IFCA: Index-Free Community-Aware Reachability Processing Over Large Dynamic Graphs

Anonymous Author(s)

ABSTRACT

Reachability is a fundamental graph operator. State-of-the-art index-based reachability processing frameworks can efficiently handle static graphs, but the recent advent of dynamic graph data poses new challenges. To address these challenges, we propose an index-free, community-aware (IFCA) reachability processing framework inspired by efficient Personalized PageRank approximation algorithms, which identifies community structures on-the-fly to accelerate query processing. On top of it, we devise a community contraction technique to bridge the gap between vertices in distinct communities, and a cost-based strategy selection procedure to efficiently handle the resulting reduced graph. We conduct experiments with realistic query workloads over large-scale real dynamic graphs, showing our approach's superior efficiency compared with index-based and index-free state-of-the-art methods.

CCS CONCEPTS

• **Information systems → Query optimization**.

KEYWORDS

reachability, dynamic graph, graph algorithms

ACM Reference Format:

Anonymous Author(s). 2018. IFCA: Index-Free Community-Aware Reachability Processing Over Large Dynamic Graphs. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The reachability query, which checks whether a path exists from a source vertex to a destination vertex on a directed graph, is a fundamental graph query operator [38]. It is integral to various applications, including but not limited to social networks, semantic web, and biological networks, and has thus been extensively studied for years.

Existing approaches. The majority of existing approaches precompute offline indexes to speed up online query processing. We call them *index-based* approaches. On static graphs, state-of-the-art index-based approaches can answer a query within a microsecond on graphs with millions of vertices and edges. Recently, an *index-free* framework [40] has been proposed, which answers reachability queries approximately on-the-fly without any index. Section 2 gives a more comprehensive survey.

Challenges. The recent advent of dynamic graph data poses new challenges to efficient reachability processing. Graphs such as e-commerce activity graphs, social networks, and web graphs are naturally highly dynamic. For example, up to 20,000 edges are updated per second at the sales peak in the Alibaba e-commerce graph [37]. On these graphs, efficient reachability processing is of critical importance. For example, reachability queries can help detect fraudulent activities in e-commerce graphs [37] and conduct access control on social networks [14]. These highly dynamic scenarios challenge reachability frameworks to handle frequent updates while offering near real-time query performance.

Under such circumstances, the existing index-based approaches are rendered increasingly ineffectual. For static indexes, significant overhead occurs reconstructing them as the graph evolves; for the indexes that support dynamic updates, the maintenance cost is still high when updates are frequent. Index-free approaches are advantageous in such scenarios since they are free of index reconstruction and maintenance costs. However, the existing index-free framework [40] cannot meet the established demand for result accuracy in some applications, such as fraud detection, where both false positives and false negatives are intolerable.

The most straightforward index-free approach to ensure accuracy is bidirectional breadth-first search (BiBFS). However, it is *structure-agnostic* in that it always constructs two spanning trees rooted at the source and destination vertices respectively, until they intersect (on positive queries) or cannot extend further (on negative queries). Contrarily, many classes of real-world graphs are rich in community structures [22], which are dense subgraphs sparsely connected with their peripheries. Positive queries with source and destination vertices in the same community are under-optimized by BiBFS: a query vertex pair in a community with n' vertices and m' edges is processed by BiBFS in $O(n' + m')$ time, where the edge access time is the bottleneck since $m' \gg n'$ in the community. We analyze this bottleneck in detail in Section 4.

Our approach. We propose an index-free, community-aware (IFCA) approach that leverages the community structures in real-world graphs to accelerate reachability queries on-the-fly. Due to the correlation between Personalized PageRank (PPR) and community structures [4], we adapt efficient PPR approximation algorithms to guide the search, which we call the *probability-guided search* strategy. For positive query vertex pairs within a community, which are typically reachable via multiple paths, such a strategy is more efficient since it can find a path without visiting the majority of edges. For positive query vertex pairs that are not in the same community, we propose a graph reduction technique, *community contraction*, that puts them into the same community in the reduced graph by periodically contracting the identified communities into super-vertices. Note that although stand-alone probability-guided

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

search is approximate, IFCA is an exact algorithm, since community contraction guarantees that all reachable vertices are visited before termination.

After community contraction, the reduced graph not only is smaller but also has fewer community structures, which becomes more favorable for BiBFS. Therefore, we design a cost model that evaluates the cost of either continuing the guided search or switching to BiBFS on the reduced graph. If the estimated cost of BiBFS is lower than that of the guided search, we switch to BiBFS. We call such a procedure *cost-based strategy selection*. By appropriately setting the parameters, our approach has lower asymptotic complexity than BiBFS on both positive and negative queries in the worst case.

Contributions. Our contributions are summarized as follows:

- We are the first to propose an index-free reachability processing framework based on PPR approximation that identifies community structures on-the-fly to accelerate query processing.
- We devise a community contraction technique to bridge the gap between vertices in distinct communities and a cost-based strategy selection procedure to handle the reduced graph with less discernible community structures adaptively.
- We theoretically guarantee that our approach has lower asymptotic complexity than BiBFS given appropriate parameters.
- We empirically validate our approach's effectiveness on large-scale, real dynamic graphs with realistic workloads.

Paper organization. In Section 2, we discuss the related work. In Section 3, we lay out the preliminaries and propose a baseline algorithm. An analysis of the baseline that exposes opportunities for improvement is conducted in Section 4, leading up to the algorithmic description of our approach in Section 5. In Section 6, we experimentally evaluate our proposed solution. We conclude the paper in Section 7.

2 RELATED WORK

Existing work in reachability can be divided into two categories based on whether they rely on indexes, which are synopses of partial or complete reachability information precomputed from the graph.

Index-based approaches. There is a plethora of research in index-based reachability processing on static graphs. Following the taxonomy first proposed in [48], the index-based approaches can be further categorized into Label-Only and Label+G. Label-Only approaches, including [1, 8, 10–13, 19, 20, 27–29, 45, 50], answer all queries accurately by only accessing the index; while Label+G approaches, including [9, 25, 32, 41, 43, 44, 46, 48, 51, 54, 56], may also traverse the graph. Though these approaches are efficient in answering reachability queries on static graphs, they are unsuited for handling dynamic graphs, since they can only reconstruct their indexes from scratch in the case of graph updates, which is expensive when updates are frequent.

More recently, some reachability indexes have been developed to support incremental maintenance [23, 26, 34, 39, 48, 52, 55]. The majority of them construct their indexes on the directed acyclic graph (DAG) resulting from condensing the strongly connected components (SCCs) of the original graph. DAGGER [52] proposes index maintenance procedures based on DAG maintenance when some SCCs merge or split due to edge insertions or deletions. TOL [55] and IP [48] achieve significant improvement in query efficiency compared with DAGGER, but their index maintenance algorithms are designed on the premise that SCCs never merge or split. DBL [34], on the other hand, constructs two lightweight, complementary indexes on the original graph without maintaining the DAG. However, it has the inherent drawback of not being able to handle edge deletions.

Index-free approaches. Index-free reachability processing has been amply studied theoretically [2, 15, 18, 42]. A recent index-free framework, ARROW [40], aims for practical performance on large-scale graphs. However, since it is based on random walks, it is by nature an approximate algorithm and thus inapplicable in real scenarios that demand accuracy. It also requires a large number of random walks to obtain high precision, which can be inefficient on large-scale real graphs.

Although community structures are prevalent in many real-world graphs, such as social, biological, and communication networks [22], none of the existing reachability frameworks optimizes query processing over community structures to our knowledge, regardless of whether they are index-based.

3 PRELIMINARY

Definition 1 formally defines the reachability problem over a directed graph. Note that our method belongs to the index-free category and can naturally handle dynamic graphs without maintaining any index. The notations frequently used in this paper are summed up in Table 1.

Definition 1 (Reachability)**.** Given a directed graph $G = (V, E)$ and a pair of vertices $s, t \in V$, the reachability query answers if there exists a directed path from s to t in G; if so, we say t is reachable from s, denoted as $s \longrightarrow t$.

Table 1: Frequently used notations.

Notation	Description
$G = (V, E)$	a directed graph
n	the total number of vertices
m	the total number of edges
$N_{out}(v), N_{in}(v)$	the set of vertex v's out- or in-neighbors
$d_{out}(v), d_{in}(v)$	the out- or in-degree of vertex v
$s \longrightarrow t$	t is reachable from s
$\mathbf{ppr}_u$	the PPR vector with respect to source vertex u
$\mathbf{ppr}_u(v)$	the PPR value of vertex v with respect to source vertex u
α	the teleportation constant
ϵ	the residue threshold
χ_u	the vector with all 0's except the u-th element, which is 1
$\mathbf{r}_u$	the residue vector from u
π_u°	the reserve vector from u

3.1 Personalized PageRank and Baseline

To make the paper self-contained, we briefly introduce the concept of Personalized PageRank (PPR) and PPR computation techniques.

PPR. First defined in [36], PPR has since been widely adopted as a measure of localized interest and relevance in graphs [17]. The PPR vector $\mathbf{ppr}_s$ concerning a given source vertex s is defined as the solution to the following equation:

$$\mathbf{ppr}_s = \alpha \cdot \chi_s + (1 - \alpha) \cdot \mathbf{ppr}_s \cdot M,$$

where α is a constant in $(0, 1)$ called the teleportation constant; χ_s is the row vector with length n of all 0's except for the s-th element, which is 1, n is the number of vertices in G; and M is the $n \times n$ matrix given by $M = D_{out}^{-1} A$, where D_{out} is the diagonal matrix of out-degrees ($D_{out}[i][j] = d_{out}(i)$ if $i = j$ and 0 otherwise), and A is the adjacency matrix ($A[i][j] = 1$ if the edge $(i, j) \in E$ and 0 otherwise). The t-th element in s's PPR vector, $\mathbf{ppr}_s(t)$, is vertex t's PPR value concerning s. It can alternatively be defined by random walks:

$$\mathbf{ppr}_s(t) = \Pr[\text{a random walk starting from } s \text{ of length } X \sim geometric(\alpha) \text{ stops at } t],$$

where the walk length X follows the geometric distribution with regard to α, i.e., $\Pr[X = k] = (1 - \alpha)^k \cdot \alpha$.

