PTIL: Partitioned Tree-Cover Interval Labeling for Restricted-Domain Reachability Queries

Huangleshuai He
huangleshuai.he@unsw.edu.au
The University of New South Wales
Sydney, Australia

Zhengyi Yang
zhengyi.yang@sydney.edu.au
The University of Sydney
Sydney, Australia

Yifu Tang*
craigtang@swin.edu.au
Swinburne University of Technology
Melbourne, Australia

Zebin Chen
zebin.chen1@unsw.edu.au
The University of New South Wales
Sydney, Australia

Dong Wen
dong.wen@unsw.edu.au
The University of New South Wales
Sydney, Australia

Qi Luo
luoqi2018@mail.sdu.edu.cn
Shandong University
Qingdao, China

Tianming Zhang
tmzhang@zjut.edu.cn
Zhejiang University of Technology
Hangzhou, China

Abstract

Reachability queries are a core operation in graph analytics. Existing labeling methods assume a global query space and construct uniform labels for all vertices. In practice, however, many systems issue queries over semantically structured endpoint domains such as $V_s \times V_t$, where only a subset of vertices serve as meaningful sources or targets. This mismatch makes global labeling redundant and inefficient. We propose *Partitioned Tree-Cover Interval Labeling* (PTIL), a restricted-domain reachability indexing framework that extends classical Tree-Cover Labeling. PTIL timestamps only target vertices and stores compact disjoint intervals at sources, reducing each query to a single membership test while avoiding unnecessary global labels. By partitioning targets into groups, PTIL provides an explicit and predictable space–time trade-off: finer grouping yields lower latency at the cost of larger index size. PTIL also supports dynamic updates to endpoint domains. We further develop two instantiations: **PTIL-g**, a grouped scheme for scalable domain-aligned indexing, and **PTIL-p**, which additionally leverages query distributions for further prioritization when available. Experiments on real and synthetic graphs show that **PTIL-g** achieves order-of-magnitude lower query latency than state-of-the-art baselines, while **PTIL-p** provides additional gains under skewed workloads, with near-linear and predictable space–time behavior.

Keywords

Reachability Query, Graph Indexing, Restricted-Domain

1 Introduction

Graphs are a universal abstraction for real-world data, where vertices denote entities and edges represent relationships. They are fundamental in domains such as biology, finance, social networks, and knowledge graphs. In graph analytics, reachability determines whether a directed path exists from a source to a target vertex, supporting tasks such as provenance analysis, program analysis, XML query processing, and large-scale graph mining [5, 8, 23, 24, 26, 35]. Over the past three decades, extensive research has produced many reachability indexing[1] and querying techniques [1–4, 6, 7, 9–13, 15–17, 20–22, 25, 27–40, 42, 43, 45].

However, real-world reachability workloads are rarely defined over the full vertex-pair space $V \times V$. In many practical systems, queries are restricted to specific subsets of vertices that play distinct endpoint roles, forming a source-target endpoint domain $V_s \times V_t$. We refer to this setting as *restricted-domain reachability*, where queries are issued over $V_s \times V_t$ rather than the full vertex-pair space $V \times V$. This paper focuses on DAG reachability, following a common setting in reachability indexing; for general directed graphs, the standard SCC condensation step can be applied first to obtain a DAG of strongly connected components.

Such endpoint role restrictions arise from application semantics, operational logic, or temporal ordering rather than graph topology. In practice, such endpoint domains may be provided by applications, derived from schema-level or role-level semantics, specified by business or monitoring rules, or inferred from query logs and access statistics. Even when vertices are structurally homogeneous, only certain vertices meaningfully act as sources and others as targets. Under this view, unrestricted $V \times V$ reachability is rarely the true workload of interest.

1.1 Reachability under Restricted-Domain

Restricted-domain reachability workloads arise in two forms, both defined over semantically meaningful endpoint subsets.

Part A: Role-restricted domains in heterogeneous contexts. In many applications, vertices naturally assume different semantic roles, and queries are issued only between role-specific subsets, forming structured domains $V_s \times V_t$.

Finance. Let V_s denote monitored, suspicious, or high-risk entities, and V_t denote investigation targets (e.g., beneficiary accounts or controlled firms). Queries ask whether funds or ownership influence can flow from entities in V_s to entities in V_t through multi-hop relations. Reachability outside this domain (e.g., arbitrary firm-to-firm connectivity) is rarely of operational interest.

Social networks. Let V_s be content originators, influencers, or suspicious accounts, and V_t be target users or monitored audiences. Queries determine whether information can propagate from V_s to V_t. Connectivity between arbitrary user pairs is typically irrelevant to tasks such as event forensics or rumor tracing.

*Corresponding author.

EDBT '27, Lille (France)

© 2026 Copyright held by the owner/author(s). Published on OpenProceedings.org under ISBN 978-3-89318-106-3, series ISSN 2367-2005. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

[1] In reachability, an index typically consists of labels; thus, we use the two terms interchangeably.

10.48786/edbt.2027.04

Biology and epidemiology. Let V_s represent sources such as stimuli, pathogens, or carriers, and V_t represent downstream targets such as regulated genes, infected populations, or affected species. Queries focus on whether effects originating in V_s can reach V_t through biological pathways. Reachability between arbitrary biological entities often has no direct interpretability.
Part B: Role-induced domains in structurally homogeneous graphs. Even when vertices represent homogeneous entities, implicit endpoint roles emerge from temporal or functional semantics, again restricting queries to structured subdomains.
Academic citation networks. Let V_s be earlier publications and V_t be recent or representative papers. Queries ask whether earlier work influences later research. Connectivity among older papers or among newer papers is rarely queried, since such relations do not correspond to forward knowledge flow.
Software call graphs. Let V_s be internal or utility functions and V_t be externally visible APIs. Queries determine whether internal changes can propagate to public interfaces. Arbitrary function-to-function reachability outside this domain has limited relevance for impact analysis.

