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

Yue Pang
Peking University
Beijing, China
michelle.py@pku.edu.cn

Lei Zou
Peking University
Beijing, China
zoulei@pku.edu.cn

Yu Liu
Peking University
Beijing, China
dokiliu@pku.edu.cn

***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.**

***Index Terms*—Reachability, Index-free, Community, Dynamic graphs, Large graphs**

I. 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 [1]. 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 [2] has been proposed, which answers reachability queries approximately on-the-fly without any index. Sec. II 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 [3]. On these graphs, efficient reachability processing is of critical importance. For example, reachability queries can help detect fraudulent activities in e-commerce graphs [3] and conduct access control on social networks [4]. 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 [2] 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 [5], 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 IV.

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 [6], 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 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 is not only 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 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 on scale-free graphs.

Although IFCA primarily optimizes reachability querying on dynamic graphs with communities, it can also efficiently handle graphs that lack discernible community structure. Interestingly, we find that BiBFS is actually more efficient than state-of-the-art reachability algorithms on dynamic graphs when considering both query and update time, while IFCA performs at least comparably with BiBFS on graphs without discernible communities since the cost-based strategy selection can effectively switch to BiBFS when it fails to detect communities (Sec. VI-C).

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 on scale-free graphs given appropriate parameters.
- We empirically validate our approach's effectiveness on large-scale, real dynamic graphs with realistic workloads.

II. 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 [7], the index-based approaches can be further categorized into Label-Only and Label+G. Label-Only approaches, including [8–20], answer all queries accurately by only accessing the index; while Label+G approaches, including [7, 21–30], 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.

Recently, some reachability indexes have been developed to support incremental maintenance [7, 31–36]. 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 [35] proposes index maintenance procedures based on DAG maintenance when some SCCs merge or split due to edge insertions or deletions. TOL [34] and IP [7] 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 [36], 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 [37–40]. A recent index-free framework, ARROW [2], 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 [5], none of the existing reachability frameworks optimizes query processing over community structures to our knowledge, regardless of whether they are index-based.

III. Preliminary

Def. 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 Tab. I.

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 $s \to t$.

TABLE I: 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
$s \to t$	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

A. 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 [41], PPR has since been widely adopted as a measure of localized interest and relevance in graphs [42]. 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** [43], 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** [6, 44], 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 [45].

As mentioned in Sec. I, 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 $s, t \in V$, $s \to t \Leftrightarrow \mathbf{ppr}_s(t) > 0$, where $\mathbf{ppr}_s(t)$ denotes the Personalized PageRank (PPR) value of vertex t with respect to s.

Thus a baseline solution to use 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 [6, 46].

Baseline. According to Property 1, the push-based framework can be adapted as a baseline solution for reachability processing, as shown in Alg. 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 [6] and backward push [44], are both fit for Alg. 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 α, the threshold ϵ
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**

IV. Analysis of Baseline: Strengths, Weaknesses & Opportunities

In the previous section, we show the applicability of the baseline to answering reachability queries. However, it has

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

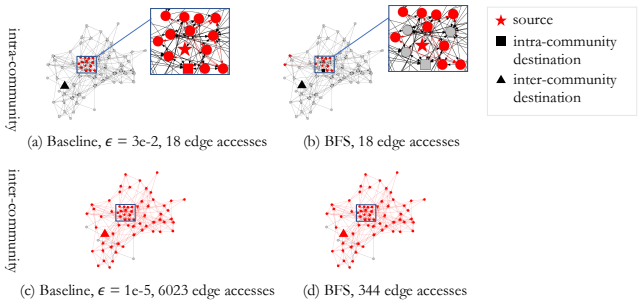


Fig. 1: The frontier expansion of BFS and Alg. 1 at different ϵ's in relation to the number of edge accesses.

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 [47], as an example. Highschool is a real graph with 70 vertices and 366 edges, representing the friendships between high school students. We show the results of the baseline with two different ϵ values and the most straightforward index-free approach, BFS, in Fig. 1. Each row shows an approach's frontier evolution; the visited vertices and edges are 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 [6], 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 Fig. 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 Fig. 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 (Alg. 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 inefficiency in handling inter-community reachable pairs, the baseline also has the following limitations:

- *No complexity bound:* there is no guarantee that the baseline's 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 bound on the time complexity on a class of scale-free graphs (Sec. V-D).

V. Our solution: IFCA

A. Algorithm Overview

To address the challenges in the different scenarios mentioned in Sec. IV, we design `IFCA` with the following components:

- *Probability-guided search (Sec. V-B, Alg. 3)*, an optimized version of the baseline (Alg. 1) that efficiently handles intra-community reachable pairs with relatively large ϵ;
- *Community contraction (Sec. V-C, Alg. 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 (Sec. V-D, Alg. 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 (Alg. 5) when it is cheaper.

Alg. 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). We find that the shrinking *step* does not impact IFCA's query efficiency substantially via the experiment in Sec. VI-A. In each iteration, probability-guided search (Alg. 3) is invoked in the forward and reverse directions (Lines 9, 12), followed by Alg. 4 (Sec. V-C), 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 (Alg. 6) estimates BiBFS to be cheaper (Lines 7-8) and switches to BiBFS (Alg. 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 Sec. V-C), 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 Sec. V-E 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*(ϵ_{cur}, f) **then**
10 return **true**
/* Try forward community contraction (Alg. 4) */
11 CommunityContraction(ϵ_{cur}, f)
/* Reverse probability-guided search */
12 **if** *ProbabilityGuidedSearch*(ϵ_{cur}, r) **then**
13 return **true**
/* Try reverse community contraction */
14 CommunityContraction(ϵ_{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($frontier_f, frontier_r$)

Algorithm 3: Probability-Guided Search

Input: The current residue threshold ϵ_{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 ϵ_{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 $\mathbf{exp}_s(u) \leftarrow \mathbf{true}$
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 $\mathbf{vis}_s(u_i) \leftarrow \mathbf{true}$
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**

B. Bidirectional Probability-Guided Search

Bidirectional search has been proven a simple yet effective strategy in reachability processing [48] 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 (Alg. 1).