There are mainly three categories of PPR computation techniques. The first category is based on **Monte Carlo simulation** [16], i.e., starting a certain number of random walks from the source vertex and taking the frequency of these walks terminating at a vertex as its approximate PPR value. The second category is generally called **push-based techniques** [3, 4], in which each vertex is associated with a *reserve* value, which will eventually be its approximate PPR value, and a *residue* value, which is a byproduct of the algorithm. All the vertices' residue and reserve are initialized as 0 except for the source's residue, which is initialized as 1. In each iteration, if a vertex has large enough residue, part of its residue is pushed to its neighbors, while the rest accumulates into the vertex's reserve. The residue and reserve values are propagated iteratively until no vertex has large enough residue. The third category is **power iteration**, which gradually refines the estimate of the PPR vector $\mathbf{ppr}_s$ by iterative matrix operations, and provably has an equivalence connection with a push-based method, forward push [49].

As mentioned in Section 1, our reachability processing method is based on the following interesting property, which directly follows from the alternative definition above:

Property 1. Given a directed graph $G = (V, E)$ and a pair of vertices $s, t \in V$, $s \rightarrow t \Leftrightarrow \mathbf{ppr}_s(t) > 0$, where $\mathbf{ppr}_s(t)$ denotes the Personalized PageRank (PPR) value of vertex t with respect to s.

Thus a baseline solution to utilize PPR for reachability processing is to employ state-of-the-art PPR algorithms for computing $\mathbf{ppr}_s(t)$ and check whether it is non-zero. We adopt push-based techniques because they can identify high-PPR vertices more efficiently than Monte Carlo simulation [4, 47].

Baseline. According to Property 1, the push-based framework can be adapted as a baseline solution for reachability processing, as shown in Algorithm 1. Line 1 initializes the residue vector. Each iteration (Lines 2-8) arbitrarily selects a vertex whose residue is above the threshold and pushes its residue to its neighbors. There are two changes in the baseline compared with the original push-based technique. Firstly, the baseline immediately returns true as soon as it reaches the destination vertex (Lines 5-6). Secondly, since we are now concerned only with whether a vertex's PPR is greater than zero instead of its exact value, the reserve maintenance can be eliminated. Since push-based techniques always generate underestimates of PPR, this baseline is an approximate algorithm that may produce false negatives.

The classic push-based techniques, forward push [4] and backward push [3], are both fit for Algorithm 1 but differ in the following respects:

- *Neighbor weights:* a vertex's residue is distributed to its neighbors according to their weights, denoted as $f_{dist}(u, u_i)$ (Line 7). Forward push distributes a vertex's residue evenly to its out-neighbors, i.e., $f_{dist}(u, u_i) = d_{out}(u)$; while backward push distributes more of a vertex's residue to its out-neighbors with smaller in-degrees, i.e., $f_{dist}(u, u_i) = d_{in}(u_i)$.[1]
- *Threshold normalization:* both forward and backward push have a residue threshold ϵ (Lines 2-3). However, forward push has an additional normalization factor $f_{norm}(u) = d_{out}(u)$, while for backward push, $f_{norm}(u) = 1$.

Algorithm 1: Baseline

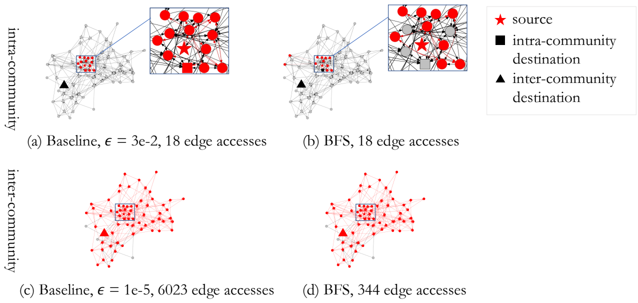
```
  Input: The source vertex $s$, the destination vertex $t$, the
         teleportation constant $\alpha$, the threshold $\epsilon$
  Output: Whether $t$ is reachable from $s$
1 $\mathbf{r}_s \leftarrow \chi_s$
2 while $\max_{u \in V} \frac{\mathbf{r}_s(u)}{f_{norm}(u)} \geq \epsilon$ do
3     Choose any $u \in V$ with $\frac{\mathbf{r}_s(u)}{f_{norm}(u)} \geq \epsilon$
4     forall $u_i \in N_{out}(u)$ do
5         if $u_i = t$ then
6             return true
7         $\mathbf{r}_s(u_i) \leftarrow \mathbf{r}_s(u_i) + (1 - \alpha) \cdot \frac{\mathbf{r}_s(u)}{f_{dist}(u, u_i)}$
8     $\mathbf{r}_s(u) \leftarrow 0$
9 return false
```

4 ANALYSIS OF BASELINE: STRENGTHS, WEAKNESSES & OPPORTUNITIES

In the previous section, we show the applicability of the baseline to answering reachability queries. However, it has great room for improvement in terms of query efficiency. We illustrate the improvement opportunities by the example below.

Motivating example. We use Highschool, a social network obtained from KONECT [30], as an example. Highschool is a small real graph with 70 vertices and 366 edges, representing the friendships between high school students. We run the baseline (with two different ϵ values) and the most straightforward index-free approach, BFS, and visualize the results in Figure 1. Each row shows an approach's frontier evolution; the visited vertices and edges are

[1] We essentially conduct a backward push on the reverse graph (i.e., the graph with all the edge directions reversed).



(a) Baseline, ϵ = 3e-2, 18 edge accesses (b) BFS, 18 edge accesses

(c) Baseline, ϵ = 1e-5, 6023 edge accesses (d) BFS, 344 edge accesses

Figure 1: The frontier expansion of BFS and Algorithm 1 at different ϵ's in relation to the number of edge accesses.

marked in red, and the unvisited in grey. The star-shaped vertex is the source, and the triangle and square vertices are the destinations of two distinct queries. The x-axis shows the number of *edge accesses*, which is the main factor influencing the query processing time of these methods. The blue box at the center of each graph, zoomed in on the right, encloses the set of vertices with the largest PPR concerning the source. The set of vertices with sufficiently large PPR concerning a source vertex can be defined as the community around it, since such a set provably has low conductance [4], which is the classic discerning metric of communities.

Intra-community reachable pairs. The baseline performs significantly better than BFS on intra-community reachable pairs, which are pairs of source and destination vertices in the same community, typically connected via many paths. The star-shaped and square vertices in Figure 1 form such a pair. In this case, the baseline reaches the destination with fewer edge accesses (18) than BFS (344) with both ϵ values, because it quickly finds one of the many paths from the star-shaped vertex to the square vertex without traversing the majority of edges. On the other hand, BFS is unaware of the community structure. At 18 edge accesses, BFS leaves four vertices in the community unvisited, including the destination, only returning to it much later. Such a performance gap can be more remarkable on large real graphs with denser communities.

Inter-community reachable pairs. The baseline performs worse than BFS on inter-community reachable pairs, which are pairs of source and destination vertices in different communities. The star-shaped and triangle vertices in Figure 1 form such a pair. In this case, the baseline with a larger ϵ quickly runs out of vertices that satisfy the condition in Line 2 (Algorithm 1) and terminates without advancing its frontier beyond the community, resulting in a false negative. The baseline with a smaller ϵ, on the other hand, can eventually visit the destination vertex, but with much more edge accesses than BFS. This is because a smaller ϵ allows residue to continuously propagate along cycles in the graph, causing already visited vertices and edges to be accessed repeatedly. Such a performance gap can be more remarkable on large real graphs with more communities.

Other limitations. In addition to its efficiency in handling inter-community reachable pairs, the baseline also has the following limitations:

- *Handling negative queries:* the baseline does not optimize negative queries, and there is no guarantee that its performance will not be worse than BFS asymptotically.
- *Approximate results:* the baseline may produce false negatives.
- *Handling graphs without discernible communities:* the baseline assumes that the graph has community structures, which is not true of all real graphs.

Solution. We devise two techniques on top of the baseline in order to address these problems.

- *Community contraction:* when the set of visited vertices form a community, we contract them into a super-vertex and reinitialize the search on the reduced graph. This technique accelerates the processing of *inter-community reachable pairs* by reducing them to intra-community pairs, and also solves the *approximate results* problem.
- *Cost-based strategy selection:* we design a cost model to assess the cost of continuing the probability-guided search and switching to BiBFS. If BiBFS is cheaper than the probability-guided search, we switch to it. This technique can accelerate the search on the sparsified reduced graph resulting from community contraction, and also helps *handle graphs without discernible communities* in general. Given appropriate parameters, this technique also helps provide a guarantee on the time complexity (Section 5.4), including when *handling negative queries.*

5 OUR SOLUTION: IFCA

5.1 Algorithm Overview

To address the challenges in different scenarios mentioned in Section 4, we design IFCA to consist of the following components:

- *Probability-guided search (Section 5.2, Algorithm 3)*, which is an optimized version of the baseline (Algorithm 1) that efficiently handles intra-community reachable pairs with relatively large ϵ;
- *Community contraction (Section 5.3, Algorithm 4)*, which efficiently handles inter-community reachable pairs by contracting the community discovered by probability-guided search into a super-vertex and reinitializing the search on the reduced graph, thus reducing inter-community pairs to intra-community pairs;
- *Cost-based strategy selection (Section 5.4, Algorithm 6)*, which efficiently handles the case where the original graph or the reduced graph resulting from community contraction has no discernible community structures by estimating the cost of probability-guided search and BiBFS, and switching to BiBFS (Algorithm 5) when it is cheaper.