Taken together, these examples demonstrate that practical reachability workloads are defined over semantically meaningful endpoint domains $V_s \times V_t$. Queries outside these domains either lack operational meaning or are unnecessary for the target tasks.

Indexing methods that materialize global $V \times V$ reachability therefore allocate resources to regions that never serve as query endpoints, incurring redundant storage and query overhead.

1.2 Limitations of Existing Methods

Despite substantial prior research, we observe three major limitations in existing approaches:

- (L1) **Lack of support for restricted-domain reachability queries**: Most existing methods assume a global query space $V \times V$ and are not designed to optimize restricted-domain queries (e.g., $V_1 \times V_2$) that arise in practice. Consequently, they maintain full global indexes even when queries are semantically restricted, causing unnecessary storage overhead and failing to provide targeted acceleration for the query subdomains that matter most.
- (L2) **Lack of effective space–time trade-offs**: Existing techniques lack explicit controls that allow practitioners to flexibly trade additional space for predictable reductions in query latency or to allocate capacity toward hot vertices under skewed workloads. Most methods provide no adjustable parameters [39, 45]. Even in parameterized approaches [28, 30, 36], although tuning parameters can affect index size, they fail to achieve continuous and effective space–time adjustment; often, only a single suitable parameter setting exists for a given graph.
- (L3) **Limited scalability on large, dense graphs**: Performance in existing methods deteriorates as average degree increases, with query latency and index maintenance costs rising sharply. As a result, many experimental studies rely on sparse datasets with low average degree, limiting their scalability and representativeness for real-world graphs that are denser and more complex.

1.3 Our Approach and Contributions

In this paper, we propose *Partitioned Tree-Cover Interval Labeling* (PTIL), a reachability indexing framework designed to align index construction with the actual query domain.

Traditional Tree-Cover methods assign interval labels to all vertices, assuming a global $V \times V$ query space. PTIL departs from this paradigm by introducing a restricted-domain-aware design.

PTIL assigns timestamps only to target vertices and materializes interval summaries only at source vertices, so that the index stores exactly the reachability information required for queries over a structured domain $V_s \times V_t$. A reachability query reduces to a single membership test, avoiding traversal and eliminating redundant global labels. This paper introduces a *partitioned* design by first dividing the restricted query domain into separate groups of source and target vertices. Labels (or positions) are assigned *only to target vertices*, while each source vertex maintains a compact set of disjoint intervals summarizing which targets it can reach. Each source vertex records different sets of intervals corresponding to different target groups, allowing local reachability information to be encoded. A reachability query then reduces to a single membership test—checking whether the target's label falls within one of the source's intervals. The trade-off between query latency and index size is governed by the target group size: smaller target groups yield faster query processing but require larger index space. This partitioned design allows PTIL to flexibly adapt to restricted-domain queries and achieve controllable space–time trade-offs. To further accommodate real-world dynamics, we also design four efficient update primitives to support incremental changes to V_s and V_t without rebuilding the index.

Based on this, we implement two specific modes: a *grouped mode* (**PTIL-g**) and a *probability-aware mode* (**PTIL-p**). In the grouped mode (**PTIL-g**), source vertices are treated as one group, while target vertices are partitioned into a chosen number of groups within a restricted-domain query space (e.g., $V_1 \times V_2$). A separate PTIL index is constructed for each target group over the corresponding query domain. At query time, each query is routed to exactly one index according to the target's group, achieving balanced query efficiency and controllable index size. The full $V \times V$ setting is naturally supported as a special case where $V_1 = V_2 = V$.

In the probability-aware mode (**PTIL-p**), vertices are further stratified into *hot sources*, *hot targets*, and *normal vertices* according to their query frequency in the workload. **PTIL-p** builds three layers of indices with different priorities: (1) queries from any source to hot targets use fine-grained groups to minimize latency; (2) queries from hot sources to normal targets also use fine-grained groups on the reverse graph for fast response; and (3) queries between normal vertices adopt coarse-grained grouping to reduce index space. This multi-layered design enables **PTIL-p** to balance performance and storage under skewed workloads.

- **Native Support for Restricted-Domain Reachability Queries with Manageable Space–Time Trade-off.** PTIL is the first reachability index explicitly designed for restricted query domains such as $V_1 \times V_2$. Rather than building a full global $V \times V$ index, PTIL materializes only the reachability information that lies inside the queried subdomain, so storage is not spent on parts of the graph the workload never touches. Within this design, the target group size acts as a single tunable knob for the space–time trade-off: smaller groups give faster queries, while larger groups give a smaller index.
- **Dynamic Update Support.** Beyond static workloads, PTIL admits incremental maintenance under endpoint-domain (source and target) changes without full index reconstruction. We design four primitives—V_s-insert, V_s-delete, V_t-insert, and V_t-delete—that locally update only the source intervals and target

timestamps affected by each operation, adapting PTIL to evolving query workloads with minimal overhead and enabling practical deployment in dynamic systems.
- **Extensive Experimental Evaluation.** Across diverse real and synthetic datasets, with graphs reaching tens of millions of nodes and hundreds of millions of edges, **PTIL-g** achieves an **order-of-magnitude** reduction in query time compared to prior baselines, while offering a **near-linear and predictable** trade-off between index size and query latency. Under workload skewness, **PTIL-p** outperforms **PTIL-g** at comparable index sizes, achieving up to an **order-of-magnitude** speedup.

Section 2 describes Related Work. Section 3 reviews preliminaries. Section 4 details PTIL construction, querying, correctness, and complexity. Section 5 introduces PTIL-Compose with **PTIL-g** and **PTIL-p** routing. Section 6 presents dynamic updates between groups. Section 7 reports empirical results. Section 8 presents the case study. Section 9 concludes.