The pseudocode of the probability-guided search from the forward direction is shown in Alg. 3. For the reverse direction, we simply reverse all the edge directions. Since we adopt a 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 (Sec. V-D).

C. Community Contraction

As analyzed in Section IV, 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 (Alg. 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), which is reflected by *conductance* [49], a metric widely used in community discovery. Conductance is defined over a set of vertices S as follows:

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

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 $\text{vol}(S) = \sum_{v \in S}(d_{out}(v) + d_{in}(v))$ 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.

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 [6]. Therefore, we tune a parameter ϵ_{pre} (Sec. VI-A) 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 Alg. 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.

D. Cost-Based Strategy Selection

Each iteration of community contraction (Sec. V-C) 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.

Algorithm 4: Community Contraction

Input: The current residue threshold ϵ_{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 ε_cur ≥ ε_pre then
2      return
3  if v^f_super ∉ V then
4      V ← V + {v^f_super}
5  forall v_i ∈ V s.t. vis_s(v_i) and v_i ≠ v^f_super do
6      if !exp_s(v_i) then
7          forall u_i ∈ N_out(v_i) do
8              if u_i ∉ N_out(v^f_super) then
9                  N_out(v^f_super) ← N_out(v^f_super) + {u_i}
10     V ← V − {v_i}
11 res_s(v^f_super) ← 1
12 vis_s(v^f_super) ← true
13 exp_s(v^f_super) ← false
14 intEdges_f ← 0
15 ε_cur ← ε_init
```

1) BiBFS: Alg. 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 (Alg. 2, Lines 18-19). Lines 3-12 take care of the forward direction, and lines 13-22 the reverse direction. The $\mathbf{vis}$ vectors are inherited from Alg. 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.

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).

Alg. 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 Sec. V-D4) for each operation (Lines 1-2). The algorithm is invoked from Alg. 2 at the start of of each iteration (Alg. 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 Sec. V-D3. If the estimated cost of BiBFS is lower than that of the probability-guided search, Alg. 6 will return $\mathbf{true}$, causing the loop in Alg. 2 to terminate and leading to BiBFS.

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**

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, Alg. 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]

We can infer the number of basic operations of the probability-guided search between two contraction invocations from Lem. 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 be the number of the remaining vertices; k_f and k_r be the estimated number of vertices 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 [52]. 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 Alg. 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

[2]The proof is similar to those in [6] and [50]. We omit the details due to limited space.

[3]Scale-free graphs are prevalent in the real world. Many highly dynamic graphs, such as social networks and web graphs, are scale-free [51].

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:

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

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 (Sec. VI).

Remark. Though the above analysis assumes that the graph is scale-free initially and after each contraction, it is also extendable to more generalized assumptions, such as power-law-bounded degree distributions [53].

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

Lemma 2. Given a directed graph $G = (V, E)$, Alg. 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 $|E'|$.

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.

E. Correctness and Complexity

Theorem 1 (Correctness)**.** Alg. 2 returns **true** if and only if $u \to v$.

Proof. ($\Rightarrow$) If Alg. 2 returns **true** during probability-guided search (Alg. 3, Line 10), the currently probed vertex has been visited from both directions. Since community contraction only contracts the visited vertices and retains all their neighbors as the super-vertex's neighbors, it preserves reachability, and thus $u \to v$. If Alg. 2 returns **true** during traversal (Alg. 5, Lines 9 and 19), some vertex on the reverse frontier is reachable from the forward frontier, so $u \to v$.

