new set ID for all nodes of $A(i_d)$, the representative node of $A(i_d)$, and the size of $A(i_d)$, respectively.

**Example 5.** Consider $G$ in Figure 3. Before processing $v_1$, $\mathcal{P}_A(0) = \mathcal{P}_D(0) = \{V\}$, $A(0) = \mathcal{D}(0) = \emptyset$, and for all nodes $v$, $i_dA(v) = i_dD(v) = 0$. For the first node $v_1$, $A_1 = \{v_1, v_4, v_6, v_1\}$, $D_1 = \{v_1, v_2, v_3, v_9, v_{10}, v_{13}, v_{15}\}$. Since all nodes in $A_1(D_1)$ have the same $i_dA(v)$($i_dD(v)$), we know that $A(1) = \{A_1\}$ and $\mathcal{D}(1) = \{D_1\}$. $\mathcal{P}_A(1) = \{A_1, V \setminus A_1\}$ and $\mathcal{P}_D(1) = \{D_1, V \setminus D_1\}$. In Table 3 the two columns under $v_1$ denote $\mathcal{P}_A(1)$ and $\mathcal{P}_D(1)$, where each 1 in the second (third) column corresponds a node in $A_1(D_1)$. Figure 3a shows the two hash tables denoting $\mathcal{A}(1)$ and $\mathcal{D}(1)$, respectively. For $H_A$, there is one (key, value) pair, denoting that $A(1)$ contains one subset $A_1$, and for all nodes in $A_1$, their set ID is 0 in $\mathcal{P}_A(0)$, thus they all belong to the same subset in $\mathcal{A}(1)$, i.e., $A(1) = \{A_1\}$. By $H_A$ in Figure 3a, we know that all nodes in $A_1$ now have the new set ID 1, the representative node of $A_1$ is $v_4$, and $|A_1| = 4$. For the second processed node $v_2$, $A_2 = \{v_2, v_3, v_9, v_{12}\}$, $D_2 = \{v_2, v_{10}, v_{13}, v_{14}\}$. As all nodes in $A_2$ have the same set ID 0 in $\mathcal{P}_A(1)$, $\mathcal{A}(2)$ contains a unique subset $A_2$, i.e., $A(2) = \{A_2\}$. As shown by Figure 3b, the key is 0, and the triple $(2, v_3, 4)$ denotes that the new set ID for all nodes in $A_2$ is 2, the representative node of $A_2$ is $v_3$, and $|A_2| = 4$. Similarly, all nodes in $D_2$ have the same set ID 1 in $\mathcal{P}_D(1)$, $\mathcal{D}(2)$ contains a unique subset $D_2$, i.e., $D(2) = \{D_2\}$, which is denoted as $H_D$ in Figure 3b.

For the third processed node $v_3$, $A_3 = \{v_3, v_4, v_5, v_6, v_{11}\}$, $D_3 = \{v_3, v_7, v_8, v_9, v_{14}\}$. For $A_3$, $v_3$ and $v_5$ have the same set ID 2 in $\mathcal{P}_A(2)$, thus they form the subset in $\mathcal{A}(3)$. Further, $v_4, v_6, v_{11}$ have the same set ID 1 in $\mathcal{P}_A(2)$, they form the second subset in $\mathcal{A}(3)$. Therefore, $\mathcal{A}(3) = \{\{v_3, v_4, v_5, v_6, v_{11}\}\}$. Similarly, we know that $\mathcal{D}(3) = \{\{v_3, v_4, v_5, v_6, v_{11}\}\}$. Both $\mathcal{A}(3)$ and $\mathcal{D}(3)$ are denoted by $H_A$ and $H_D$ in Figure 3c, respectively.