2 Related Work

Graph reachability queries have been studied for over three decades, producing many indexing and online techniques [12, 41, 44]. Existing indexing methods can be divided into *Label-Only* and *Label+Graph* approaches. Label-only methods answer queries using constructed labels, whereas Label+Graph methods use labels for pruning or partial answering and fall back to graph traversal when needed.

2-hop labeling encodes reachability using two label sets, $L_{out}(v)$ and $L_{in}(v)$, and answers queries by set intersection [7]. Representative methods include *HOPI* [25], *3-Hop* [16], *PathHop* [3], and *PPL* [39], which improve compression through path or chain decompositions. *TF-Label* [6] further reduces label size using topological folding. Parallel PLL variants such as BVC-PLL [14] reduce preprocessing time through parallel construction, which is complementary to PTIL's restricted-domain indexing. *TOL* [45] generalizes 2-hop labeling with a vertex order and Butterfly index, supporting efficient maintenance in dynamic graphs. These label-only methods provide fast queries, but their labels are often expensive to construct and offer limited parameterized control.

Another line compresses the transitive closure matrix. Exact transitive closure supports $O(1)$ queries but requires $O(n^2)$ space. Compressed methods reduce space at the cost of approximate or incomplete labels. *BFL* [30] uses Bloom filters for probabilistic pruning, while *Independent Permutation Labeling* (IP) [36] uses k-min-wise permutations and supports static and dynamic maintenance. These methods are typically Label+Graph approaches: although their indexes are compact, unresolved cases still require graph traversal, which can be expensive.

Tree-cover methods assign positions to vertices in a global order and store intervals to encode reachability, reducing queries to interval membership tests. Early methods include *Tree-Cover* [1] and *Tree+SSPI* [33], which combine interval encodings with auxiliary structures for non-tree edges. *Dual Labeling* [35] integrates spanning-tree intervals with condensed closure of non-tree edges. *GRAIL* [43] uses randomized DFS traversals to generate multiple interval labels, while *Ferrari* [28] bounds the number of intervals to trade index size for query precision. Unlike classical tree-cover methods, PTIL does not build a global interval index for all vertices; it timestamps only target vertices and materializes interval summaries only for admissible sources, aligning the index with the restricted query domain.

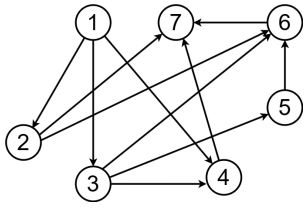


Figure 1: Example DAG

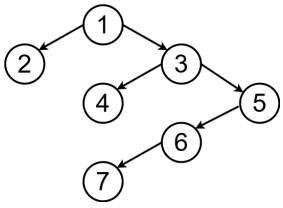


Figure 2: A Spanning Tree

Beyond these frameworks, *FELINE* [34] maps vertices into a two-dimensional space for fast non-reachability pruning, and *PReaCH* [21] combines contraction hierarchies with pruning heuristics. For dynamic graphs, *DAGGER* [40] extends interval labeling to dynamic SCC DAGs, *TOL* [45] supports insertions and deletions under a total-order framework, *IP* [36] provides randomized label maintenance, and *DBL* [20] combines Dynamic Landmark and Bidirectional Leaf labels. Purely online methods such as *ARROW* [27] and *IFCA* [22] avoid preprocessing, but generally trade query efficiency or exactness for adaptability.

In summary, existing methods mainly optimize global reachability over $V \times V$. In contrast, PTIL targets restricted-domain reachability, where only queries over structured endpoint domains are materialized and accelerated.

3 Preliminary

We consider a finite, simple directed graph $G = (V, E)$, where V is the vertex set and $E \subseteq V \times V$ is the set of directed edges. Let $n = |V|$ and $m = |E|$. A vertex with in-degree zero is a *source*, and a vertex with out-degree zero is a *sink*.

Definition 3.1 (Reachability). A sequence of vertices $s = v_0 \to v_1 \to \cdots \to v_k = t$ with $k \geq 0$ is an s-t path if $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \ldots, k-1\}$. We write $s \rightsquigarrow t$ if there exists at least one s-t path in G.

Definition 3.2 (Reachability Query). Given $u, v \in V$, a reachability query asks whether $u \rightsquigarrow v$. We define $\textsc{Query}(u, v) = \texttt{true}$ if $u \rightsquigarrow v$, and $\textsc{Query}(u, v) = \texttt{false}$ otherwise.

A query is *positive* if its output is `true`, and *negative* otherwise.

Example 3.3. In Figure 1, $\textsc{Query}(1, 3) = \texttt{true}$, $\textsc{Query}(3, 1) = \texttt{false}$, and $\textsc{Query}(1, 7) = \texttt{true}$.

Definition 3.4 (Directed Acyclic Graph). A directed acyclic graph (DAG) is a directed graph with no directed cycle; that is, there is no directed path that starts and ends at the same vertex.

In a DAG, the edge orientation enforces a strict partial order on the vertices, precluding any cyclic traversal.

Definition 3.5 (Spanning Tree). Let $G = (V, E)$ be an undirected graph. A spanning tree is a subgraph $T = (V, E_T)$ such that: (i) T is a tree (connected and acyclic); (ii) T spans G (its vertex set is V); (iii) $E_T \subseteq E$; and (iv) $|E_T| = |V| - 1$.

Definition 3.6 (Tree vs. Non-Tree Edge). Given a DAG $G = (V, E)$ and a spanning tree $T = (V, E_T)$ of its undirected graph, an edge $(u, v) \in E_T$ is a *tree edge*; any $(u, v) \in E \setminus E_T$ is a *non-tree edge*.

Example 3.7. A valid spanning tree of the graph in Figure 1 is shown in Figure 2. In this example, $(1, 3)$ is a tree edge, whereas $(2, 7)$ is a non-tree edge.