Algorithm 2 is the main algorithm of IFCA. Across all algorithms, f and r denote the forward and reverse directions of search, which follow the original and reversed edge directions, respectively. Lines 1-5 initialize the data structures and statistics. The work is mainly done inside a while loop, which gradually shrinks the current residue threshold ϵ_{cur} (Line 15). In each iteration, probability-guided search (Algorithm 3) is invoked in the forward and reverse directions (Lines 9, 12), followed by Algorithm 4 (Section 5.3), which detects if the visited vertices form a community and contracts them into a super-vertex if so (Lines 11, 14). The loop terminates when the cost-based strategy selection procedure (Algorithm 6) estimates BiBFS to be cheaper (Lines 7-8) and switches to BiBFS (Algorithm 5) from the current frontiers (Lines 18-20).

IFCA terminates when the probability guided search finds a path from the source to the destination (Lines 10, 13), the super-vertices have 0-degree (Lines 16-17, to be explained in Section 5.3), or during BiBFS (Line 20). IFCA is an exact algorithm that can handle both positive and negative queries with 100% precision, and has lower asymptotic complexity than BiBFS given appropriate parameters. (Please refer to Section 5.5 for the correctness and complexity proofs.) It is a natural fit for handling dynamic graphs, since no index is precomputed and maintained. When the graph is updated, only the adjacency lists are modified accordingly, incurring no extra overhead nor affecting the algorithmic procedure.

Algorithm 2: IFCA

```
    Input: The source vertex $s$, the destination vertex $t$
    Output: Whether $t$ is reachable from $s$
1   $\mathbf{r}_s \leftarrow \chi_s$, $\mathbf{r}_t \leftarrow \chi_t$
2   $\mathbf{vis}_s \leftarrow \chi_s$, $\mathbf{vis}_t \leftarrow \chi_t$
3   $\mathbf{exp}_s \leftarrow \mathbf{0}$, $\mathbf{exp}_t \leftarrow \mathbf{0}$
4   $intEdges_s \leftarrow 0$, $intEdges_t \leftarrow 0$
5   $\epsilon_{cur} \leftarrow \epsilon_{init}$
6   while true do
        /* Cost-based strategy selection (Alg. 6) */
7       if CostBasedStrategySelection() then
8           break
        /* Forward probability-guided search (Alg. 3) */
9       if ProbabilityGuidedSearch($\epsilon_{cur}, f$) then
10          return true
        /* Try forward community contraction (Alg. 4) */
11      CommunityContraction($\epsilon_{cur}, f$)
        /* Reverse probability-guided search */
12      if ProbabilityGuidedSearch($\epsilon_{cur}, r$) then
13          return true
        /* Try reverse community contraction */
14      CommunityContraction($\epsilon_{cur}, r$)
15      $\epsilon_{cur} \leftarrow \epsilon_{cur}/step$
16      if $d_{out}(v^f_{super}) = 0$ and $d_{in}(v^r_{super}) = 0$ then
17          return false
    /* BiBFS takes over (Alg. 5) */
18  $frontier_f \leftarrow \{v_i | residue_f(v_i) > 0\}$
19  $frontier_r \leftarrow \{v_i | residue_r(v_i) > 0\}$
20  return BiBFS($frontier_f, frontier_r$)
```

5.2 Bidirectional Probability-Guided Search

Bidirectional search has been proven a simple yet effective strategy in reachability processing [6] since it prevents the frontier from expanding to enormous sizes by reducing the search depth in both direction approximately by half. We hence propose a bidirectional probability-guided search scheme based on the baseline (Algorithm 1).

The pseudocode of the probability-guided search from the forward direction is shown in Algorithm 3. For the reverse direction, we simply reverse all the edge directions. Since we adopt the bidirectional search scheme, we can return true as soon as a vertex has been visited from both directions (Line 9). In addition, $\mathbf{exp}_s$ (or $\mathbf{exp}_t$) keeps track of whether a vertex has been explored (i.e., whether its out-neighbors has been visited from it). When a vertex is explored for the first time, its out-degree is accumulated into $intEdges_s$ (or $intEdges_t$), which is the estimate of internal edges in the current community, to be used in the cost-based strategy selection process (Section 5.4).

Algorithm 3: Probability-Guided Search

```
    Input: The current residue threshold $\epsilon_{cur}$, the direction for
           performing push (assume $f$ w.l.o.g.)
    Output: Whether a path can be found from the source
            vertex $s$ to the destination vertex $t$ at $\epsilon_{cur}$
1   while $\max_{u \in V} \frac{\mathbf{r}_s(u)}{f_{norm}(u)} \geq \epsilon_{cur}$ do
2       Choose any $u \in V$ with $\frac{\mathbf{r}_s(u)}{f_{norm}(u)} \geq \epsilon_{cur}$
3       if $!\mathbf{exp}_s(u)$ then
4           $\mathbf{exp}_s(u) \leftarrow$ true
5           $intEdges_s \leftarrow intEdges_s + d_{out}(u)$
6       forall $u_i \in N_{out}(u)$ do
7           if $!\mathbf{vis}_s(u_i)$ then
8               if $\mathbf{vis}_t(u_i)$ then
9                   return true
10              $\mathbf{vis}_s(u_i) \leftarrow$ true
11          $\mathbf{r}_s(u_i) \leftarrow \mathbf{r}_s(u_i) + (1-\alpha) \cdot \frac{\mathbf{r}_s(u)}{f_{dist}(u,u_i)}$
12      $\mathbf{r}_s(u) \leftarrow 0$
13  return false
```

5.3 Community Contraction

As analyzed in Section 4, when processing inter-community reachable pairs, the probability-guided search may cause already visited vertices and edges to be accessed repeatedly within communities. To avoid this, after each iteration of the search (Algorithm 3), we contract communities into super-vertices.

The criterion for a local community is that it should be sparsely connected with its peripheries but densely connected within; that is, it should have far fewer external edges (i.e., edges between vertices inside the community and those outside it) than internal edges (i.e., edges between vertices inside the community). This is reflected by *conductance* [35], a metric widely used in community discovery. Conductance is defined over a set of vertices S as follows:

$$\Phi(S) = \frac{|\theta(S)|}{\min(\mathrm{vol}(S), 2m - \mathrm{vol}(S))}$$

where $\theta(S) = \{\langle u, v\rangle | u \in S, v \notin S\}$ is the set of external edges, m is the number of edges in G, and $\mathrm{vol}(S) = \sum_{v \in S}(d_{out}(v) + d_{in}(v))$ is an approximate number of internal edges. $(2m - vol(S))$ keeps the denominator below m, preventing the false classification of overly large vertex sets as communities. The lower the conductance, the more densely connected the community is.

Algorithm 4: Community Contraction

```
Input: The current residue threshold $\epsilon_{cur}$, the direction for
       performing contraction (assume $f$ w.l.o.g.)
Output: $\mathbf{res}_s$, $\mathbf{vis}_s$, $\mathbf{exp}_s$, and $intEdges_f$
1  if $\epsilon_{cur} \geq \epsilon_{pre}$ then
2      return
3  if $v^f_{super} \notin V$ then
4      $V \leftarrow V + \{v^f_{super}\}$
5  forall $v_i \in V$ s.t. $\mathbf{vis}_s(v_i)$ and $v_i \neq v^f_{super}$ do
6      if $!\mathbf{exp}_s(v_i)$ then
7          forall $u_i \in N_{out}(v_i)$ do
8              if $u_i \notin N_{out}(v^f_{super})$ then
9                  $N_{out}(v^f_{super}) \leftarrow N_{out}(v^f_{super}) + \{u_i\}$
10     $V \leftarrow V - \{v_i\}$
11 $\mathbf{res}_s(v^f_{super}) \leftarrow 1$
12 $\mathbf{vis}_s(v^f_{super}) \leftarrow$ true
13 $\mathbf{exp}_s(v^f_{super}) \leftarrow$ false
14 $intEdges_f \leftarrow 0$
15 $\epsilon_{cur} \leftarrow \epsilon_{init}$
```

It is best to perform contraction as soon as the set of visited vertices has sufficiently low conductance. However, accurately collating $|\theta(S)|$ requires at least $O(m)$ and is too expensive to conduct in each iteration. Instead, we exploit the relation between PPR and conductance; that is, a community around the source vertex is formed from the vertices with top-ranking PPR [4]. Therefore, we tune a parameter ϵ_{pre} (Section 6.1) and perform contraction as soon as the current residue threshold ϵ_{cur} reaches below it, since this indicates that the PPR of all the visited vertices are above $O(\epsilon_{pre})$, thus forming a superset of the top-ranking PPR vertices.

The contraction procedure is presented in Algorithm 4, assuming the forward direction without loss of generality. When contraction is performed for the first time in the current direction, a super-vertex (v^f_{super}) is added to the graph (Lines 3-4). The frontier vertices' neighbors are added to the adjacency list of the super-vertex with duplicates removed (Lines 6-9). All the visited vertices are then deleted (Line 10), resulting in a reduced graph. In order to reduce overhead, we perform *virtual updates* on all the affected vertices' adjacency lists after contraction: instead of removing the contracted vertices from the adjacency lists, they are mapped to the super-vertex. The residue of the super-vertex is reset to 1 since it is now the new source vertex in the reduced graph (Line 11); the threshold ϵ is also restored to the initial value. The super-vertex is visited but not explored since its neighbors have not been visited yet (Lines 12-13). The estimated number of internal edges is reinitialized as zero (Line 14), while that of external edges does not change since it is still equal to the number of the super-vertex's out-neighbors.