| Table 3: The status of set IDs for all nodes. |
|---|
| Node | $\mathcal{id}_A(v)$ | $\mathcal{id}_D(v)$ | $\mathcal{id}_A(v)$ | $\mathcal{id}_D(v)$ | $\mathcal{id}_A(v)$ | $\mathcal{id}_D(v)$ |
| $v_1$ | 1 | 1 | 1 | 1 | 1 | 1 |
| $v_2$ | 1 | 2 | 2 | 2 | 2 | 2 |
| $v_3$ | 2 | 3 | 3 | 3 | 3 |
| $v_4$ | 1 | 1 | 1 | 4 |
| $v_5$ | 2 | 3 | 3 | 3 |
| $v_6$ | 1 | 1 | 1 | 4 |
| $v_7$ | 1 | 1 | 1 | 4 |
| $v_8$ | 2 | 3 | 3 | 3 |
| $v_9$ | 1 | 1 | 1 | 4 |
| $v_{10}$ | 2 | 3 | 3 | 3 |
| $v_{11}$ | 1 | 1 | 1 | 4 |
| $v_{12}$ | 2 | 3 | 3 | 3 |
| $v_{13}$ | 2 | 3 | 3 | 3 |
| $v_{14}$ | 2 | 3 | 3 | 3 |
| $v_{15}$ | 2 | 3 | 3 | 3 |

**The Algorithm:** As shown by Algorithm 2 for each hop-node $v_i$, we first perform forward and backward BFS to get $D_i$ (lines 6-15) and $A_i$ (lines 16-25). At the same time, we generate their partitions $\mathcal{D}(i)$ and $\mathcal{A}(i)$, for which each subset is recorded in $H_D$ and $H_A$, respectively. In lines 26-29, we compute $\lambda$ according to Equation 14, which is the number of reachable queries that are covered by $\mathcal{L}_t$. In line 30, we get the number of reachable queries that can be covered by $\mathcal{L}_t$, but cannot be covered by $\mathcal{L}_{t-1}$. After that, we have the total number of covered reachable queries in line 31, and the reachability ratio in line 32. Last, we generate $\mathcal{L}_t$ in lines 33-36, and return the reachability ratio of $S_k$ in line 37.

**Analysis:** Same as Algorithm 1 and Algorithm 2. Algorithm 3 completes Step-1 by performing both forward and backward BFS from each hop-node, during which it first computes the two sets $A_i$ and $D_i$ in lines 1-25, and at the same time computes $H_A$ and $H_D$. Then, it computes the new partial 2-hop labels $\mathcal{L}_t$ in lines 33-36. The time cost is $O(k^2(|V| + |E|))$. Different with Algorithm 1 and Algorithm 2, the benefits of Algorithm 3 lie in Step-2. The cost of Step-2 for each hop-node is $O(|i\mathcal{A}(i)| + |\mathcal{D}(i)|)$. For hop-node, the cost is $O(|\sum_{i \in [1,k]} i\mathcal{A}(i)| + |\mathcal{D}(i)|)$. Therefore, the time complexity of Algorithm 3 is $O(k^2(|V| + |E|) + \sum_{i \in [1,k]} |i\mathcal{A}(i)| + |\mathcal{D}(i)|)$. Simlar to Algorithm 1, we need to maintain the 2-hop labels w.r.t. at most $k$ hop-nodes during the processing. As $A_i$, $D_i$, $H_A, H_D$ are bounded by $V$, and $A_i(D_i)$ can be used to store nodes of $A_{i+1}(D_{i+1})$, the space complexity of Algorithm 3 is $O(k|V|)$.
Table 5: Statistics of datasets, where $d = 2|E|/|V|$ is the average degree of $G$, $TC(\cdot)$ is the average number of reachable nodes for nodes of $G$, and $n_v$ is the number of topological levels (the length of the longest path) of $G$.