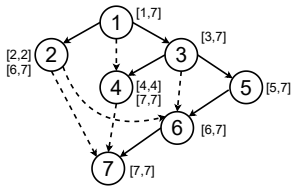


Figure 3: An Example of Tree Cover

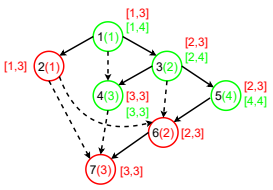


Figure 4: An Example of PTIL

Definition 3.8 (Topological Sorting [18]). For a DAG $G = (V, E)$, a topological ordering is a linear ordering $(v_1, \ldots, v_n)$ of V such that for every edge $(u, v) \in E$, u appears before v in the sequence.

Example 3.9. In Figure 1, valid topological orderings include $[1, 2, 3, 5, 6, 7, 4]$ and $[1, 3, 4, 2, 5, 6, 7]$.

Definition 3.10 (Topological Level [6]). For a DAG $G = (V, E)$, the topological level of v is $\ell(v) = 1$ if $\Gamma^-(v) = \emptyset$, and $\ell(v) = 1 + \max_{(u,v)\in E} \ell(u)$ otherwise. The i-th level is $L_i = \{v \in V \mid \ell(v) = i\}$.

Example 3.11. In Figure 1, the levels are $L_1 = \{1\}$, $L_2 = \{2, 3\}$, $L_3 = \{4, 5\}$, $L_4 = \{6\}$, and $L_5 = \{7\}$.

Definition 3.12 (DFS Postorder Timestamps). Given a directed graph $G = (V, E)$, DFS assigns each vertex v a discovery time $\text{tin}(v)$ and a finishing time $\text{tout}(v)$, where $\text{tout}(v)$ is recorded after all descendants of v have been explored. The mapping $v \mapsto \text{tout}(v)$ is called *DFS postorder timestamping*.

For a DAG, nonincreasing $\text{tout}(\cdot)$ is consistent with a topological order, and the DFS parent pointers form a spanning forest usable as the tree-cover backbone.

Tree Cover. A tree cover uses a *spanning tree* as the backbone and encodes reachability with DFS-based intervals. For each vertex v, DFS assigns $\text{tin}(v)$ and $\text{tout}(v)$, where $[\text{tin}(v), \text{tout}(v)]$ represents the subtree of v. Tree edges are captured by interval nesting, while non-tree edges are handled by propagating and normalizing successor interval sets. Each vertex therefore stores a disjoint interval family summarizing its reachable vertices. A reachability query $Query(s, t)$ checks whether $\text{tout}(t)$ lies in the interval family of s. Fig. 3 shows an example tree cover for Fig. 1.

4 Partitioned Tree-cover Interval Labeling

In many practical settings, reachability queries are source-oriented within a constrained query domain: the task is to determine which targets are reachable from a source. Unlike 2-hop and transitive-closure (TC) labeling schemes, which typically inspect labels on the source u and target v, Tree-Cover methods answer queries using only the source-side index. This model naturally fits restricted-domain workloads, where a query is resolved by testing whether the target falls within the source's reachable target intervals. Motivated by this, PTIL adopts a partitioned design that encodes source-to-target reachability within the structured query domain. PTIL is not a simple subset of classical tree-cover labeling; it changes the indexed object from global vertex-to-vertex reachability to target timestamps reachable from admissible sources. Thus, PTIL is a restricted-domain-aware indexing framework rather than a pruning of a global tree-cover index.

4.1 Problem statement

Let $G = (V, E)$ be a DAG (Directed Acyclic Graph). Given two subsets V_1 (targets) and V_2 (sources), we consider the *partitioned subdomain*

Algorithm 1 PTIL_Construct$(G = (V, E), V_1, V_2)$

Require: DAG $G = (V, E)$, targets $V_1 \subseteq V$, sources $V_2 \subseteq V$
Ensure: $\{\text{tout}[v] \mid v \in V_1\}, \{L[u] \mid u \in V_2\}$

$\pi \leftarrow$ TopologicalSort(G)
parent $\leftarrow$ BuildSpanningTree(G, π)
for each $v \in V$ **do** $\text{tin}[v] \leftarrow \perp$; $\text{tout}[v] \leftarrow \perp$
$post \leftarrow 0$
for each root r in parent **do**
  DFS_Label(r)
end for
function DFS_Label(x)
  for each c in ChildrenInTree(x) **do**
    DFS_Label(c)
  end for
  if $x \in V_1$ **then**
    $\text{tout}[x] \leftarrow post$; $post \leftarrow post + 1$
  end if
end function
for each $v \in V$ **do** $L[v] \leftarrow \varnothing$
for v in Reverse(π) **do**
  $S \leftarrow \varnothing$
  for each edge $(v \to w) \in E$ **do**
    $S \leftarrow S \cup L[w]$
  end for
  if $v \in V_1$ **then**
    $S \leftarrow S \cup \{[\text{tout}[v], \text{tout}[v]]\}$
  end if
  $L[v] \leftarrow$ NormalizeIntervals(S) ▷ sort + merge
end for
for each $v \in V$ **do**
  if $v \notin V_2$ **then**
    $L[v] \leftarrow \varnothing$ ▷ materialize only on sources
  end if
end for
return $\{\text{tout}[v] \mid v \in V_1\}, \{L[u] \mid u \in V_2\}$

$$\mathcal{D}(V_1, V_2) := V_2 \times V_1,$$

and aim to materialize *only* the information necessary to answer reachability on $\mathcal{D}(V_1, V_2)$, while enabling explicit space–time trade-offs. This *partitioned* capability allows storing *partial* reachability for the graph in an exact, partitioned manner.