($\Leftarrow$) Prove by contraction. If $u \to v$ but Alg. 2 returns **false**, then either one of the two super-vertices has zero out- or in-degree (Alg. 2, Lines 16-17), or BiBFS fails to find a path (Alg. 5, Line 23). Therefore, there must be a path from u to v without any frontier vertex, meaning that an explored vertex has an edge to an unvisited vertex, which is impossible. □

Theorem 2. On graphs that are scale-free initially and after each contraction, Alg. 2 (IFCA) answers a positive query in time $SubLinear(n + m)$ if $\epsilon_{pre} < c$.

Proof. According to the analysis in Sec. V-D3, 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 Lem. 2, BiBFS on the reduced graph runs in time $O(|V'| + |E'|)$, where $|V'| < n$ and $|E'| < m$. Therefore, the overall time complexity is $SubLinear(n + m)$. □

Thm. 2 only applies to positive queries, since Eq. (1) assumes the PPR values with respect to the source are non-zero for each vertex. Note that it is impossible for any online algorithm to answer a negative query in sublinear time in the worst case, since it is unsafe to terminate with a negative answer without visiting all the reachable vertices and edges.

VI. Experimental Evaluation

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++[5], and we adopt the C++ implementations of all the competitors kindly provided by their authors.

Datasets. We select ten real graphs for our experiments, the statistics of which are listed in Tab. II. EN and WT are communication networks with edges representing emails and messages. EP, DF, FL, LJ and FR are social networks. WG, WD, WF, ZS and DL are web graphs where edges represent hyperlinks between web pages. Among them, EN, EP, and FL are datasets used in recent work [2]. WT is obtained from SNAP [54], while the rest are obtained from KONECT [47].

Since IFCA aims to speed up reachability by leveraging community structures, we categorize these real graphs into those with discernible communities and those without by their clustering coefficients c, displayed in the upper and lower

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

[5]Our implementation is available at https://github.com/SoftlySpoken/IFCA.

TABLE II: Real Datasets.

Category	Name	Dataset	$n = \|V\|$	$m = \|E\|$ (Initial)	# Edge insertions	# Edge deletions	Negative queries (%)	Clustering coefficient
Graphs with discernible communities	EN	Enron	87,273	16,095	1,453,087	295,027	58.37	0.071648
	EP	Epinions	131,828	42,068	799,304	824,962	56.78	0.065679
	DF	Digg friends	279,630	86,582	3,375,054	1,548,274	67.99	0.061426
	FL	Flickr	2,302,925	1,657,000	47,588,228	31,953,578	28.71	0.107648
	LJ	LiveJournal	4,847,571	68,993,773	65,051,622	61,627,852	36.53	0.117916
	FR	Friendster	68,349,466	129,307,393	245,684,047	232,753,308	59.73	0.017372
Graphs without discernible communities	WT	wiki-talk-temporal	1,140,149	165,479	10,977,252	2,975,489	47.87	0.002204
	WG	Wikipedia growth (en)	1,870,709	1,997,657	37,955,488	27,235,531	14.06	0.003089
	WD	Wikipedia dynamic (de)	2,162,457	3,441,403	55,029,876	27,364,893	8.61	0.007344
	WF	Wikipedia dynamic (fr)	2,162,618	2,348,976	39,331,205	17,239,188	12.77	0.004945
	ZS	Zhishi	7,827,192	3,242,098	61,599,875	58,357,776	55.14	0.002079
	DL	DBpedia Links	18,268,992	6,826,878	129,710,688	122,883,810	18.36	0.001691

halves of Tab. II, respectively; graphs with $c \geq 0.01$ are viewed as having discernible communities.

All of the real graphs are *temporal*, i.e., each edge has a timestamp, except LJ, FR, ZS and DL, which are large static graphs for complementing the relatively small sizes of publicly available temporal graphs. We randomly assign unique timestamps to the edges in LJ, FR, ZS and DL. The edges with the minimum timestamp appear in the initial state, and all the rest are edge inserts. WD and WF 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 [2].

We generate synthetic graphs using stochastic block models (SBMs) [55] 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 from all the vertices with at least one out-edge and all those with at least one in-edge in the current snapshot, respectively, and discarding queries whose source and destination vertices are identical. The total number of queries on each dataset is 10^6, and the number of intervals is 20, resulting in 50,000 queries per batch. We generate a batch of 50,000 queries in the same way for each synthetic graph.

Goals. We conduct experiments to validate that `IFCA` meets our expectations in the following aspects:

- The influence of the parameters (ϵ_{pre}, ϵ_{init}, α, and `step`) on `IFCA`'s efficiency should be minor, meaning that they can be chosen heuristically (Sec. VI-A);
- The optimizations we propose, namely community contraction and cost-based strategy selection, should improve the efficiency of our approach (Sec. VI-B);
- `IFCA`, as an index-free community-aware algorithm, should significantly outperform state-of-the-art algorithms on graphs with discernible communities and perform at least comparably with them on graphs without discernible communities (Sec. VI-C).

A. Parameter Study

There are four parameters in our main algorithm (Alg. 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. Note that the y-axes of the figures in this section have the default counting unit of $10^0 = 1$ if their counting units are not explicitly shown.

1) The termination residue threshold ϵ_{pre}: The average query time with varying ϵ_{pre} is shown in Fig. 2. α, ϵ_{init} and `step` are fixed as the default in Sec. VI-A4. The time complexity of our algorithm (Thm. 2) seems to suggest that ϵ_{pre} should be as small as possible to achieve the least query time, but Fig. 2 shows that the average query time first decreases and then increases as ϵ_{pre} grows. Such a trend shows that Thm. 2 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 Fig. 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 sublinearly (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.

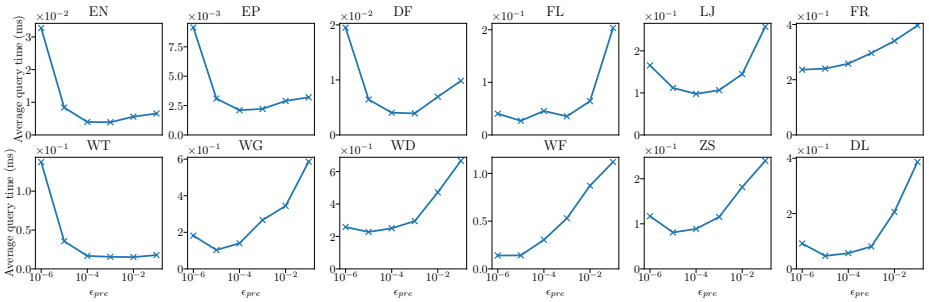


Fig. 2: Average query time varying ϵ_{pre}.

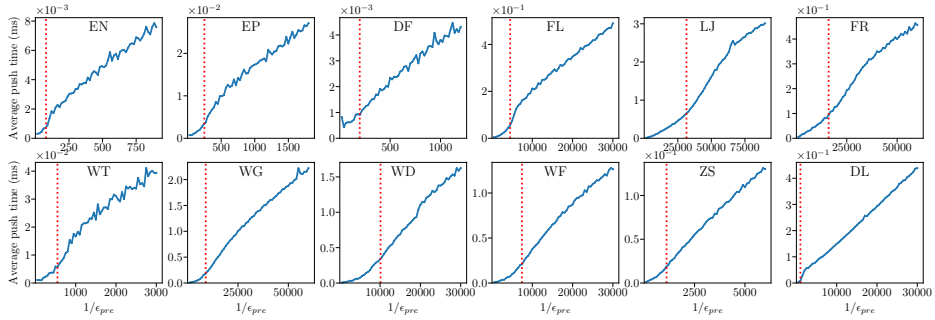


Fig. 3: Average push time varying $1/\epsilon_{pre}$ (explaining Fig. 2).

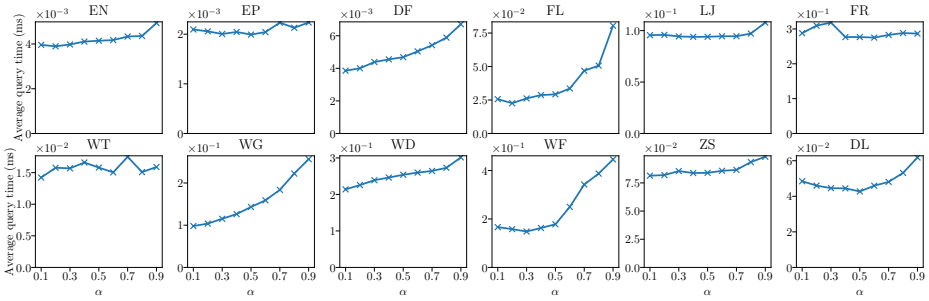


Fig. 4: Average query time varying α.

2) The teleportation constant α: Intuitively, α determines how likely the random walks underlying PPR will halt than proceed, thus a large α is disadvantageous for reachability. To study α's effect on `IFCA`'s efficiency, we fix ϵ_{pre} as the best found in Sec. VI-A1, and ϵ_{init} and `step` as the default in Sec. VI-A4. We find that when $\alpha > 0.5$, the average query time grows sharply with α on all graphs except WT, where the fluctuation in query time caused by α is negligible, as shown in Fig. 4.

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` (Thm. 2). 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 VI-A1 and VI-A2, 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 insignificant, as shown in Fig. 5.

4) Default parameters: To prove that `IFCA`'s advantage over existing approaches is not reliant on the best parameters, we select `IFCA`'s parameters heuristically in the subsequent experiments as follows: $\epsilon_{pre} = 100/m$, where m is the number of edges on the current snapshot, following the intuition that ϵ_{pre} should be smaller on larger and denser graphs; $\alpha = 0.1$ following recent work in local community detection [56]; $\epsilon_{init} = 100\epsilon_{pre}$; and `step` $= 10$.

B. 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`), and the

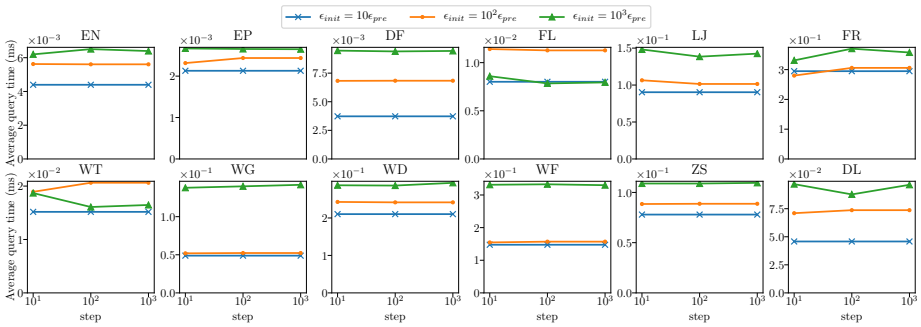
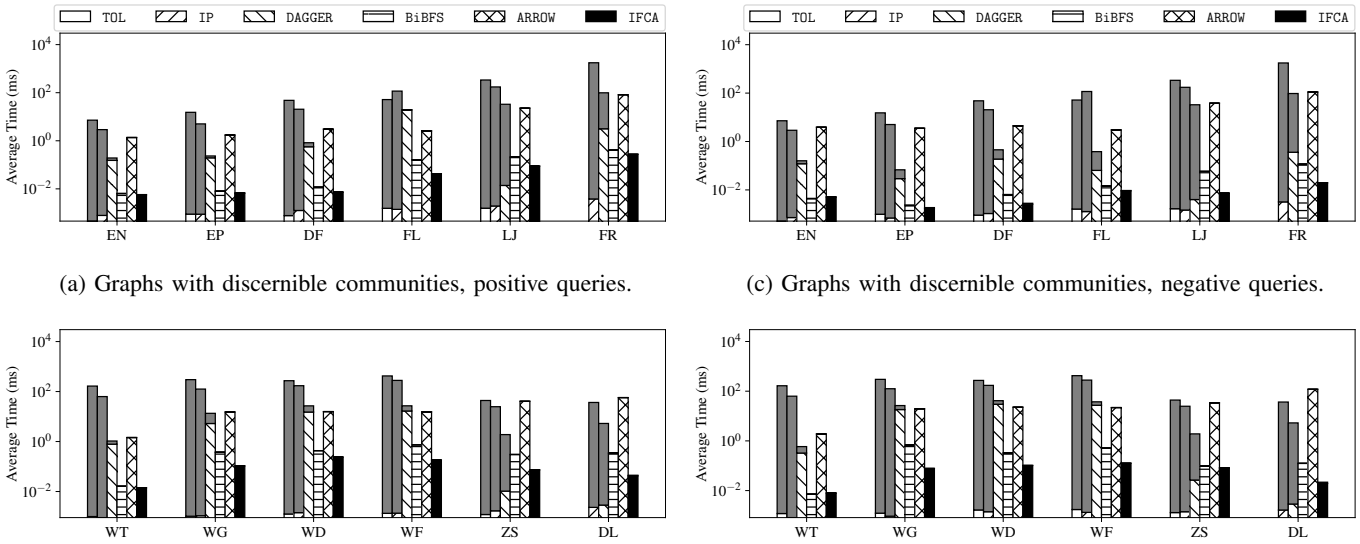


Fig. 5: Average query time varying ϵ_{init} and `step`.



(a) Graphs with discernible communities, positive queries.

(c) Graphs with discernible communities, negative queries.

(b) Graphs without discernible communities, positive queries.

(d) Graphs without discernible communities, negative queries.

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

TABLE III: Average query time (ms) of `IFCA` and BiBFS.

		EN	EP	DF	FL	LJ	FR	WT	WG	WD	WF	ZS	DL
Positive queries	BiBFS	0.00634	0.00793	0.0116	0.157	0.209	0.408	0.0162	0.373	0.433	0.742	0.302	0.344
	`IFCA`	0.00562	0.00661	0.00721	0.0424	0.0881	0.275	0.0141	0.105	0.238	0.182	0.0728	0.0424
	Speedup	1.13	1.20	1.61	3.72	2.38	1.48	1.15	3.54	1.82	4.08	4.16	8.10
Negative queries	BiBFS	0.00432	0.00222	0.00623	0.0144	0.0582	0.116	0.00706	0.689	0.326	0.51	0.0979	0.124
	`IFCA`	0.00503	0.0017	0.00257	0.00916	0.00724	0.0195	0.00785	0.0776	0.103	0.126	0.0819	0.0193
	Speedup	0.858	1.31	2.43	1.57	8.03	5.92	0.899	8.89	3.18	4.05	1.20	6.41
Overall	Speedup	1.04	1.27	1.98	3.71	2.40	1.57	1.07	4.02	1.91	4.08	3.02	8.04

full method (`IFCA`). Fig. 7 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 can be drawn from Fig. 7:

- `Base` is a competitive approximate algorithm, which performs comparably with `IFCA` at 90% accuracy. However, it is unsuitable for accurate querying, being orders of magnitude slower than `IFCA` at 100% accuracy.