| Dataset   | $|V|$ | $|E|$ | $d$ | $TC(\cdot)$ | $n_v$ |
|-----------|------|------|-----|------------|------|
| amaze     | 3,710| 3,600| 1.94| 639        | 16   |
| human     | 38,811| 39,576| 2.04| 9          | 18   |
| anthra    | 12,499| 13,104| 2.10| 12         | 16   |
| agrocycle | 12,684| 13,408| 2.11| 13         | 16   |
| ecoc                | 12,620| 13,350| 2.12| 14         | 22   |
| vchosy     | 9,491| 10,143| 2.14| 14         | 21   |
| kegg       | 3,017| 3,908| 2.16| 729        | 26   |
| arxiv      | 6,000| 66,707| 22.24| 928   | 167  |
| email      | 231,000| 223,004| 1.93| 11,698    | 7    |
| url        | 971,252| 1,024,140| 2.11| 206,907   | 24   |
| web        | 371,764| 517,805| 2.79| 55,055    | 34   |
| 10cit-Patent | 1,097,775| 1,651,894| 4.01| 3          | 7    |
| 10citseeerx | 770,539| 1,501,126| 3.90| 70         | 36   |
| 05cit-Patent | 1,671,488| 3,303,789| 3.95| 8          | 12   |
| 05citseeerx | 1,457,057| 3,002,252| 4.12| 116        | 36   |
| citeseerx  | 6,540,401| 15,011,260| 4.59| 15,510    | 39   |
| doddapia   | 3,365,623| 7,989,191| 4.75| 83,659    | 146  |
| patent     | 3,774,768| 16,518,947| 8.75| 1,544      | 22   |
| twitter    | 18,121,168| 18,359,487| 2.05| 1,346,820 | 22   |
| web-uk     | 22,753,644| 38,184,039| 3.36| 3,417,930 | 2793 |

Table 4: Comparison of time and space complexities, where $A = \cup_{i \in [1,k]}A_i$, $D = \cup_{i \in [1,k]}D_i$, and $A(i)(D(i))$ is the partition of $A_i(D_i)$.

| Algorithm     | Time Complexity of Step-2 | Space Complexity |
|---------------|---------------------------|------------------|
| bRR           | $O(k|A(D)|)$              | $O(k|V|)$        |
| incRR         | $O_{\sum_{i \in [1,k]}|A_i||D_i|}$ | $O(k|V|)$        |
| incRR+        | $O_{\sum_{i \in [1,k]}A(i)(|D(i)|)}$ | $O(k|V|)$        |

Comparison of their time and space complexities. For time complexity, we do not show the cost of Step-1, due to that for Step-1, the cost is same for all three algorithms. We will show in Experiment that the incRR+ algorithm works much more efficiently than the bRR and incRR algorithms.

Example 6. Consider $G$ in Figure 3. Assume that we want to compute the reachability ratio of $S_3 = \{v_1, v_2, v_3\}$.

For $v_1$, as it is the first processed node, there is no covered reachability relationship, thus we do not need to test any reachability relationship in lines 28. As $|A_1| = 4$, $|D_1| = 7$, thus $n_1 = |A_1| \times |D_1| - 1 = 27$ in line 30 of Algorithm 3.
Datasets: Table 5 shows the statistics of 20 real datasets, where the first eight are small datasets (|V| < 100, 000) downloaded from the same web page. The following 12 datasets are large ones (|V| > 100, 000). These datasets are usually used in the recent works w.r.t. reachability queries processing. Among these datasets, amaze and kegg are metabolic networks, human, antara, agrocyc, eeco, vchocyc are graphs describing the genome and biochemical machinery of E. coli K-12 MG1655. email is an email network. LJ is an online social network soc-LiveJournal is a web graph web-Google, arxiv, 10cit-Patent, 10citeseerx, 05cit-Patent, 05citeseex, citeseerx, and patent are all citation networks. dbpedia is a knowledge graph Dbpedia. twitter is a DAG transformed from a large-scale social network obtained from a crawl of twitter.com. web-uk is a DAG of a web graph dataset. For these datasets, email, LJ, web, and the first seven small graphs are directed graphs initially. We transform each of them into a DAG by coalescing each strongly connected component into a node. Note that this can be done in linear time. All other datasets are DAGs initially. The statistics in Table 5 are that of DAGs.