Note that if we keep performing contraction, a super-vertex will eventually be formed in the forward direction with zero out-degree (or in the reverse direction with zero in-degree), which is representative of all the vertices that are reachable from the source (or reachable to the destination). Therefore, community contraction also helps achieve full precision.

5.4 Cost-Based Strategy Selection

Each iteration of community contraction (Section 5.3) results in a reduced graph that is smaller and has fewer communities than the original. It is thus more and more favorable for BiBFS to take over the remaining search process. We thus design a cost model to help decide when to switch from bidirectional probability-guided search to BiBFS by assessing and comparing the costs of these two strategies.

5.4.1 BiBFS. Algorithm 5 shows the BiBFS algorithm we adopt, initiated from vertex frontiers instead of source and destination vertices. The forward and reverse input frontiers contain vertices with positive residue in the corresponding direction (Algorithm 2, Lines 18-19). Lines 3-12 take care of the forward direction, and lines 13-22 the reverse direction. The **vis** vectors are inherited from Algorithm 3. Traversals from the two directions are interleaved at the granularity of a *layer*; that is, the algorithm switches to the other direction when all the neighbors of the vertices on the current frontier have been visited.

5.4.2 The cost model. The key insight behind our cost model is that both the probability-guided search and BiBFS are composed of similar basic operations: that of accessing and executing some computation on one of the current vertex's neighbors. Therefore, when evaluating the costs of these strategies, we need to take two features into account: the number of operations to perform until termination (i.e., the asymptotic complexity), and the time it takes to perform each operation (i.e., the constant factor).

Algorithm 6 shows the cost model and decision procedure. The estimated cost of each strategy is its estimated number of basic operations multiplied by its estimated execution time (normalized by the ratio λ, to be introduced in Section 5.4.4) for each operation (Lines 1-2). The algorithm is invoked from the main algorithm at the start of of each iteration (Algorithm 2, Line 7). The constant coefficient 2 in Line 1 is due to the bidirectionality of the probability-guided search. k_f and k_r are set according to the analysis in Section 5.4.3. If the estimated cost of BiBFS is lower than that of the probability-guided search, Algorithm 6 will return **true**, causing the loop in Algorithm 2 to terminate and leading to BiBFS.

5.4.3 The number of operations. We estimate the number of operations required by the two strategies as follows.

1) Continuing the probability-guided search. We first estimate the number of operations if we continue the probability-guided search.

Lemma 1. Given a threshold ϵ and a teleportation constant α, Algorithm 3 conducts $O(\frac{1}{\alpha\epsilon})$ basic operations until termination using forward push, and conducts $O(\frac{d_{avg}}{\alpha\epsilon})$ basic operations until termination using backward push, where $d_{avg} = m/n$ is the average degree.[2]

[2] The proof is similar to those in [4] and [33]. We omit the details due to limited space.

Algorithm 5: BiBFS

```
Input: The forward frontier $frontier_f$, the reverse frontier $frontier_r$
Output: Whether $t$ is reachable from $s$
$next_f \leftarrow \phi$, $next_r \leftarrow \phi$
while $frontier_f \neq \phi \vee frontier_r \neq \phi$ do
    while $frontier_f \neq \phi$ do
        Choose any $u \in frontier_f$
        $frontier_f \leftarrow frontier_f - \{u\}$
        forall $u_i \in N_{out}(u)$ do
            if $!\mathbf{vis}_s(u_i)$ then
                if $\mathbf{vis}_t(u_i)$ then
                    return true
                $\mathbf{vis}_s(u_i) \leftarrow$ true
                $next_f \leftarrow next_f + \{u_i\}$
    $swap(frontier_f, next_f)$
    while $frontier_r \neq \phi$ do
        Choose any $u \in frontier_r$
        $frontier_r \leftarrow frontier_r - \{u\}$
        forall $u_i \in N_{in}(u)$ do
            if $!\mathbf{vis}_t(u_i)$ then
                if $\mathbf{vis}_s(u_i)$ then
                    return true
                $\mathbf{vis}_t(u_i) \leftarrow$ true
                $next_r \leftarrow next_r + \{u_i\}$
    $swap(frontier_r, next_r)$
return false
```

Algorithm 6: Cost-Based Strategy Selection

```
Output: Whether to switch from probability-guided search to BiBFS
$cost_{Push} \leftarrow \lambda[2(\frac{1}{\alpha\epsilon_{pre}} - \frac{1}{\alpha\epsilon_{cur}}) + \left(\frac{n_f}{k_f} + \frac{n_r}{k_r}\right)(\frac{1}{\alpha\epsilon_{pre}} - \frac{1}{\alpha\epsilon_{init}})]$
$cost_{BiBFS} \leftarrow (|V'_f| + m'_f - intEdges_f) + (|V'_r| + m'_r - intEdges_r)$
if $cost_{Push} > cost_{BiBFS}$ then
    return true
else
    return false
```

We can infer the number of basic operations of the probability-guided search between two contraction invocations from Lemma 1. The total number of basic operations before the next contraction is $O(\frac{1}{\alpha\epsilon_{pre}})$ $(O(\frac{d_{avg}}{\alpha\epsilon_{pre}})$ if we use backward push; similar in the following), and $O(\frac{1}{\alpha\epsilon_{cur}})$ operations have already been performed. Therefore, the number of basic operations up to the next contraction is $O(\frac{1}{\alpha\epsilon_{pre}} - \frac{1}{\alpha\epsilon_{cur}})$. In addition, the cost of continuing the search on the reduced graph resulting from contraction should also be taken into account. The number of basic operations between two adjacent contractions is always $O(\frac{1}{\alpha\epsilon_{pre}} - \frac{1}{\alpha\epsilon_{init}})$ since ϵ is restored to the initial value after contraction.

What remains to be estimated is how many times contraction will be performed in total, denoted as N. Let n_f and n_r denote the number of the remaining vertices; k_f and k_r denote the estimated number of vertices that are visited between two invocations of contraction in the forward and reverse search, respectively. The number of contractions can thus be estimated by $N = \frac{n_f}{k_f} + \frac{n_r}{k_r}$. During the probability-guided search, n_f and n_r can be obtained by subtracting the number of explored vertices from n. k_f and k_r, however, cannot be known exactly in advance. We can estimate them based on graph structure characteristics. For example, we can obtain upper and lower bounds on k_f and k_r by assuming that the graph is scale-free[3] and thus has a power-law PPR distribution [5]. Without loss of generality, we only discuss how to estimate k_f in the following.

Upper bound. When PPR satisfies the power-law distribution, the j-th largest PPR can be expressed as follows:

$$\mathbf{ppr}_s(u_j) = c \cdot j^{-\beta} \tag{1}$$

where c is the power-law coefficient, and β is the exponent that falls in the interval $(0, 1)$. According to Line 7 in Algorithm 1, all k_f visited vertices have a residue that is at least $(1-\alpha)\epsilon_{pre}$ at some point. Since an α portion of a vertex's residue is accumulated into its reserve when it pushes the residue to its neighbors, the reserves of the visited vertices are at least $\alpha(1-\alpha)\epsilon_{pre}$. As push-based algorithms always underestimate PPR, the PPR of all visited vertices is also at least $\alpha(1-\alpha)\epsilon_{pre}$, including the vertex with the smallest PPR among them, whose PPR is at most the k_f-th largest. (Though we no longer maintain the reserve, this relation still holds.) Hence we have an upper bound on k_f:

$$c \cdot (k_f)^{-\beta} \geq \alpha(1-\alpha)\epsilon_{pre} \Rightarrow k_f \leq \left(\frac{c}{\alpha(1-\alpha)\epsilon_{pre}}\right)^{\frac{1}{\beta}} \tag{2}$$

Lower bound. On the other hand, considering backward push, a lower bound on k_f is also obtainable. Backward push guarantees that the reserve of each vertex v satisfies the following property before contraction:

$$\mathbf{ppr}_s(v) - \pi_s^{\circ}(v) \leq \epsilon_{pre} \tag{3}$$

Therefore, all vertices with zero reserve have a PPR smaller than or equal to ϵ_{pre}, so the k_f vertices with PPR larger than ϵ_{pre} are exactly those with non-zero reserve, composing a subset of the k_f visited vertices. Suppose k'_f is the number of vertices with PPR larger than ϵ_{pre}, we have $k_f \geq k'_f$.

We also have the following property for the vertex with the $(k'_f + 1)$-th largest PPR , which leads to a lower bound on k_f:

$$c \cdot (k'_f + 1)^{-\beta} \leq \epsilon_{pre} \Rightarrow k_f \geq k'_f \geq (\frac{c}{\epsilon_{pre}})^{\frac{1}{\beta}} - 1 \tag{4}$$

[3] Scale-free graphs are prevalent in the real world. Many highly dynamic graphs, such as social networks and web graphs , are scale-free [24].

The upper and lower bounds above contain the constants c and β. β directly derives from the graph structure. Since PPR is a probability distribution, we have $\sum_{j=1}^{n_f} c \cdot j^{-\beta} = 1 \Rightarrow c = 1/(\sum_{j=1}^{n_f} j^{-\beta})$.

The number of contractions N can thus be estimated by substituting k_f and k_r by any value between their upper and lower bounds. The closer the chosen value is to the upper bound, the more the cost model favors continuing the probability-guided search, and vice versa. We approximate k_f and k_r by their upper bounds in the experiments (Section 6).

Remark. Though the above analysis assumes that the graph is scale-free, it is also extendable to more generalized assumptions, such as power-law-bounded degree distributions [7].

2) Switching to BiBFS. The number of operations in BiBFS is analyzed as follows.