4.2 PTIL(G, V_1, V_2)

Method Overview. We build a partitioned index for queries (u, v) with $u \in V_2$ and $v \in V_1$ in three steps: (i) run a DFS on a tree cover and assign timestamps *only* to targets V_1; (ii) for each $u \in V_2$, aggregate the interval sets of its out-neighbors (and, when applicable, a target's self-interval) and normalize into a *sorted, disjoint* interval list $L(u)$; (iii) answer by membership on the time axis: return true iff $\text{tout}(v) \in L(u)$.

Example. In Figure 4, the red vertices form $V_1 = \{2, 6, 7\}$, and $V_2 = V$. PTIL timestamps only targets: $\text{tout}(2) = 1$, $\text{tout}(6) = 2$, and $\text{tout}(7) = 3$; non-target vertices receive no timestamps. After reverse-topological propagation and normalization, labels summarize reachable targets, e.g., $L(1) = [1, 3]$, $L(3) = [2, 3]$, $L(4) = [3, 3]$, and $L(5) = [2, 3]$.

Queries reduce to membership tests: $Q(1, 6)$ is true because $\text{tout}(6) = 2 \in [1, 3]$, while $Q(3, 2)$ is false because $\text{tout}(2) = 1 \notin [2, 3]$. Compared with classical tree cover, PTIL keeps only target

Algorithm 2 BUILD_GROUPED

Require: DAG $G = (V, E)$, source set V_1, target set V_2, number of target groups k

1: $\{T_1, T_2, \ldots, T_k\} \leftarrow \text{RANDOMPARTITION}(V_2, k)$
2: **for** each $v \in V_2$ **do**
3: group_of_target$[v] \leftarrow i$ such that $v \in T_i$
4: **end for**
5: **for** $i \in \{1, \ldots, k\}$ **do**
6: $(\text{Label}_i, \text{Tout}_i) \leftarrow \text{PTIL\_CONSTRUCT}(G, T_i, V_1)$
7: IDX_T$[i]$.label $\leftarrow$ Label$_i$ ▷ IDX_T$[i]$.label$[u]$, $u \in V_1$
8: IDX_T$[i]$.tout $\leftarrow$ Tout$_i$ ▷ IDX_T$[i]$.tout$[v]$, $v \in T_i$
9: **end for**

Algorithm 3 QUERY_GROUPED

Require: Source $u \in V_1$, target $v \in V_2$

1: $i \leftarrow$ group_of_target$[v]$ ▷ v's target group
2: $L_u \leftarrow$ IDX_T$[i]$.label$[u]$ ▷ interval list of u for group i
3: $t_v \leftarrow$ IDX_T$[i]$.tout$[v]$ ▷ tout of v in group i
4: **return** $\text{EXISTINGCHECK}(L_u, t_v)$

timestamps and source-side intervals for $V_2 \times V_1$, avoiding global labels and out-of-domain information.

4.3 Index Construction

Phase I: Spanning tree. Compute a topological ordering π of G. For each vertex, choose as its tree parent the in-neighbor with the maximum topological rank (roots have no parent). This yields a forest serving as the DFS backbone.

Phase II: Timestamps on V_1. Run DFS over the tree cover and assign timestamp tout(v) *only* for $v \in V_1$ (in practice, queries only require tout). Vertices in $V \setminus V_1$ receive invalid timestamps.

Phase III: Global propagation. Process vertices in reverse order of π. For each v, let S be the multiset union of $L(w)$ over all out-neighbors w of v; if $v \in V_1$, also add the singleton interval $[\text{tout}(v), \text{tout}(v)]$ to S. Normalize S by sorting and merging overlapping and adjacent intervals to obtain a canonical, disjoint union $L(v)$. After the sweep finishes, we *retain* $L(u)$ only for $u \in V_2$; the other $L(\cdot)$ are discarded as construction intermediates. The final index thus consists of $\{\text{tout}(v) \mid v \in V_1\}$ and $\{L(u) \mid u \in V_2\}$.

The complete procedure is shown in detail in Algorithm 1. *Lines 1–2:* topological ordering and build spanning tree. *Lines 3–15:* initialize and assign timestamps. *Lines 16:* initialize interval sets of vertices. *Lines 17–26:* reverse-topological reduce: aggregate intervals of successors and add the self-interval for $v \in V_1$. *Lines 27–31:* partitioned materialization: keep $L(\cdot)$ only at V_2.

4.4 Query Answering

Given $u \in V_2$ and $v \in V_1$, return

$$Q(u, v) \equiv \big[\, \text{tout}(v) \in L(u) \,\big].$$

4.5 Correctness

For a vertex $x \in V$, define the *target time-set*

$$T(x) \stackrel{\text{def}}{=} \{\, \text{tout}(t) \;\big|\; t \in V_1,\ x \rightsquigarrow t \,\},$$

i.e., the multiset of DFS postorder timestamps of targets $t \in V_1$ reachable from x. For a set $S \subseteq \mathbb{Z}$, let $\text{IC}(S)$ denote the *minimal closed-interval cover* of S on the time axis (adjacent endpoints may be merged). The normalization operator $\text{canon}(\cdot)$, as used by

Algorithm 4 BUILD_PROBABILITY_AWARE

Require: DAG $G = (V, E)$; hot-target: $\{V_{t1}, \ldots, V_{tk_1}\}$ with $\bigcup_i V_{ti} = V_{ht}$; hot-source: $\{V_{s1}, \ldots, V_{sk_2}\}$ with $\bigcup_i V_{si} = V_{hs}$; normal: $\{V_{n1}, \ldots, V_{nk_3}\}$ with $\bigcup_i V_{ni} = V_n$