- `Contract` guarantees 100% accuracy and is consistently faster than `Base`@100% except on DL, verifying the effectiveness of community contraction.
- `IFCA` is consistently faster than `Contract`, verifying the effectiveness of cost-based strategy selection.

Cost Model's effectiveness. To verify our cost model's ability in choosing a near-optimal switching point, 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 Tab. IV. `IFCA` employs the cost-based strategy selection scheme, while `Contract` and BiBFS represent two extremes: never switching to BiBFS, and switching at the beginning.

`IFCA`'s average query time is closest to the oracle on all

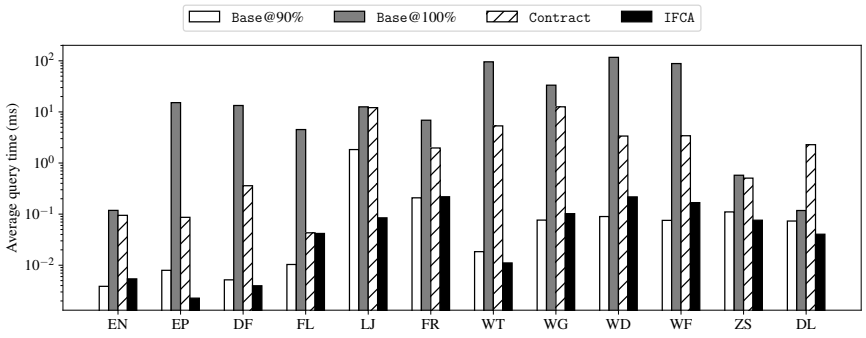


Fig. 7: Relation between precision and average query time of our proposed techniques.

the datasets. Note that the average query time of Contract is generally closer to the oracle on the graphs with discernible communities than those without, while the opposite is true for BiBFS, which shows the applicability of community contraction to graphs with discernible communities.

TABLE IV: Performance of the cost model. (Time in *ms*)

	Oracle	IFCA	Contract	BiBFS
EN	0.00265	**0.00541**	0.0947	0.0056
EP	0.00152	**0.00227**	0.0865	0.00288
DF	0.00249	**0.00396**	0.359	0.00785
FL	0.0223	**0.0418**	0.0431	0.155
LJ	0.0458	**0.0846**	12.1	0.203
FR	0.098	**0.219**	1.97	0.343
WT	0.00689	**0.0111**	5.33	0.0118
WG	0.0698	**0.102**	12.6	0.411
WD	0.124	**0.217**	3.37	0.416
WF	0.0792	**0.168**	3.42	0.685
ZS	0.0391	**0.0760**	0.506	0.23
DL	0.0286	**0.0404**	2.28	0.325

C. Comparison With State of the Art

In this subsection, we compare the performance of IFCA on reachability queries on real dynamic graphs with state-of-the-art approaches, including TOL [34], IP [7] and DAGGER [35], the state-of-the-art index-based reachability frameworks on dynamic graphs; ARROW [2], 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 [35]. DBL is excluded from our comparison because it cannot handle edge deletions. We set $k = 2$, $h = 2$ and $\mu = 100$ for IP as advised in [7], since all the snapshots of the real graphs are sparse ($d_{avg} < 2$). We set $c_{walkLength} = 1$ as advised in [2] and gradually enlarge $c_{numWalks}$ with the initial value and step as 0.01 until the accuracy exceeds 95%.

We plot each method's average query and update time on each real graph in two stacked bar charts (Fig. 6), representing graphs with and without discernible communities, respectively, where the average update time is stacked upon the average query time and shown in grey. The corresponding bar is omitted if a method runs out of memory.

1) Update time: TOL has the longest average update time, closely followed by IP, except on FL where IP is slightly slower than TOL. TOL runs out of memory during index update on the largest datasets, FR and DL. IP generally handles updates faster than TOL because it needs to update a smaller number of labels, consistent with the trend and analysis in [7]. DAGGER has much shorter average update time than TOL and IP because it can update its index incrementally when SCCs split or merge, while TOL and IP can only reconstruct at least part of their indexes. The update time of TOL and IP dominates their query time on all the datasets by up to five orders of magnitude. IFCA, BiBFS and ARROW have similar average update time, which is at least an order of magnitude shorter than DAGGER's. The update time of IFCA, BiBFS and ARROW is dominated by their query time by at least an order of magnitude.

2) Query time: TOL and IP perform comparably and have the shortest query time on all the datasets, except when TOL runs out of memory during index maintenance on the largest datasets, FR and DL. They handle queries significantly faster than IFCA, which is expected because IFCA does not build any index to prune its search. IFCA is consistently faster than BiBFS. We give the average query time of IFCA and BiBFS in Tab. III for direct comparison. The overall speedup of IFCA over BiBFS exceeds 2x on half of the datasets. IFCA is faster than BiBFS on positive queries on all the datasets, showing the effectiveness of its optimization strategies. IFCA is also faster than BiBFS on negative queries on nearly all the datasets, because the probability-guided search can cover the entire neighborhood of the source or destination vertex and conclude that the pair is unreachable in fewer iterations than BiBFS and reduce the search cost of the other direction if the neighborhood does not contain cycles and vertices with very large out- or in-degrees. However, if the neighborhoods of the source and destination vertices in a negative query do not have such properties, IFCA will incur some overhead during the probability-guided search. DAGGER has longer average query time than BiBFS on most datasets because it conducts pruned unidirectional DFS, and the disadvantage of unidirectional search and depth-first strategy outweighs the advantage of pruning by interval labels and traversing the DAG