Lemma 2. Given a directed graph $G = (V, E)$, Algorithm 5 conducts $O(|V'| + |E'|)$ basic operations until termination, where V' is the set of the vertices that are unvisited or on the input frontier, and $E' = \{\langle v_i, v_j \rangle | v_i \in V', v_j \in V', \langle v_i, v_j \rangle \in E\}$.

The proof is omitted due to its plainness. $|V'|$ is equal to n minus the number of explored vertices, which can be precisely collated. $|E'|$ is difficult to collate precisely since scanning all edges in each iteration is too expensive. Fortunately, we have maintained an estimate of the number of internal edges $intEdges$, which is approximated by the sum of the out-degrees (in-degrees for the reverse direction) of the vertices that are visited but not on the frontier. We can thus maintain a counter initialized as m, and subtract $intEdges$ from it each time contraction is performed; we denote this counter's value as m'. At decision time, we further subtract m' by the current $intEdges$ to estimate $|E'|$.

5.4.4 Execution time of the basic operations. We observe from the pseudocode of Algorithms 3 and 5 that a basic operation of probability-guided search needs more computation than BiBFS (i.e., updating $residue$ and maintaining $intEdges$ and $extEdges$). How this impacts their execution costs is difficult to model theoretically. Instead, we perform each type of basic operation under the same setting for the same number of times respectively, calculate their average running time, and divide the average running time of the probability-guided search by that of BiBFS to obtain the ratio λ.

5.5 Correctness and Complexity

Theorem 1 (Correctness)**.** Algorithm 2 returns **true** if and only if v is reachable from u.

Proof. (⇒) In all cases where the main algorithm returns **true**, v is reachable from u.

- Returns **true** during probability-guided search (Algorithm 3, Line 10). The return condition is that the currently probed vertex has been visited from both directions. If no contraction has been performed, the claim obviously holds. Otherwise, the super-vertex in the reverse direction is reachable from the forward direction. Since only the visited vertices are contracted, and all their neighbors are retained in the super-vertex's adjacency list, contraction preserves reachability, and the claim still holds.
- Returns **true** during graph traversal (Algorithm 5, Lines 9, 19). This indicates that some vertex on the reverse frontier is reachable from some other vertex on the forward frontier; thus the claim holds.

(⇐) Prove by contradiction. Suppose v is reachable from u, but the main algorithm returns **false**. The only possible cause is failing to find any path connecting the two frontiers, which either manifests as at least one of the two super-vertices having a degree of 0 (Algorithm 2, Line 16-17) or BiBFS failing to meet halfway (Algorithm 5, Line 23). Therefore, there must exist some path from u to v that does not contain any vertex on the frontiers. All its vertices must then be either explored or unvisited, so there must be at least one edge from an explored vertex to an unvisited vertex. This, however, is impossible: the neighbor of an explored vertex must be either explored or on the frontier.

□

Theorem 2. On scale-free graphs, IFCA's main algorithm runs in time $SubLinear(n + m)$ if $\epsilon_{pre} < c$.

Proof. According to the analysis in Section 5.4.3, the number of visited vertices between two contraction invocations is $k_f \geq (c/\epsilon_{pre})^{1/\beta} - 1 \approx (c/\epsilon_{pre})^{1/\beta}$, so without switching to BiBFS, the probability-guided search with community contraction runs in time $O(n/k_f \cdot d_{avg}/\epsilon_{pre}) = O(m \cdot (c/\epsilon_{pre})^{1-1/\beta})$.[4] When $\epsilon_{pre} < c$, we have $O(m \cdot (c/\epsilon_{pre})^{1-1/\beta}) = SubLinear(m)$. According to Lemma 2, BiBFS on the reduced graph runs in time $O(|V'| + |E'|)$, where $|V'| < n$ and $|E'| < m$. Therefore, the overall time complexity is $SubLinear(n + m)$. □

6 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the effectiveness of IFCA in processing reachability queries on dynamic graphs.

Environment. All of our experiments are conducted on a machine with an Intel Xeon 2.1GHz CPU and 128GB RAM, running CentOS Linux 7. Our algorithms are implemented in C++, and we adopt the C++ implementations of all the competitors kindly provided by their authors.[5]

Datasets. We select twelve real graphs for our experiments, the statistics of which are listed in Table 2. Enron and Wiki-talk are communication networks with edges representing emails and messages. Epinions, Digg-friends, Flickr-growth, Pokec and LiveJournal are social networks. SuperUser and StackOverflow are contact networks in which edges represent answers or comments posted by users. Wiki-growth, Dynamic-Dewiki, and Dynamic-FrWiki are web graphs where edges represent hyperlinks between web pages. Among them, Enron, Epinions, and Flickr-growth are datasets used in recent work [40]. SuperUser, Wiki-talk, Pokec and LiveJournal are obtained from SNAP [31], while the rest are obtained from KONECT [30].

All of these real graphs are *temporal*, i.e., each edge has a timestamp, except Pokec and LiveJournal, which are static snapshots of the social networks. We randomly assign unique timestamps

[4] Since $k_f \leq n$, ϵ_{pre} cannot be arbitrarily small.

[5] Our implementation is available at https://anonymous.4open.science/r/IFCA/.

Table 2: Real Datasets.

Dataset	$n = \|V\|$	$m = \|E\|$ (Initial)	# Edge insertions	# Edge deletions	# QpU	Negative queries (%)	Clustering coefficient
Enron	87,273	16,095	1,453,087	295,027	0.57	58.37	0.071648
Epinions	131,828	42,068	799,304	824,962	0.62	56.78	0.065679
SuperUser	567,316	46,244	2,321,980	832,384	0.32	45.04	0.010608
Digg-friends	279,630	86,582	3,375,054	1,548,274	0.20	67.99	0.061426
Wiki-talk	1,140,149	165,479	10,977,252	2,975,489	0.07	47.87	0.002204
Wiki-growth	1,870,709	1,997,657	37,955,488	27,235,531	0.02	14.06	0.003089
Flickr-growth	2,302,925	1,657,000	47,588,228	31,953,578	0.01	28.71	0.107648
StackOverflow	2,601,977	1,811,672	97,918,827	32,520,708	0.01	19.51	0.195353
Dynamic-DeWiki	2,162,457	3,441,403	55,029,876	27,364,893	0.01	8.61	0.007344
Dynamic-FrWiki	2,162,618	2,348,976	39,331,205	17,239,188	0.01	12.77	0.004945
Pokec	1,632,803	30,622,564	29,091,436	27,560,308	0.01	31.57	0.046821
LiveJournal	4,847,571	68,993,773	65,051,622	61,627,852	0.01	36.53	0.117916

to the edges in Pokec and LiveJournal. The edges with the minimum timestamp appear in the initial state, and all the rest are edge inserts. Dynamic-Dewiki and Dynamic-FrWiki have explicit edge deletions. For all the others, we suppose that each edge expires $\frac{T}{10}$ after its insertion, where T is the span between the minimum and maximum timestamps. We believe such a workload based on real temporal graphs is closer to actual application scenarios than randomly generated edge insertions and deletions on static graphs adopted by all previous works on dynamic graphs except [40].

We generate synthetic graphs using stochastic block models (SBMs) [21] since they model community structures. We use two-block SBMs, modeling graphs with two communities. The two blocks (i.e., communities) are of the same size varying from 10^5 to 10^7. We also vary the average vertex degree from 2.5 to 10 by adjusting the edge probabilities; the probability of an edge between vertices in the same community is configured to be ten times that of edges between different communities. Since synthetic graphs are for studying the scalability of our query algorithm, we view them as snapshots of dynamic graphs without generating edge insertions or deletions.

Queries. On real graphs, we split the span between the minimum and maximum timestamps evenly into intervals, corresponding to batches of updates. After each batch of updates, a batch of queries is generated by choosing the source and destination vertices independently and uniformly at random. The total number of queries on each dataset is a million, and the number of intervals is twenty, resulting in fifty thousand queries per batch. The average number of queries per update is shown in the # QpU column of Table 2. We generate a batch of fifty thousand queries in the same way for each synthetic graph.

6.1 Parameter Study

There are four parameters in our main algorithm (Algorithm 2): the termination residue threshold, ϵ_{pre}; the initial residue threshold, ϵ_{init}; the teleportation constant, α; and the residue threshold decreasing step, `step`. We investigate the effect of these parameters on `IFCA`'s performance in the following.

6.1.1 The termination residue threshold ϵ_{pre}. The average query time with varying ϵ_{pre} is shown in Figure 2. α, ϵ_{init} and `step` are fixed as the default in Section 6.1.4. The time complexity of our algorithm (Theorem **??**) seems to suggest that ϵ_{pre} should be as small as possible to achieve the least query time, but Figure 2 shows that the average query time first decreases and then increases as ϵ_{pre} grows. Such a trend shows that Theorem **??** does not give a tight upper bound since $O(1/\epsilon_{pre})$ is not a tight upper bound on the time complexity of forward push (as is $O(d_{avg}/\epsilon_{pre})$ for backward push), especially with a larger ϵ_{pre}. To validate this claim, we sample 1,000 vertices from each real graph, conduct forward push from them varying $1/\epsilon_{pre}$, and record the average push time. We show the results in Figure 3, where there is an observable turning point (marked with a vertical dashed line) on each graph, after which the average push time grows linearly with $1/\epsilon_{pre}$ (i.e., $O(1/\epsilon_{pre})$ is tight), but before which it grows sub-linearly (i.e., $O(1/\epsilon_{pre})$ is not tight). Intuitively, the turning point marks that the community frontier is exactly discovered. Therefore, choosing an ϵ_{pre} close to that at the turning point may lead to our algorithm running faster.