6.1 Reachability Ratio Computation

Reachability Ratio and Index Size: We show the reachability ratio (RR) and the index size ratio (ISR) of the 20 real datasets in Figure 5, where ISR denotes the ratio of the size of partial 2-hop labels w.r.t. k hop-nodes over the size of the 2-hop labels w.r.t. all nodes. From Figure 5, we have the following observation.

First, we can classify all datasets into three categories according to the value of their reachability ratio. The first kind of datasets (D1) includes amaze, kegg, email, LJ, web, citeseex, dbpedia, twitter, and web-uk, for which the RR is more than 99% even when k = 1, and both the RR and ISR almost do not change with the increase of k. The second kind of datasets (D2) includes human, antara, agrocyc, eeco, vchocyc, and arxiv, for which both RR and ISR will increase with larger increase of k. The third kind of datasets (D3) includes 10cit-Patent, 10citeseex, 05cit-Patent, 05citeseex, and patent, for which both RR and ISR are very small or even approach zero; and with the increase of k, both RR and ISR almost do not change. The value of k, therefore, only affects the second kind of datasets, and the reachability ratio is more than 80% when k ≥ 16 for all datasets of the second kind, which indicates that we may benefit from using partial 2-hop labels on datasets of both the first and second kinds.

Second, the storage space used to maintain partial 2-hop labels is small compared with the reachability ratio value. For example, for the first kind of datasets, we can use about 1/4 storage space (ISR ≈ 25%) to maintain more than 99% (RR > 99%) reachability information.

Running Time: We show in Figure 6 the comparison of running time for reachability ratio computation, from which we have the following observations.

First, incRR+ is much faster than both blRR and incRR on all datasets, and incRR works faster than blRR on most datasets. For instance, incRR+ is faster than blRR by more than two or three orders of magnitude on most datasets, and incRR is ten times faster than blRR on amaze, email, LJ, web, citeseex, and dbpedia. The reason can be explained as follows. On one hand, Figure 5 shows the reachability ratio of different graphs w.r.t. different k. From Figure 5, we know that for amaze, human, antara, agrocyc, eeco, vchocyc, kegg, arxiv, email, LJ, web, citeseex, dbpedia, twitter and web-uk, the reachability ratio is more than 80% when k = 32 for all datasets. On the other hand, according to the last column of Table 5, we know that the average number of reachable nodes for nodes of each graph is usually big. Therefore, the number of tested reachability queries is blRR significantly large. Even though incRR can reduce the number of tested reachability queries, it still needs to test much more reachability queries than incRR+ for some datasets. For example, consider the number of tested reachability queries on kegg dataset when k = 32. The tested number of reachability queries of blRR (incRR) is 100 (10) times more than that of incRR+. Moreover, blRR and incRR cannot get the value of reachability ratio on both twitter and web-uk for k ≥ 2 in limited time (24 hours), due to testing too many reachability queries.

Second, both blRR and incRR work efficiently on datasets where the reachability ratio is small. For instance, from Figure 5, we know that incRR+ is faster than blRR and incRR by less than ten times on 10cit-Patent, 10citeseex, 05cit-Patent and 05citeseex, and patent. The reason lies in that for these datasets, the reachability ratio is very small according to Figure 5 which means that for all algorithms, the number of tested reachability queries is much less than other datasets, therefore does not need to consume more time.

It is worth noting that when k = 1, the three algorithms consume similar time. The reason is that when k = 1, for the first hop-node v1, after we get A−v1 and D−v1, we immediately know the number of reachability queries covered by L1 = |A−v1| × |D−v1| − 1, and therefore do not need to actually test any reachability queries.

By the above experimental results, we know that our incRR+ algorithm can be used to efficiently compute the reachability ratio for a given dataset, which brings us a chance to determine whether we should use partial 2-hop labels to facilitate reachability queries processing.