1: **for** each $v \in \bigcup_{i=1}^{k_1} V_{ti}$ **do** ▷ group maps
2: tgt_hot_group$[v] \leftarrow i$
3: **end for**
4: **for** each $u \in \bigcup_{i=1}^{k_2} V_{si}$ **do**
5: src_hot_group$[u] \leftarrow i$
6: **end for**
7: **for** each $v \in \bigcup_{i=1}^{k_3} V_{ni}$ **do**
8: tgt_norm_group$[v] \leftarrow i$
9: **end for**
10: **for** $i \leftarrow 1$ **to** k_1 **do** ▷ $\mathbf{L_T}$: (∗, hot) on G
11: $(\text{Label}_{T,i}, \text{Tout}_{T,i}) \leftarrow \text{PTIL\_CONSTRUCT}(G, V_{ti}, V)$
12: $L_T[i]$.label $\leftarrow$ Label$_{T,i}$
13: $L_T[i]$.tout $\leftarrow$ Tout$_{T,i}$
14: **end for**
15: $V_n^{\text{all}} \leftarrow \bigcup_{i=1}^{k_3} V_{ni}$
16: $G_r \leftarrow \text{REVERSEGRAPH}(G)$
17: **for** $i \leftarrow 1$ **to** k_2 **do** ▷ $\mathbf{L_S}$: (hot, n) on G_r
18: $(\text{Label}_{S,i}, \text{Tout}_{S,i}) \leftarrow \text{PTIL\_CONSTRUCT}(G_r, V_{si}, V_n^{\text{all}})$
19: $L_S[i]$.label $\leftarrow$ Label$_{S,i}$
20: $L_S[i]$.tout $\leftarrow$ Tout$_{S,i}$
21: **end for**
22: **for** $i \leftarrow 1$ **to** k_3 **do** ▷ $\mathbf{L_N}$: (n, n) on G
23: $(\text{Label}_{N,i}, \text{Tout}_{N,i}) \leftarrow \text{PTIL\_CONSTRUCT}(G, V_{ni}, V_n^{\text{all}})$
24: $L_N[i]$.label $\leftarrow$ Label$_{N,i}$
25: $L_N[i]$.tout $\leftarrow$ Tout$_{N,i}$
26: **end for**

PTIL, refers to the standard *sort + merge-overlapping and adjacent* routine that outputs a *sorted, pairwise disjoint* interval family.

LEMMA 4.1. *During the reverse-topological construction, once vertex v has been reduced, the maintained set satisfies*

$$L(v) \;=\; \text{IC}\big(T(v)\big).$$

PROOF. Order vertices as $v_1, \ldots, v_{|V|}$ by the reverse of a topological ordering. Proceed by induction on this order.

Base case. If v is a sink, then $T(v) = \{\text{tout}(v)\}$ iff $v \in V_1$, and $T(v) = \varnothing$ otherwise. The algorithm adds $[\text{tout}(v), \text{tout}(v)]$ exactly when $v \in V_1$ and normalizes, yielding $\text{IC}(T(v))$.

Inductive step. Assume the claim holds for all processed out-neighbors w of v. Since

$$T(v) \;=\; \Big(\bigcup_{(v,w)\in E} T(w) \Big) \;\cup\; \big(\mathbf{1}_{v\in V_1} \cdot \{\text{tout}(v)\}\big),$$

and interval cover is closed under unions, we obtain

$$\text{IC}(T(v)) = \text{canon}\Big(\bigcup\nolimits_{(v,w)} \text{IC}(T(w)) \cup \mathbf{1}_{v\in V_1} \cdot [\text{tout}(v), \text{tout}(v)] \Big),$$

which is the update for $L(v)$. Hence $L(v) = \text{IC}(T(v))$. □

LEMMA 4.2. *For any multiset of closed intervals S, we have* $\text{canon}(S) \equiv \text{IC}(\mathcal{P}(S))$*, where $\mathcal{P}(S)$ is the union of all integer time points covered by intervals in S. Moreover, among all interval families covering $\mathcal{P}(S)$,* $\text{canon}(S)$ *has the minimum number of intervals.*

PROOF. Sorting and repeatedly merging overlapping and adjacent intervals preserves the covered point set $\mathcal{P}(S)$. The result is a disjoint family in which no two consecutive intervals can be merged; if there were a smaller cover, at least two intervals in the canonical family would be mergeable, a contradiction. □

Algorithm 5 Query_Probability_Aware

Require: Query pair (u, v)
if $v \in \bigcup_{i=1}^{k_1} V_{t_i}$ **then** ▷ use L_T; covers (hot,hot) and (n,hot)
  $i \leftarrow$ tgt_hot_group$[v]$
  $L_u \leftarrow L_T[i]$.label$[u]$
  $t_v \leftarrow L_T[i]$.tout$[v]$
  return ExistingCheck(L_u, t_v)
else if $u \in \bigcup_{i=1}^{k_2} V_{s_i}$ **then** ▷ use L_S; covers (hot,n)
  $i \leftarrow$ src_hot_group$[u]$
  $L_v \leftarrow L_S[i]$.label$[v]$ ▷ in G_r, v acts as a source
  $t_u^{(r)} \leftarrow L_S[i]$.tout$[u]$ ▷ tout of u in G_r
  return ExistingCheck$(L_v, t_u^{(r)})$
else ▷ use L_N; covers (n,n)
  $j \leftarrow$ tgt_norm_group$[v]$
  $L_u \leftarrow L_N[j]$.label$[u]$
  $t_v \leftarrow L_N[j]$.tout$[v]$
  return ExistingCheck(L_u, t_v)
end if

Lemma 4.3. *For any $v \in V_1$, adding $[\text{tout}(v), \text{tout}(v)]$ at v is equivalent to inserting the singleton point $\text{tout}(v)$ into $T(v)$.*

Proof. Because a degenerate closed interval covers exactly its endpoint and the normalization canon$(\cdot)$ (equivalently IC$(\cdot)$ by Lemma 4.2) preserves the covered point set, adding $[\text{tout}(v), \text{tout}(v)]$ contributes precisely the singleton $\{\text{tout}(v)\}$, which is the same as inserting $\text{tout}(v)$ into $T(v)$. □