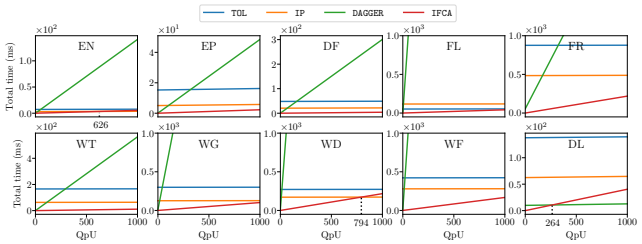


Fig. 8: Total time of IFCA and index-based methods varying QpU.

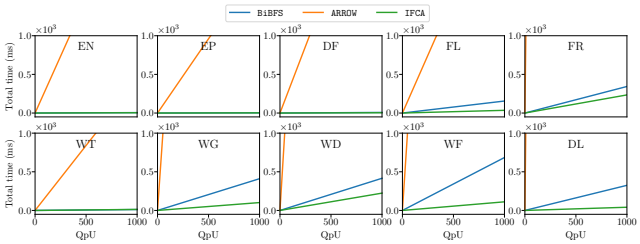


Fig. 9: Total time of IFCA and index-free methods varying QpU.

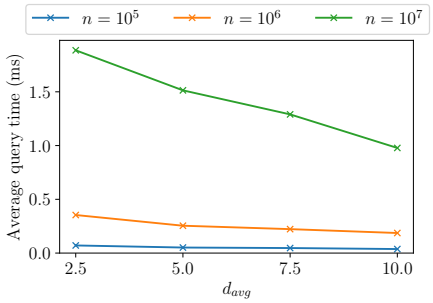


Fig. 10: Average query time on synthetic graphs.

instead of the original graph on these datasets. ARROW has the longest average query time on most datasets because it is an approximate method based on random walks and many random walks are needed for it to answer a query accurately.

Overall, considering both update and query time, index-free approaches are advantageous compared to their index-based counterparts. IFCA performs the best, while BiBFS is the second-best.

3) Varying QpU: The more queries there are compared with updates, the more efficient index-based methods will become. To see when the advantage is reversed between IFCA and the index-based methods (TOL, IP and DAGGER), we plot their total time of performing an update and a certain number of queries varying the query-per-update ratio (QpU) in Fig. 9. A line's starting point on the y-axis is the method's average update time, and the line's growth rate indicates the method's average query time. TOL's line starts at the highest points on all the datasets, followed by IP's, DAGGER's and IFCA's. TOL and IP's lines are almost flat due to their short average query time, while DAGGER and IFCA's grow visibly. However, the lines of TOL, IP and IFCA do not intersect with QpU below 1000 on all the datasets except EN and WD, since the update time of TOL and IP dominates IFCA's query time. The lines of DAGGER and IFCA do not intersect with QpU below 1000 on all the datasets except DL, on which DAGGER handles queries faster than IFCA. Therefore, IFCA is still widely applicable to query-heavy scenarios. Note that on highly dynamic graphs, such as the Alibaba e-commerce graph [3] which has up to 20,000 updates per second, and WF which has up to 9,180 updates per second, it is unlikely for QpU to be very large.

D. 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 VI-B and VI-C since the synthetic graphs are denser than the real graphs. The average query time of IFCA on synthetic graphs is shown in Fig. 10[6]. 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 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 $n = 10^6$).

VII. 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 search and switching to BiBFS and chooses the cheaper strategy accordingly. Experimental studies show that our approach is significantly more efficient than both the index-based and index-free state-of-the-art methods on large-scale real dynamic graphs. In the future, we plan to explore adapting our approach for various forms of constrained reachability queries.

[6]To expose the effect of the synthetic graphs' scale on IFCA 's efficiency, we fix $\epsilon_{pre} = 0.0001$. Other parameters follow the setting in Sec. VI-A4.

ACKNOWLEDGMENT

This work was supported by NSFC under grant 61932001 and U20A20174. Lei Zou is the corresponding author of this paper.

REFERENCES

[1] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing," *Proc. VLDB Endow.*, vol. 11, no. 4, p. 420–431, dec 2017. [Online]. Available: https://doi.org/10.1145/3186728.3164139

[2] N. Sengupta, A. Bagchi, M. Ramanath, and S. Bedathur, "ARROW: Approximating Reachability Using Random Walks Over Web-Scale Graphs," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. Macao, Macao: IEEE, Apr. 2019, pp. 470–481.

[3] X. Qiu, W. Cen, Z. Qian, Y. Peng, Y. Zhang, X. Lin, and J. Zhou, "Real-time constrained cycle detection in large dynamic graphs," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1876–1888, 2018.

[4] I. B. Dhia, "Access control in social networks: a reachability-based approach," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, 2012, pp. 227–232.

[5] X. Huang, L. V. Lakshmanan, and J. Xu, "Community search over big graphs: Models, algorithms, and opportunities," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017, pp. 1451–1454.