6.2 Reachability Queries Processing

In this section, we combine partial 2-hop labels with the state-of-the-art algorithm, namely FELINE [12] (abbreviated as FL), to show the impact of partial 2-hop labels on reachability queries processing, in terms of index size, index construction time and query time. The experimental results are shown, respectively, in Table 6 Table 7 and Table 8, where FL-k denotes the FL algorithm combined with partial 2-hop labels that are generated based on k hop-nodes. Hence, FL-0 is the FL algorithm without partial 2-hop labels. Note that for reachability queries processing, we do not set k = 1, 2, 4, 8, due to that when k = 16, we only need to use one integer as a bit-vector for each node v to represent both L16out(v) and L16in(v).

Index Size: Table 6 shows the impacts of k on index size, from which we know that with the increase of k, the index size will increase accordingly. For example, for web-uk dataset, the index size of FL-128 is more than two times bigger than that of FL-0 on all datasets. The reason is obvious. The larger the value of k, the more the space we need to maintain the partial 2-hop labels.

Index Construction Time: Figure 7 shows the impacts of k on index construction time, from which we know that with the increase of k, we need more time for index construction. Note that partial 2-hop labels can be constructed efficiently, and the increased time for
Fig. 5: Comparison of Reachability Ratio (RR) and Index Size Ratio (ISR), where ISR is ratio of the index size of partial 2-hop labels w.r.t. $k$ hop nodes over that of the total 2-hop label size w.r.t. all nodes.
Fig. 6: Running Time of different algorithms on reachability ratio computation (ms).
index construction could be omitted, due to that index construction is a one-time activity performed off-line for reachability queries processing.

**Query Time:** We report the query time about equal workload, which contains 1,000,000 reachability queries for each dataset. The equal workload consists of 50% reachable queries and 50% unreachable queries. The reason that we use equal workload is: using completely random queries is heavily skewed towards unreachable queries \([15, 16]\), which is highly unlikely for the real workload as the node pair in a query tends to have a certain connection \([5]\).

Here, unreachable queries are generated by sampling node pairs with the same probability until we reach the required number of unreachable queries by testing each query using the FL algorithm. For reachable queries, we cannot choose them randomly by sampling the TC, because TC computation suffers from high time and space complexity, we cannot get it within limited time and memory size for large graphs. To this problem, we randomly pick a node \(u\) in each iteration, then randomly select an out-neighbor \(v\) recursively until \(v\) has no out-neighbor. Then, we have a path \(p\) from \(u\). At last, we randomly select a node \(v \neq u\) from \(p\) to get a reachable query \(u \sim v\). This operation will be continued until we reach the required number of reachable queries.

We show the comparison of query time for FL-0 to FL-128 in Table [5] from which we have the following observations.

First, FL-16 and FL-32 usually need the least time on the first kind of datasets \(D_1\), including amaze, kegg, email, Lj, web, citeseerx, dbpedia, twitter and web-uk, where the reachability ratio is more than 99% even when \(k = 1\). For these datasets, although the index size becomes larger and the index construction time becomes longer than that of FL-0, we use the least cost to achieve significant improvements. For example, compared with FL-0, FL-16 and FL-32 use about 1.5 times index size and 1.2 times index construction time to achieve more than 1,000 times improvements on query time.

Second, FL-128 suffers from the largest index size (about 3 times bigger than FL-0) and longest index construction time (about 1.3 times longer than FL-0), but achieves the best query performance on the second kind of datasets \(D_2\), due to that on these datasets, the reachability ratio will become larger with the increase of \(k\). These datasets include human, anthra, agrocyc, ecoo, vchocyc and arxiv.

Third, for the third kind of datasets \(D_3\), including 10cit-Patent, 10citeseerx, 05cit-Patent, 05citeseerx and patent, the reachability ratio is very small or even approach zero, and almost does not change with the increase of \(k\). For these datasets, FL-0 works best and the use of partial 2-hop labels cannot bring us any positive results. For example, compared with FL-0 on 05cit-Patent, the index size of FL-128 is 2.6 times bigger than FL-0, and the index construction time and query time of FL-128 are 1.04 and 1.5 times longer than that of FL-0.