6.1.2 The teleportation constant α. The average query time with varying α is shown in Figure 4. ϵ_{pre} is fixed as the best found in Section 6.1.1; ϵ_{init} and `step` are fixed as the default in Section 6.1.4. When $\alpha > 0.5$, the average query time grows sharply with α on all graphs except SuperUser, Wiki-talk and StackOverflow, where the average query time fluctuates in a small range without observable trends. $\alpha > 0.5$ means the random walks underlying PPR are more likely to halt than proceed at each step, which is disadvantageous for reachability intuitively. It is thus advisable to choose $\alpha \leq 0.5$.

6.1.3 The initial residue threshold ϵ_{init} and the residue threshold decreasing step `step`*.* ϵ_{init} and `step` do not impact the time complexity of `IFCA` (Section 5.5). Intuitively, with ϵ_{pre} fixed, ϵ_{init} and `step` jointly determine the granularity of the search, i.e., how much the frontiers advance in each iteration. With ϵ_{pre} and α fixed as the best found in Sections 6.1.1 and 6.1.2, we vary ϵ_{init} from ϵ_{pre} to $10^3\epsilon_{pre}$ and `step` from 10 to 10^3, and find that their impact on the average query time is negligible. We omit the figure due to limited space.

6.1.4 Default parameters. Though all four parameters impact performance to some extent, `IFCA`'s advantage over existing approaches is not reliant on choosing the best parameters. To prove this, we select `IFCA`'s parameters heuristically in the subsequent experiments as follows: $\epsilon_{pre} = 10^4/m$ following the intuition that ϵ_{pre} should be smaller on larger and denser graphs, $\alpha = 0.1$ following recent work in local community detection [53], $\epsilon_{init} = 100\epsilon_{pre}$, and `step` $= 10$.

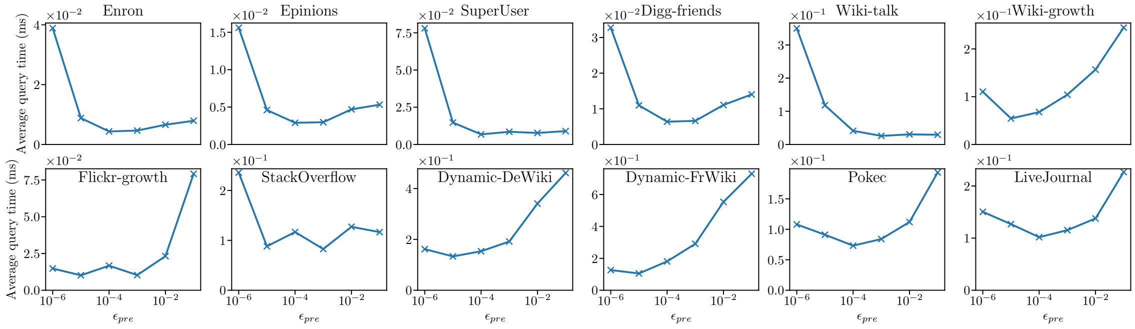


Figure 2: Average query time varying ϵ_{pre}.

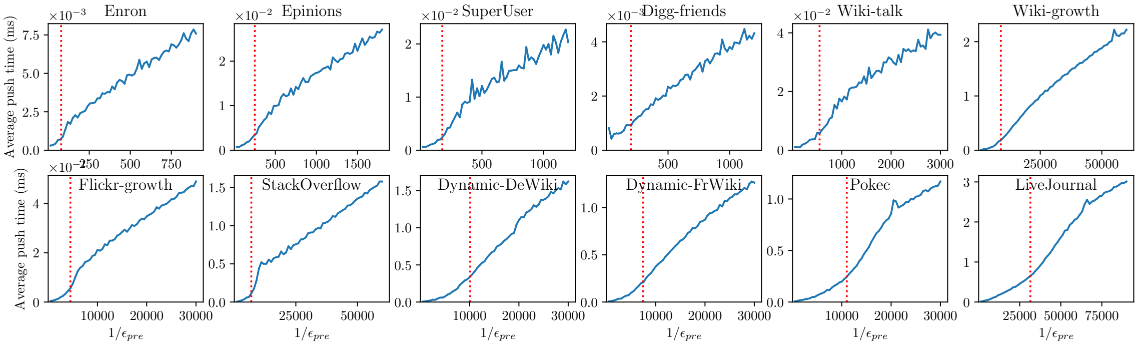


Figure 3: Average push time varying $1/\epsilon_{pre}$ (explaining Figure 2).

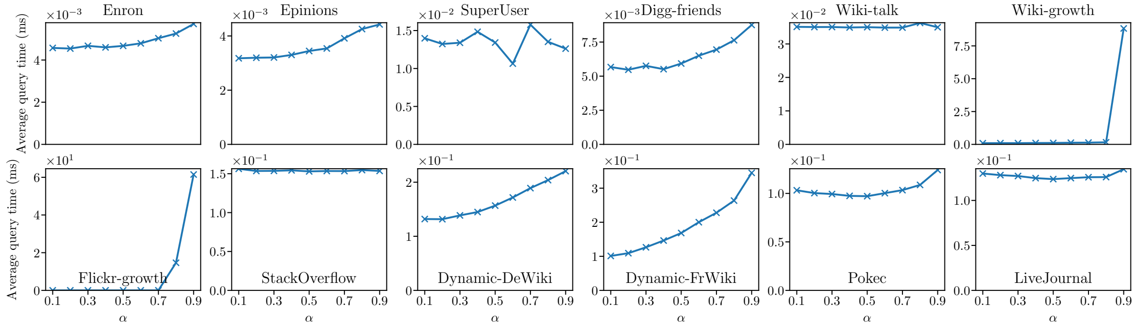


Figure 4: Average query time varying α.

6.2 Effectiveness of Optimizations

In this subsection, we empirically verify the effectiveness of our optimizations, community contraction and cost-based strategy selection. We compare the performance of the baseline (`Base`), the baseline with contraction (`Contract`), the full method (`IFCA`), and BiBFS. Figure 5 shows the average query time of these approaches on all the real datasets. Since the baseline is an approximate algorithm, we iteratively lower ϵ until the precision is at least 90% and equal to 100%, denoted as `Base@90%` and `Base@100%`, respectively. The following conclusions regarding the effectiveness of our optimizations can be drawn from Figure 5:

- `Base` is a competitive approximate algorithm. It is faster than BiBFS by up to two orders of magnitude on all real graphs except LiveJournal at 90% accuracy. However, it is unsuitable for accurate querying, being orders of magnitude slower than BiBFS at 100% accuracy.
- `Contract` guarantees 100% accuracy and is consistently faster than `Base@100%`, verifying the effectiveness of community

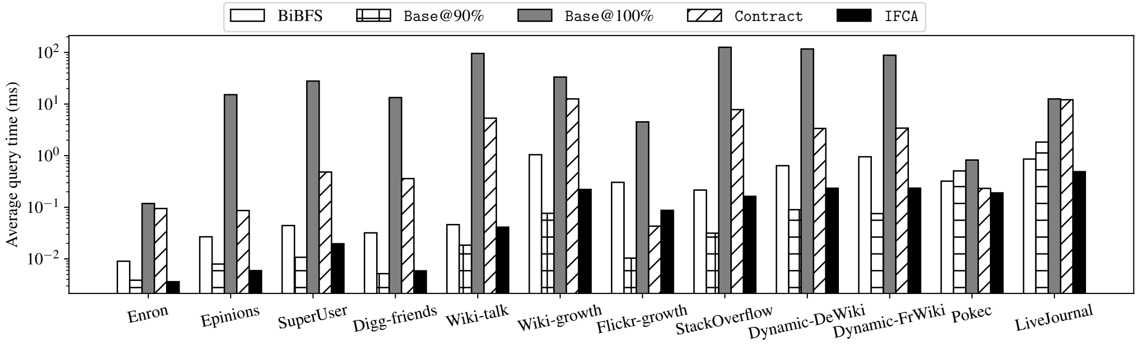


Figure 5: Effectiveness of optimizations.

contraction. However, it is still slower than BiBFS on most datasets.

- IFCA is consistently faster than Contract and BiBFS (except on Flickr-growth where it is slightly slower than Contract), verifying the effectiveness of cost-based strategy selection.

The degree to which IFCA is faster than Contract and BiBFS varies on different graphs, depending on how discernible the community structures are. The prominence of community structures on a graph is reflected by its clustering coefficient, shown in Table 2.

Graphs with indiscernible community structures. Wiki-talk has the lowest clustering coefficient among all the real graphs, which means its community structures are relatively indiscernible. According to our analysis in Section 4, graphs like this are favorable for BiBFS, corroborating the experimental results, where IFCA's advantage over BiBFS is the least significant.

Graphs with discernible community structures. Flickr-growth has the third highest clustering coefficient among all the real graphs, which means its community structures are relatively discernible. According to our analysis in Section 4, graphs like this are favorable for probability-guided search, corroborating the experimental results, where IFCA's advantage over BiBFS is significant. On Flickr-growth, Contract is also slightly faster than IFCA, which means the cost model overestimates the cost of probability-guided search and switches to BiBFS earlier than it should.

Cost Model's effectiveness. To verify our cost model's ability in choosing a near-optimal switching point, we design the following experiments. We suppose there exists an *oracle* that always selects the switching point that leads to the shortest processing time for each query, implemented by trying every possible switching point of each query and averaging the shortest query time. We test how close the performance of IFCA is to the oracle and show the results in Table 3. IFCA employs the cost-based strategy selection scheme, while Contract and BiBFS represent two extremes: never switching to BiBFS, and switching to BiBFS at the beginning.

The average query time of the oracle is shorter than those of Contract and BiBFS by up to two orders of magnitude, while IFCA performs comparably to the oracle, its average query time not exceeding 8 times that of the oracle on all the real graphs. Note that on Wiki-talk, the average query time of BiBFS is almost as close to the oracle as IFCA; in contrast, on Flickr-growth, the average query time of Contract is closer to the oracle than IFCA. As mentioned above, Wiki-talk and Flickr-growth have the lowest and one of the highest clustering coefficients, and thus the least and nearly the most discernible community structures, respectively. Although IFCA misses the optimal switching point on Flickr-growth, its performance is still comparable to the oracle.