[6] R. Andersen, F. Chung, and K. Lang, "Local Graph Partitioning using PageRank Vectors," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. Berkeley, CA, USA: IEEE, 2006, pp. 475–486.

[7] H. Wei, J. X. Yu, C. Lu, and R. Jin, "Reachability querying: An independent permutation labeling approach," *The VLDB Journal*, vol. 27, no. 1, pp. 1–26, Feb. 2018.

[8] R. Agrawal, A. Borgida, and H. V. Jagadish, "Efficient management of transitive relationships in large data and knowledge bases," in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data - SIGMOD '89*. Portland, Oregon, United States: ACM Press, 1989, pp. 253–262.

[9] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and Distance Queries via 2-Hop Labels," *SIAM Journal on Computing*, 2003.

[10] H. He, H. Wang, J. Yang, and P. S. Yu, "Compact reachability labeling for graph-structured data," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management - CIKM '05*. Bremen, Germany: ACM Press, 2005, p. 594.

[11] Haixun Wang, Hao He, Jun Yang, P. Yu, and J. Yu, "Dual Labeling: Answering Graph Reachability Queries in Constant Time," in *22nd International Conference on Data Engineering (ICDE'06)*. Atlanta, GA, USA: IEEE, 2006, pp. 75–75.

[12] Y. Chen and Y. Chen, "An Efficient Algorithm for Answering Graph Reachability Queries," in *2008 IEEE 24th International Conference on Data Engineering*. Cancun, Mexico: IEEE, Apr. 2008, pp. 893–902.

[13] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu, "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*, 2008.

[14] R. Jin, Y. Xiang, N. Ruan, and H. Wang, "Efficiently answering reachability queries on very large directed graphs," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data - SIGMOD '08*. Vancouver, Canada: ACM Press, 2008, p. 595.

[15] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry, "3-HOP: A high-compression indexing scheme for reachability query," in *Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD '09*. Providence, Rhode Island, USA: ACM Press, 2009, p. 813.

[16] J. Cai and C. K. Poon, "Path-hop: Efficiently indexing large graphs for reachability queries," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management - CIKM '10*. Toronto, ON, Canada: ACM Press, 2010, p. 119.

[17] S. J. van Schaik and O. de Moor, "A memory efficient reachability data structure through bit vector compression," in *Proceedings of the 2011 International Conference on Management of Data - SIGMOD '11*. Athens, Greece: ACM Press, 2011, p. 913.

[18] J. Cheng, S. Huang, H. Wu, and A. W.-C. Fu, "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*. New York, New York, USA: ACM Press, 2013, p. 193.

[19] R. Jin and G. Wang, "Simple, fast, and scalable reachability oracle," *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1978–1989, Sep. 2013.

[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 *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management - CIKM '13*. San Francisco, California, USA: ACM Press, 2013, pp. 1601–1606.

[21] L. Chen, A. Gupta, and M. E. Kurul, "Stack-based Algorithms for Pattern Matching on DAGs," p. 12, 2005.

[22] S. Trißl and U. Leser, "Fast and practical indexing and querying of very large graphs," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data - SIGMOD '07*. Beijing, China: ACM Press, 2007, p. 845.

[23] L. Zhu, B. Choi, B. He, J. X. Yu, and W. K. Ng, "A Uniform Framework for Ad-Hoc Indexes to Answer Reachability Queries on Large Graphs," in *Database Systems for Advanced Applications*, X. Zhou, H. Yokota, K. Deng, and Q. Liu, Eds. Berlin, Heidelberg: Springer

Berlin Heidelberg, 2009, vol. 5463, pp. 138–152.

[24] R. Jin, N. Ruan, S. Dey, and J. Y. Xu, "SCARAB: Scaling reachability computation on large graphs," in *Proceedings of the 2012 International Conference on Management of Data - SIGMOD '12*. Scottsdale, Arizona, USA: ACM Press, 2012, p. 169.

[25] H. Yıldırım, V. Chaoji, and M. J. Zaki, "GRAIL: A scalable index for reachability queries in very large graphs," *The VLDB Journal*, vol. 21, no. 4, pp. 509–534, Aug. 2012.

[26] S. Seufert, A. Anand, S. Bedathur, and G. Weikum, "FERRARI: Flexible and efficient reachability range assignment for graph indexing," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. Brisbane, QLD: IEEE, Apr. 2013, pp. 1009–1020.

[27] R. R. Veloso, L. Cerf, W. M. Junior, and M. J. Zaki, "Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach," 2014.

[28] L. Li, W. Hua, and X. Zhou, "HD-GDD: High dimensional graph dominance drawing approach for reachability query," *World Wide Web*, vol. 20, no. 4, pp. 677–696, Jul. 2017.