At last, we choose one dataset from each kind and show the trend of its query time w.r.t. \(k\) in Figure [7] from which we can give out the suggestions on how to use partial 2-hop labels: (1) For the first kind of datasets \(D_1\), we highly recommend using partial 2-hop labels with \(k = 16\) to process reachability queries, due to that we can speed up reachability queries answering significantly by affording only a little more index size and index construction time. (2) For the second kind of datasets \(D_2\), we also recommend using partial 2-hop labels, due to that we can speed up reachability queries answering by partial 2-hop labels. But for the value of \(k\), it depends on your concerns on how much you could and would like to afford for the increased index size and index construction time. In general, the larger the value of \(k\), the less the query time, but the more the index construction time and the bigger the index size. (3) For the third kind of datasets \(D_3\), we do not recommend using partial 2-hop labels to process reachability queries.

### TABLE 6: Comparison of the index size (MB).

| Dataset       | FL-0 | FL-16 | FL-32 | FL-64 | FL-128 |
|---------------|------|-------|-------|-------|--------|
| amaze         | 0.07 | 0.08  | 0.10  | 0.12  | 0.18   |
| human         | 0.74 | 0.89  | 1.04  | 1.33  | 1.92   |
| anthra        | 0.24 | 0.29  | 0.33  | 0.43  | 0.62   |
| agrocyc       | 0.24 | 0.29  | 0.34  | 0.43  | 0.63   |
| ecoo          | 0.24 | 0.29  | 0.34  | 0.43  | 0.62   |
| vchocyc       | 0.18 | 0.22  | 0.25  | 0.32  | 0.47   |
| kegg          | 0.07 | 0.08  | 0.10  | 0.12  | 0.18   |
| arxiv         | 0.11 | 0.14  | 0.16  | 0.20  | 0.30   |
| email         | 4.4  | 5.3   | 6.2   | 7.9   | 11.5   |
| Lj            | 18.5 | 22.2  | 25.9  | 33.3  | 48.2   |
| web           | 7.1  | 8.5   | 9.9   | 12.8  | 18.4   |
| 10cit-Patent   | 20.9 | 25.1  | 29.3  | 37.7  | 54.3   |
| 10citeseerx    | 14.7 | 17.6  | 20.6  | 26.5  | 38.2   |
| 05cit-Patent   | 31.9 | 38.3  | 44.6  | 57.4  | 82.9   |
| 05citeseerx    | 27.8 | 33.3  | 38.9  | 50.0  | 72.3   |
| citeseerx      | 124.7| 149.7 | 174.6 | 224.5 | 324.3  |
| dbpedia        | 64.2 | 77.0  | 89.9  | 115.5 | 166.9  |
| patent         | 72.0 | 86.4  | 100.8 | 129.6 | 187.2  |
| twitter        | 345.6| 414.8 | 483.9 | 622.1 | 898.6  |
| web-uk         | 434.0| 520.8 | 607.6 | 781.2 | 1128.4 |