Table 3: Performance of the cost model. (Time in *ms*)

	Oracle	IFCA	Contract	BiBFS
Enron	0.00285	**0.00419**	0.0947	0.00901
Epinions	0.00339	**0.00678**	0.0865	0.0268
SuperUser	0.0126	**0.0248**	0.482	0.0442
Digg-friends	0.00335	**0.00806**	0.359	0.0320
Wiki-talk	0.0261	0.0648	5.33	**0.0461**
Wiki-growth	0.0460	**0.222**	12.6	1.04
Flickr-growth	0.0136	**0.0412**	0.0431	0.304
StackOverflow	0.0258	**0.163**	7.79	0.215
Dynamic-DeWiki	0.100	**0.234**	3.37	0.640
Dynamic-FrWiki	0.0454	**0.235**	3.42	0.952
Pokec	0.0683	**0.190**	0.233	0.321
LiveJournal	0.0850	**0.490**	12.6	0.860

6.3 Comparison With State of the Art

In this subsection, we compare the performance of IFCA on reachability queries onreal dynamic graphs with state-of-the-art approaches, including TOL [55], IP [48] and DAGGER [52], the state-of-the-art index-based reachability processing frameworks on dynamic graphs; ARROW [40], the state-of-the-art index-free framework; and BiBFS. Since TOL and IP are designed for directed acyclic graphs (DAGs), we enable them to work on general directed graphs by maintaining a reachability-preserving DAG with the method proposed in DAGGER [52]. Since ARROW is an approximate algorithm, we tune the number of random walks it invokes and report the shortest query time that can yield over 95% accuracy. DBL is excluded from our comparison because it cannot handle edge deletions.

We plot each method's average query and update time on each real graph in a stacked bar chart (Figure 6), where the average

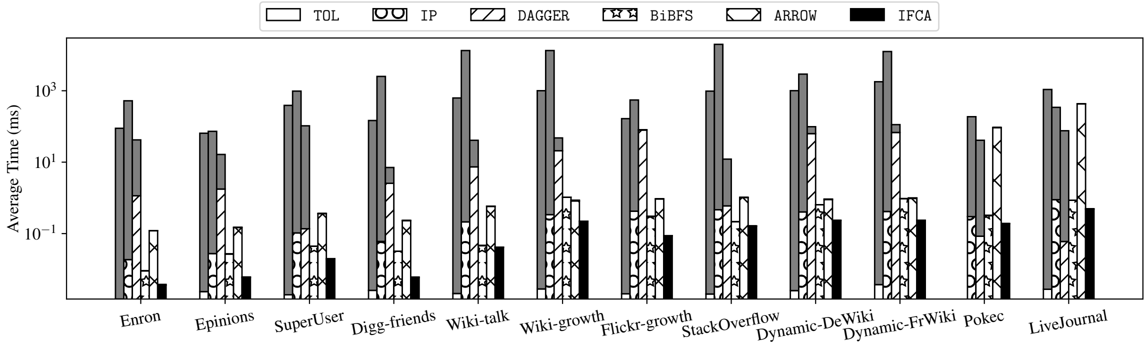


Figure 6: Average query and update time on real graphs. (Update time shown in grey)

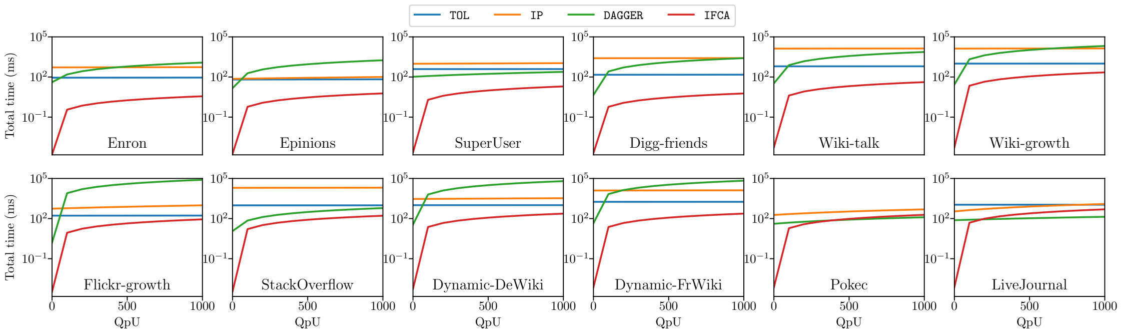


Figure 7: Total time varying QpU on real graphs.

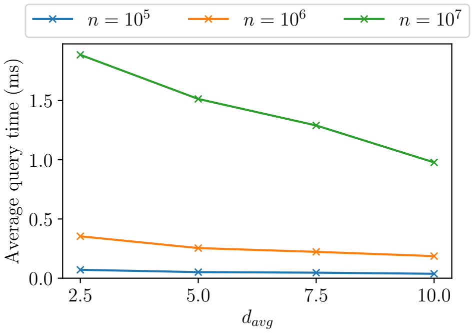


Figure 8: Average query time on synthetic graphs.

update time is stacked upon the average query time and uniformly shown in grey. Notably, for the index-based methods TOL and IP, updates take up a significant portion of time and can be up to five orders of magnitude slower than queries. Although TOL and IP process queries up to two orders of magnitude faster on average than BiBFS, and TOL can be up to two orders of magnitude faster than IFCA (IP is slower than IFCA on all real datasets), their advantage is lost compared with index-free methods when taking the update time into account. The main reason why handling updates is so expensive for these two methods is that when some SCCs of the graph split or merge as a result of an edge deletion or insertion, these methods do not provide an efficient solution since their update algorithms operate based on the assumption that SCCs remain unchanged. Aside from the SCC maintenance cost of the procedure borrowed from DAGGER, these methods also have to reconstruct at least part of their indexes to handle such events. On the other hand, DAGGER, which has the most lightweight index among the three, handles updates observably faster at the cost of query performance, with an average query time that is not significantly better than BBFS on most datasets.

On the other hand, the update time of the index-free methods, including IFCA, ARROW and BiBFS, is almost negligible since all these methods only need to update the graph. Overall, ARROW performs comparably with BBFS without guaranteeing 100% precision. IFCA consistently beats all the other methods except TOL on average query time on all the real datasets and BBFS on Wiki-talk, where the margin is negligible (explained in Section 6.2); it also beats TOL when the average update time is taken into account.

We further show the total time it takes for the index-based approaches (TOL, IP and DAGGER) and IFCA to perform an update and a certain number of queries varying the query-per-update ratio (QpU) in Figure 7. The total time of TOL and IP is almost stable, since the update time dominates the query time. The total time of DAGGER grows slightly with QpU, since its update time and query time are relatively balanced. The total time of IFCA grows at the sharpest rate among the compared approaches, since the query time dominates the update time. Such a phenomenon is consistent with the common sense that the larger QpU is, the more favorable index-based approaches become. However, the index-based approaches only outperform IFCA after QpU exceeds 1000 on all the datasets except Pokec and LiveJournal, on which the index-based approaches only outperform IFCA when QpU is at least 100. Therefore, IFCA is still widely applicable to query-intensive scenarios. (Note that on highly dynamic graphs, such as the Alibaba e-commerce graph [37] which has up to 20,000 updates per second, and Dynamic-FrWiki which has up to 9,180 updates per second, it is unlikely for QpU to be very large.)

6.4 Scalability Study

To evaluate the scalability of IFCA, we generate synthetic graphs with SBM, varying the number of vertices and the average degree. Note that the setting complements those in Sections 6.2 and 6.3 since the synthetic graphs are denser than the real graphs. The average query time of IFCA on synthetic graphs is shown in Figure 8. IFCA can answer reachability queries within two microseconds on dense graphs with up to a billion edges. Interestingly, IFCA runs slower on synthetic graphs with a larger number of vertices, but slightly faster on synthetic graphs with the same number of vertices but a larger average degree. The latter phenomenon can be attributed to the following factors:

- The ratio of negative queries, which is lower on denser graphs (around 5% when $d_{avg} = 2.5$, but 0% when $d_{avg} > 2.5$). IFCA runs slower on negative queries than positive queries since it cannot terminate early with a positive result. Answering negative queries on dense graphs can be particularly slow, since all the reachable vertices from the source and destination need to be visited. The drop in average query time from $d_{avg} = 2.5$ to 5 is particularly noticeable for this reason.
- The average distance between positive query vertex pairs, which is significantly longer on sparser graphs (e.g., 14.2 with $d_{avg} = 5$ and 6.3 with $d_{avg} = 20$ when $n = 10^6$). The longer the distance is, the less benefit is gained from the probability-guided search phase.

7 CONCLUSIONS

In this work, we propose IFCA, an index-free approach for reachability processing that adapts to large-scale real dynamic graphs. We adopt a bidirectional probability-guided graph search scheme inspired by Personalized PageRank approximation techniques, and devise a community contraction technique to leverage community structures prevalent in real graphs for accelerating query processing. Furthermore, to handle the reduced graph resulting from community contraction more efficiently, we design a cost-based strategy selection procedure that estimates the cost of continuing the guided push and switching to BiBFS and chooses the cheaper strategy accordingly. Experimental studies show that our approach is significantly more efficient than the index-free state-of-the-art and competitive with the index-based state-of-the-art on large-scale real dynamic graphs. In the future, we plan to explore adapting our approach for various forms of constrained reachability queries.

REFERENCES

[1] R. Agrawal, A. Borgida, and H. V. Jagadish. 1989. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data - SIGMOD '89*. ACM Press, Portland, Oregon, United States, 253–262. https://doi.org/10.1145/67544.66950

[2] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. 2012. Algorithms on evolving graphs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. 149–160.