Theorem 4.1. *For any $u \in V_2$ and $v \in V_1$, the final index answers*

$$Q(u, v) = \text{true} \iff \text{tout}(v) \in L(u).$$

Proof. ($\Rightarrow$) If $u \rightsquigarrow v$, then $\text{tout}(v) \in T(u)$. By Lemma 4.1, $L(u) = \text{IC}(T(u))$, which contains $\text{tout}(v)$.
($\Leftarrow$) If $\text{tout}(v) \in L(u)$, then by Lemmas 4.1 and 4.2, $\text{tout}(v) \in \text{IC}(T(u))$ implies $\text{tout}(v) \in T(u)$, hence $u \rightsquigarrow v$. □

Corollary 4.2. *Materializing only $\{L(u) \mid u \in V_2\}$ and $\{\text{tout}(v) \mid v \in V_1\}$ is sufficient and complete for all queries with $u \in V_2$ and $v \in V_1$, and makes no claim outside this partitioned scope.*

Proof. By Theorem 4.1, $Q(u, v) = \text{true} \iff \text{tout}(v) \in L(u)$. Our materialization policy stores exactly these quantities for $u \in V_2$ and $v \in V_1$, hence the index is sufficient and complete. □

4.6 Complexity

Space. The index consists of (i) timestamps and group IDs for targets in V_1, and (ii) interval lists for sources in V_2:

$$O\left(|V_1| + \sum_{u \in V_2} |L(u)|\right).$$

Construction time. Topological sorting and DFS cost $O(|V| + |E|)$. The reverse-topological reduction is dominated by multiway union and normalization:

$$O\left(|V| + |E| + \sum_{v \in V} \text{merge\_work}(v)\right),$$

where merge_work(v) scales with the total list size merged at v.
Query time. Membership testing with a sorted, disjoint interval list is $O(\log |L(u)|)$ via binary search.

5 PTIL-Compose: Group & Probability-Aware

PTIL's grouped construction supports not only partial reachability materialization over a restricted query domain, but also global reachability through composition of multiple PTIL indices; when query probabilities are known or can be estimated, a layered+grouped scheme further prioritizes storage and construction resources for high-probability (hot) vertices, thereby reducing average latency. We present two usages:

Grouped reachability: restricted-domain query space (e.g., $V_1 \times V_2$), partition the target set V_2 randomly and uniformly into k groups, and build one PTIL for each group over the corresponding query domain. During querying, route by the target's group to the appropriate index and perform a single membership test. By adjusting k, users can achieve a controllable trade-off between index size and query latency. The full-graph $V \times V$ case is naturally supported as a special case with $V_1 = V_2 = V$.

Probability-aware: according to query probabilities, partition the vertices into three categories—"hot sources / hot targets / normal"—and further refine them into $k_1/k_2/k_3$ groups respectively; construct three layers of indexes L_T, L_S, L_N, and route with the fixed priority "hot target first → hot source → normal residual."

5.1 Group Reachability

Grouping and construction. Partition the target set V_2 randomly and uniformly into k pairwise-disjoint groups:

$$V_2 = \bigcup_{i=1}^{k} T_i.$$

For each T_i as the target side, build an independent PTIL over the restricted-domain query space:

$$\text{PTIL}(G, T_1, V_1), \ \ldots, \ \text{PTIL}(G, T_k, V_1).$$

Each index records reachability information from the source set V_1 to the i-th target group T_i. Together, these k indices cover the restricted query domain $V_1 \times V_2$. The construction procedure is shown in Algorithm 2. Across target groups, the graph representation, topological order, and tree-cover backbone are shared, while timestamping and interval materialization are group-specific. Thus, sequential preprocessing grows roughly linearly with k, and group-level indexes can be built independently.
Query routing and explanation. For a query (u, v) with $u \in V_1$ and $v \in V_2$, determine the target group index i such that $v \in T_i$, and consult only the corresponding group-level index. Specifically, retrieve the interval list IDX_T$[i]$.label$[u]$ and the timestamp IDX_T$[i]$.tout$[v]$, and decide reachability by testing whether IDX_T$[i]$.tout$[v]$ falls within IDX_T$[i]$.label$[u]$. Owing to the independence of groups, each query requires accessing only a single index. The full procedure is given in Algorithm 3.

The full-graph $V \times V$ setting is naturally supported as a special case by letting $V_1 = V_2 = V$.
Example. In Figure 4, partition all vertices into two disjoint *target groups*: $V_1 = \{2, 6, 7\}$ (red) and $V_2' = \{1, 3, 4, 5\}$ (green). For each group, we build an independent PTIL on the whole graph G, yielding two group-level indexes $\text{IDX}_T[1]$(red) and $\text{IDX}_T[2]$(green).

Query by label: $Q(1, 6)$: $v = 6 \in V_1 \Rightarrow i = 1$. $\text{tout}(6) = 2 \in \text{IDX}_T[1].\text{label}[1] = [1, 3] \Rightarrow$ true. $Q(3, 2)$: $v = 2 \in V_1 \Rightarrow i = 1$. $\text{tout}(2) = 1 \notin \text{IDX}_T[1].\text{label}[3] = [2, 3] \Rightarrow$ false.
Correctness. We analyze the correctness below.