### TABLE 7: Comparison of the index construction time (ms).

| Dataset       | FL-0 | FL-16 | FL-32 | FL-64 | FL-128 |
|---------------|------|-------|-------|-------|--------|
| amaze         | 1.03 | 1.10  | 1.35  | 1.41  | 1.49   |
| human         | 9.01 | 10.01 | 11.50 | 11.65 | 11.67  |
| anthra        | 2.89 | 2.99  | 3.67  | 3.72  | 3.85   |
| agrocyc       | 2.96 | 3.27  | 3.63  | 4.19  | 3.96   |
| ecoo          | 3.08 | 3.27  | 3.91  | 6.05  | 4.00   |
| vchocyc       | 2.22 | 2.57  | 3.16  | 3.47  | 3.65   |
| kegg          | 1.11 | 1.20  | 1.62  | 1.65  | 1.39   |
| arxiv         | 4.71 | 4.61  | 6.34  | 7.25  | 8.44   |
| email         | 81.3 | 77.2  | 91.0  | 86.5  | 87.1   |
| Lj            | 325.1| 327.3 | 383.9 | 376.3 | 387.8  |
| web           | 175.8| 177.9 | 203.8 | 202.7 | 207.6  |
| 10cit-Patent   | 801.3| 803.3 | 862.0 | 832.0 | 862.1  |
| 10citeseerx    | 376.3| 385.3 | 415.5 | 419.1 | 437.6  |
| 05cit-Patent   | 1517.8| 1495.2 | 1518.4 | 1566.9 | 1577.2 |
| 05citeseerx    | 475.6| 478.5 | 540.0 | 540.0 | 540.1  |
| dbpedia        | 463.9| 403.7 | 4500.5 | 4441.1 | 4546.0 |
| twitter        | 2264.3| 2271.0 | 2288.6 | 2260.2 | 2298.4 |
| patent         | 5022.3| 5152.1 | 5238.3 | 5400.1 | 5372.3 |
| web-uk         | 6287.6| 6446.3 | 7236.2 | 7233.9 | 7719.2 |
| web-uk         | 6889.7| 7874.7 | 9945.0 | 9991.6 | 10366.0 |
ratio for a given graph. Our second experimental results show that by combining partial 2-hop labels with an existing reachability algorithm, the query performance has different trends with the increase of the number of hop-nodes k. And based on the second experimental results, we finally give out our findings on whether we should use partial 2-hop labels for reachability queries processing. Specifically, (1) for datasets with large reachability ratio, partial 2-hop labels should be used with k = 16; (2) for datasets with small reachability ratio, we do not recommend using partial 2-hop labels; and (3) for the remaining datasets, partial 2-hop labels can be used, and users can determine k’s value themselves according to their requirements on index size, index construction time and query time.

8 Acknowledgments

This work was partly supported by the grants from the Natural Science Foundation of Shanghai (No. 20ZR1402700), and from the Natural Science Foundation of China (No.: 61472339, 61572421, 61272124). The authors would like to thank the anonymous referees for their insightful and valuable comments.