[29] J. Su, Q. Zhu, H. Wei, and J. X. Yu, "Reachability Querying: Can It Be Even Faster?" *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 3, pp. 683–697, Mar. 2017.

[30] S. Zhou, P. Yuan, L. Liu, and H. Jin, "MGTag: A Multi-Dimensional Graph Labeling Scheme for Fast Reachability Queries," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. Paris: IEEE, Apr. 2018, pp. 1372–1375.

[31] H. V. Jagadish, "A compression technique to materialize transitive closure," *ACM Transactions on Database Systems*, vol. 15, no. 4, pp. 558–598, Dec. 1990.

[32] R. Schenkel, A. Theobald, and G. Weikum, "Efficient Creation and Incremental Maintenance of the HOPI Index for Complex XML Document Collections," in *21st International Conference on Data Engineering (ICDE'05)*. Tokyo, Japan: IEEE, 2005, pp. 360–371.

[33] R. Jin, N. Ruan, Y. Xiang, and H. Wang, "Path-tree: An efficient reachability indexing scheme for large directed graphs," *ACM Transactions on Database Systems*, vol. 36, no. 1, pp. 1–44, Mar. 2011.

[34] A. D. Zhu, W. Lin, S. Wang, and X. Xiao, "Reachability queries on large dynamic graphs: A total order approach," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. Snowbird Utah USA: ACM, Jun. 2014, pp. 1323–1334.

[35] H. Yildirim, V. Chaoji, and M. J. Zaki, "DAGGER: A Scalable Index for Reachability Queries in Large Dynamic Graphs," *arXiv:1301.0977 [cs]*, Jan. 2013.

[36] Q. Lyu, Y. Li, B. He, and B. Gong, "DBL: Efficient Reachability Queries on Dynamic Graphs (Complete Version)," *arXiv:2101.09441 [cs]*, Jan. 2021.

[37] U. Feige, "A fast randomized logspace algorithm for graph connectivity," *Theoretical Computer Science*, vol. 169, no. 2, pp. 147–160, 1996.

[38] I. Gorodezky and I. Pak, "Generalized loop-erased random walks and approximate reachability," *Random Structures & Algorithms*, vol. 44, no. 2, pp. 201–223, 2014.

[39] M. Starnini, A. Baronchelli, A. Barrat, and R. Pastor-Satorras, "Random walks on temporal networks," *Physical Review E*, vol. 85, no. 5, p. 056115, 2012.

[40] A. Anagnostopoulos, R. Kumar, M. Mahdian, E. Upfal, and F. Vandin, "Algorithms on evolving graphs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012, pp. 149–160.

[41] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Tech. Rep., 1999.

[42] D. F. Gleich, "Pagerank beyond the web," *SIAM Review*, 2015.

[43] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments," *Internet Mathematics*, vol. 2, no. 3, pp. 333–358, 2005.

[44] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. S. Mirrokni, and S.-H. Teng, "Local computation of pagerank contributions," in *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 2007, pp. 150–165.

[45] H. Wu, J. Gan, Z. Wei, and R. Zhang, "Unifying the global and local approaches: an efficient power iteration with forward push," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1996–2008.

[46] S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang, "FORA: Simple and Effective Approximate Single-Source Personalized PageRank," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Halifax NS Canada: ACM, Aug. 2017, pp. 505–514.

[47] J. Kunegis, "Konect: the koblenz network collection," in *Proceedings of the 22nd international conference on world wide web*, 2013, pp. 1343–1350.

[48] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing breadth-first search," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–10.

[49] M. Mihail, "Conductance and convergence of markov chains-a combinatorial treatment of expanders," in *30th Annual Symposium on Foundations of Computer Science*, 1989, pp. 526–531.

[50] P. Lofgren, S. Banerjee, and A. Goel, "Personalized pagerank estimation and search: A bidirectional approach," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, 2016, pp. 163–172.

[51] M. Jiang, A. W.-C. Fu, R. C.-W. Wong, and Y. Xu, "Hop doubling label indexing for point-to-point distance querying on scale-free networks," *Proceedings of the*

VLDB Endowment, vol. 7, no. 12, 2014.

[52] B. Bahmani, A. Chowdhury, and A. Goel, "Fast Incremental and Personalized PageRank," *arXiv:1006.2880 [cs]*, Aug. 2010.

[53] P. Brach, M. Cygan, J. Łącki, and P. Sankowski, "Algorithmic complexity of power law networks," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2016, pp. 1306–1325.

[54] J. Leskovec and A. Krevl, "Snap datasets: Stanford large network dataset collection," 2014.

[55] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.

[56] N. Yudong, Y. Li, and J. Fan, "Local Clustering over Labeled Graphs: An Index-Free Approach [Technical Report]," p. 16, 2022.