[3] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. 2007. Local computation of pagerank contributions. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 150–165.

[4] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local Graph Partitioning Using PageRank Vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, Berkeley, CA, USA, 475–486. https://doi.org/10.1109/FOCS.2006.44

[5] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *arXiv:1006.2880 [cs]* (Aug. 2010). arXiv:1006.2880 [cs]

[6] Scott Beamer, Krste Asanovic, and David Patterson. 2012. Direction-optimizing breadth-first search. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–10.

[7] Paweł Brach, Marek Cygan, Jakub Łącki, and Piotr Sankowski. 2016. Algorithmic complexity of power law networks. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1306–1325.

[8] Jing Cai and Chung Keung Poon. 2010. Path-Hop: Efficiently Indexing Large Graphs for Reachability Queries. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management - CIKM '10*. ACM Press, Toronto, ON, Canada, 119. 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. (2005), 12.

[10] Yangjun Chen and Yibin Chen. 2008. An Efficient Algorithm for Answering Graph Reachability Queries. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, Cancun, Mexico, 893–902. https://doi.org/10.1109/ICDE.2008.4497498

[11] 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 *Proceedings of the 2013 International Conference on Management of Data - SIGMOD '13*. ACM Press, New York, New York, USA, 193. https://doi.org/10.1145/2463676.2465286

[12] Jiefeng Cheng, Jeffrey Xu Yu, Xuemin Lin, Haixun Wang, and Philip S. Yu. 2008. Fast Computing Reachability Labelings for Large Graphs with High Compression Rate. In *Proceedings of the 11th International Conference on Extending Database Technology Advances in Database Technology - EDBT '08*.

[13] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.* (2003).

[14] Imen Ben Dhia. 2012. Access control in social networks: a reachability-based approach. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. 227–232.

[15] Uriel Feige. 1996. A fast randomized LOGSPACE algorithm for graph connectivity. *Theoretical Computer Science* 169, 2 (1996), 147–160.

[16] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics* 2, 3 (2005), 333–358.

[17] David F Gleich. 2015. PageRank beyond the Web. *SIAM Rev.* (2015).

[18] Igor Gorodezky and Igor Pak. 2014. Generalized loop-erased random walks and approximate reachability. *Random Structures & Algorithms* 44, 2 (2014), 201–223.

[19] Haixun Wang, Hao He, Jun Yang, P.S. Yu, and J.X. Yu. 2006. Dual Labeling: Answering Graph Reachability Queries in Constant Time. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, Atlanta, GA, USA, 75–75. https://doi.org/10.1109/ICDE.2006.53

[20] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. 2005. Compact Reachability Labeling for Graph-Structured Data. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management - CIKM '05*. ACM Press, Bremen, Germany, 594. https://doi.org/10.1145/1099554.1099708

[21] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. 1983. Stochastic blockmodels: First steps. *Social networks* 5, 2 (1983), 109–137.

[22] Xin Huang, Laks V.S. Lakshmanan, and Jianliang Xu. 2017. Community Search over Big Graphs: Models, Algorithms, and Opportunities. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 1451–1454. https://doi.org/10.1109/ICDE.2017.211

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

[24] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. 2014. Hop Doubling Label Indexing for Point-to-Point Distance Querying on Scale-Free Networks. *Proceedings of the VLDB Endowment* 7, 12 (2014).

[25] Ruoming Jin, Ning Ruan, Saikat Dey, and Jeffrey Yu Xu. 2012. SCARAB: Scaling Reachability Computation on Large Graphs. In *Proceedings of the 2012 International Conference on Management of Data - SIGMOD '12*. ACM Press, Scottsdale, Arizona, USA, 169. https://doi.org/10.1145/2213836.2213856

[26] Ruoming Jin, Ning Ruan, Yang Xiang, and Haixun Wang. 2011. Path-Tree: An Efficient Reachability Indexing Scheme for Large Directed Graphs. *ACM Transactions on Database Systems* 36, 1 (March 2011), 1–44. https://doi.org/10.1145/1929934.1929941

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

[28] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 2009. 3-HOP: A High-Compression Indexing Scheme for Reachability Query. In *Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD '09*. ACM Press, Providence, Rhode Island, USA, 813. https://doi.org/10.1145/1559845.1559930

[29] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. 2008. Efficiently Answering Reachability Queries on Very Large Directed Graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data - SIGMOD '08*. ACM Press, Vancouver, Canada, 595. https://doi.org/10.1145/1376616.1376677

[30] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*. 1343–1350.

[31] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection.

[32] Lei Li, Wen Hua, and Xiaofang Zhou. 2017. HD-GDD: High Dimensional Graph Dominance Drawing Approach for Reachability Query. *World Wide Web* 20, 4 (July 2017), 677–696. https://doi.org/10.1007/s11280-016-0407-z

[33] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized pagerank estimation and search: A bidirectional approach. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 163–172.

[34] Qiuyi Lyu, Yuchen Li, Bingsheng He, and Bin Gong. 2021. DBL: Efficient Reachability Queries on Dynamic Graphs (Complete Version). *arXiv:2101.09441 [cs]* (Jan. 2021). arXiv:2101.09441 [cs]

[35] M. Mihail. 1989. Conductance and convergence of Markov chains-a combinatorial treatment of expanders. In *30th Annual Symposium on Foundations of Computer Science*. 526–531. https://doi.org/10.1109/SFCS.1989.63529

[36] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report.

[37] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-time constrained cycle detection in large dynamic graphs. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1876–1888.

[38] 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 Endow.* 11, 4 (dec 2017), 420–431. https://doi.org/10.1145/3186728.3164139

[39] R. Schenkel, A. Theobald, and G. Weikum. 2005. Efficient Creation and Incremental Maintenance of the HOPI Index for Complex XML Document Collections. In *21st International Conference on Data Engineering (ICDE'05)*. IEEE, Tokyo, Japan, 360–371. https://doi.org/10.1109/ICDE.2005.57

[40] Neha Sengupta, Amitabha Bagchi, Maya Ramanath, and Srikanta Bedathur. 2019. ARROW: Approximating Reachability Using Random Walks Over Web-Scale Graphs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, Macao, Macao, 470–481. https://doi.org/10.1109/ICDE.2019.00049

[41] S. Seufert, A. Anand, S. Bedathur, and G. Weikum. 2013. FERRARI: Flexible and Efficient Reachability Range Assignment for Graph Indexing. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, Brisbane, QLD, 1009–1020. https://doi.org/10.1109/ICDE.2013.6544893

[42] Michele Starnini, Andrea Baronchelli, Alain Barrat, and Romualdo Pastor-Satorras. 2012. Random walks on temporal networks. *Physical Review E* 85, 5 (2012), 056115.

[43] Jiao Su, Qing Zhu, Hao Wei, and Jeffrey Xu Yu. 2017. Reachability Querying: Can It Be Even Faster? *IEEE Transactions on Knowledge and Data Engineering* 29, 3 (March 2017), 683–697. https://doi.org/10.1109/TKDE.2016.2631160

[44] Silke Trißl and Ulf Leser. 2007. Fast and Practical Indexing and Querying of Very Large Graphs. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data - SIGMOD '07*. ACM Press, Beijing, China, 845. https://doi.org/10.1145/1247480.1247573

[45] Sebastiaan J. van Schaik and Oege de Moor. 2011. A Memory Efficient Reachability Data Structure through Bit Vector Compression. In *Proceedings of the 2011 International Conference on Management of Data - SIGMOD '11*. ACM Press, Athens, Greece, 913. https://doi.org/10.1145/1989323.1989419

[46] Renê R. Veloso, Loïc Cerf, Wagner Meira Junior, and Mohammed J. Zaki. 2014. Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach. https://doi.org/10.5441/002/EDBT.2014.46

[47] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax NS Canada, 505–514. https://doi.org/

10.1145/3097983.3098072

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

[49] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the global and local approaches: an efficient power iteration with forward push. In *Proceedings of the 2021 International Conference on Management of Data*. 1996–2008.

[50] 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 *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management - CIKM '13*. ACM Press, San Francisco, California, USA, 1601–1606. https://doi.org/10.1145/2505515.2505724

[51] Hilmi Yıldırım, Vineet Chaoji, and Mohammed J. Zaki. 2012. GRAIL: A Scalable Index for Reachability Queries in Very Large Graphs. *The VLDB Journal* 21, 4 (Aug. 2012), 509–534. https://doi.org/10.1007/s00778-011-0256-4

[52] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. 2013. DAGGER: A Scalable Index for Reachability Queries in Large Dynamic Graphs. *arXiv:1301.0977 [cs]* (Jan. 2013). arXiv:1301.0977 [cs]

[53] Niu Yudong, Yuchen Li, and Ju Fan. 2022. Local Clustering over Labeled Graphs: An Index-Free Approach [Technical Report]. (2022), 16.

[54] Shuang Zhou, Pingpeng Yuan, Ling Liu, and Hai Jin. 2018. MGTag: A Multi-Dimensional Graph Labeling Scheme for Fast Reachability Queries. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, Paris, 1372–1375. https://doi.org/10.1109/ICDE.2018.00153

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

[56] Linhong Zhu, Byron Choi, Bingsheng He, Jeffrey Xu Yu, and Wee Keong Ng. 2009. A Uniform Framework for Ad-Hoc Indexes to Answer Reachability Queries on Large Graphs. In *Database Systems for Advanced Applications*, Xiaofang Zhou, Haruo Yokota, Ke Deng, and Qing Liu (Eds.). Vol. 5463. Springer Berlin Heidelberg, Berlin, Heidelberg, 138–152. https://doi.org/10.1007/978-3-642-00887-0_12