References

[1] A. Agrawal, A. Borgida, and H. V. Jagadish, “Efficient management of transitive relationships in large data and knowledge bases,” in SIGMOD, pp. 253–262, 1989.
[2] Y. Chen and Y. Chen, “An efficient algorithm for answering graph reachability queries,” in ICDE, pp. 893–902, 2008.
[3] J. Cheng, S. Huang, H. Wu, and A. W. Fu, “TT-label: a topological-folding labeling scheme for reachability querying in a large graph,” in SIGMOD, pp. 193–204, 2013.
[4] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, “Reachability and distance queries via 2-hop labels,” in ACM-SIAM, pp. 937–946, 2002.
[5] R. Jin, N. Ruan, S. Dey, and J. X. Yu, “SCARAB: scaling reachability computation on large graphs,” in SIGMOD, pp. 169–180, 2012.
[6] R. Jin, N. Ruan, Y. Xiang, and H. Wang, “Path-tree: An efficient reachability indexing scheme for large directed graphs,” ACM Trans. Database Syst., vol. 36, no. 1, p. 7, 2011.
[7] R. Jin and G. Wang, “Simple, fast, and scalable reachability oracle,” PVLDB, vol. 6, no. 14, pp. 1978–1989, 2013.
[8] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry, “‘3-hop: a high-compression indexing scheme for reachability query,” in SIGMOD, pp. 813–826, 2009.
[9] S. Seufert, A. Anand, S. J. Bedathur, and G. Weikum, “FERRARI: flexible and efficient reachability range assignment for graph indexing,” in ICDE, pp. 1009–1020, 2013.
[10] S. Trißl and U. Leser, “Fast and practical indexing and querying of very large graphs,” in SIGMOD, pp. 845–856, 2007.
[11] S. J. van Schaik and O. de Moor, “A memory efficient reachability data structure through bit vector compression,” in SIGMOD, pp. 913–924, 2011.
[12] R. Veloso, L. Cerf, W. M. Junior, and M. J. Zaki, “Reachability queries in very large graphs: A fast refined online search approach,” in EDBT, pp. 511–522, 2014.
[13] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu, “Dual labeling: Answering graph reachability queries in constant time,” in ICDE, p. 75, 2006.
[14] H. Wei, J. X. Yu, C. Lu, and R. Jin, “Reachability querying: An independent permutation labeling approach,” PVLDB, vol. 7, no. 12, pp. 1191–1202, 2014.
[15] H. Yıldırım, V. Chaoji, and M. J. Zaki, “GRAIL: scalable reachability index for large graphs,” PVLDB, vol. 3, no. 1, pp. 276–284, 2010.
[16] H. Yıldırım, V. Chaoji, and M. J. Zaki, “GRAIL: a scalable index for reachability queries in very large graphs,” VLDB J., vol. 21, no. 4, pp. 509–534, 2012.
[17] J. X. Yu and J. Cheng, “Graph reachability queries: A survey,” in Managing and Mining Graph Data, pp. 181–215, 2010.
[18] Z. Zhang, J. X. Yu, L. Qin, Q. Zhu, and X. Zhou, “I/O cost minimization: reachability queries processing over massive graphs,” in EDBT, pp. 468–479, 2012.
[19] A. D. Zhou, W. Lin, S. Wang, and X. Xiao, “Reachability queries on large dynamic graphs: a total order approach,” in SIGMOD, pp. 1323–1334, 2014.
[20] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida, “Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths,” in CIKM, pp. 1601–1606, 2013.
[21] J. Su, Q. Zhu, H. Wei, and J. X. Yu, “Reachability querying: Can it be even faster?”, IEEE Trans. Knowl. Data Eng., vol. 29, no. 3, pp. 683–697, 2017.
[22] J. Zhou, S. Zhou, J. X. Yu, H. Wei, and Z. Chen, “DAG reduction: Fast answering reachability queries,” in SIGMOD, pp. 375–390, 2017.
[23] J. Zhou, J. X. Yu, N. Li, H. Wei, Z. Chen, and X. Tang, “Accelerating reachability query processing based on DAG reduction,” VLDB J., vol. 27, no. 2, pp. 271–296, 2018.
[24] N. Sengupta, A. Bagchi, M. Ramanath, and S. Bedathur, “ARROW: approximating reachability using random walks over web-scale graphs,” in 35th IEEE International Conference on Data Engineering, ICDE 2019, Macau, China, April 8-11, 2019, pp. 470–481, 2019.
[25] S. Wadhwa, A. Prasad, S. Ranu, A. Bagchi, and S. Bedathur, “Efficiently answering regular simple path queries on large labeled networks,” in SIGMOD 2019, 2019.
[26] M. Du, A. Yang, J. Zhou, X. Tang, Z. Chen, and Y. Zuo, “HT: A novel labeling scheme for k-hop reachability queries on dags,” IEEE Access, vol. 7, pp. 172110–172122, 2019.
[27] X. Tang, Z. Chen, K. Li, and X. Liu, “ Efficient computation of the transitive closure size,” Clust. Comput., vol. 22, no. Supplement, pp. 6517–6527, 2019.
[28] R. E. Tarjan, “Depth-first search and linear graph algorithms,” SIAM J. Comput., vol. 1, no. 2, pp. 146–160, 1972.
[29] T. Akiba, Y. Iwata, and Y. Yoshida, “Fast exact shortest-path distance queries on large networks by pruned landmark labeling,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013, pp. 349–360, 2013.
[30] M. Cha, H. Haddadi, F. Benevenuto, and P. K. Gummadi, “Measuring user influence in twitter: The million follower fallacy,” in ICWSM, 2010.