Lemma 5.1. *For any target group $T_i \subseteq V_2$, let*

$$I_i = \textit{PTIL\_Construct}(G, T_i, V_1).$$

Then, for any $u \in V_1$ and $v \in T_i$,

$$Q(u,v) = \mathit{true} \iff \mathrm{tout}_i(v) \in L_i(u).$$

Proof. Directly inherits the reasoning chain from Lemma 4.1, Lemma 4.2, and Theorem 4.1, restricted to the query domain $V_1 \times T_i$. □

Corollary 5.2. *Any $v \in V_2$ belongs to a unique target group T_j. Therefore, for any query (u,v) with $u \in V_1$ and $v \in V_2$, the answer is equivalent to testing* $\mathrm{tout}_j(v) \in L_j(u)$ *on* $\mathrm{PTIL}(G, T_j, V_1)$*, hence correct and complete over the restricted query space $V_1 \times V_2$.*

Proof. Follows immediately from pairwise-disjoint target groups whose union is V_2, together with Lemma 5.1. □

The full-graph $V \times V$ case is a special instance with $V_1 = V_2 = V$.

5.2 PTIL-Probability-Aware

Probability definitions. We assume vertex hotness is known or can be estimated from query logs or system statistics, following standard assumptions in workload-aware indexing. Given the probability distribution of queries, define three vertex categories:

- $\mathbf{V_{hs}}$ (High-probability Sources): vertices that have high probability to appear as the *source* u in (u,v);
- $\mathbf{V_{ht}}$ (High-probability Targets): vertices that have high probability to appear as the *target* v in (u,v);
- $\mathbf{V_n}$ (Neutral/Normal): vertices that belong to neither V_{hs} nor V_{ht} (i.e., relatively infrequent in queries).

Grouping and construction. For parallelism and resource budgeting, each category is further partitioned into mutually disjoint *groups*:

$$V_{ht} = \bigcup_{i=1}^{k_1} V_{ti}, \qquad V_{hs} = \bigcup_{i=1}^{k_2} V_{si}, \qquad V_n = \bigcup_{i=1}^{k_3} V_{ni}.$$

Three-layer design and routing priority:

- $\mathbf{L_T}$ (hot-target layer) — covers $(*, \mathrm{hot})$: all queries with $v \in V_{ht}$; constructed on the *original graph* G.
- $\mathbf{L_S}$ (hot-source layer, reverse graph) — covers $(\mathrm{hot}, \mathrm{n})$: queries with $u \in V_{hs}$ and $v \in V_n$; constructed on the *reverse graph* G_r.
- $\mathbf{L_N}$ (normal residual layer) — covers (n, n): both ends are normal; constructed on the *original graph* G.

Under full coverage of all (u,v) queries, allocate more construction and storage resources to V_{ht} and V_{hs} so as to shorten hot-query paths and reduce the number of intervals. The complete procedure is shown in detail in Algorithm 4.

Query routing and explanation. For any query (u,v), the procedure consists of two phases, selecting the layer and performing a single membership test on that layer: (1) If $v \in V_{ht}$: set $i = \mathrm{tgt\_hot\_group}[v]$ and decide reachability by testing membership of $L_T[i].\mathrm{tout}[v]$ in $L_T[i].\mathrm{label}[u]$; (2) Otherwise, if $u \in V_{hs}$: set $i = \mathrm{src\_hot\_group}[u]$ and decide reachability by testing membership of $L_S[i].\mathrm{tout}[u]$ in $L_S[i].\mathrm{label}[v]$. (3) Otherwise $(u, v \in V_n)$: set $j = \mathrm{tgt\_norm\_group}[v]$ and decide reachability by testing membership of $L_N[j].\mathrm{tout}[v]$ in $L_N[j].\mathrm{label}[u]$. Since these categories and their groups are pairwise disjoint and their unions cover all vertices, each (u,v) uniquely matches one layer and one group. The complete procedure is shown in Algorithm 5.

Correctness. We analyze the correctness below.

Lemma 5.3. *For any DAG G and its reverse graph G_r, we have*

$$u \rightsquigarrow_G v \iff v \rightsquigarrow_{G_r} u.$$

Algorithm 6 VtDelete(t): delete a target and maintain timestamps/labels

Require: $t \in V_t$
1: $\tau \leftarrow \mathrm{tout}(t)$
2: Remove t from V_t
3: **for all** $t' \in V_t$ with $\mathrm{tout}(t') > \tau$ **do**
4: $\quad \mathrm{tout}(t') \leftarrow \mathrm{tout}(t') - 1$
5: **end for**
6: **for all** $s \in V_s$ **do**
7: $\quad$ remove τ if $\tau \in L_s$;
8: $\quad$ shift all values $> \tau$ in L_s by -1; keep canonical
9: **end for**

Proof. Reversing all edges in a path yields a path in the opposite direction; thus, path existence is in bijection. □

Lemma 5.4. *For L_T and L_N (PTIL on G): for any group and any (u,v) in its domain,*

$$Q(u,v) = \mathit{true} \iff \mathrm{tout}_G(v) \in L_G(u).$$

For L_S (PTIL on G_r): for any group and any (u,v) with $u \in V_{si}$, $v \in V_n$,

$$Q(u,v) = \mathit{true} \iff \mathrm{tout}_{G_r}(u) \in L_{G_r}(v).$$

Proof. Apply Corollary 4.2 on G and G_r respectively, and combine with Lemma 5.3 for equivalence to the original query on G. □

Theorem 5.5. *Under the routing policy "first $L_T \to$ then $L_S \to$ finally L_N," every query (u,v) selects a* unique *layer and a* unique *group, and the returned decision is correct.*

Proof. The three cases form a partition of the query space: every query (u,v) falls into *exactly one* case and cannot belong to any other. Specifically, (i) $v \in V_{ht} \Rightarrow L_T[i]$; (ii) else if $u \in V_{hs} \wedge v \in V_n \Rightarrow L_S[i]$; (iii) else $u, v \in V_n \Rightarrow L_N[j]$. In-layer correctness follows from Lemma 5.4; reverse-graph consistency follows from Lemma 5.3. □

Corollary 5.6. *The union of the three layers yields a complete coverage of the entire query space; each layer/group assigns timestamps and materializes labels strictly within its own domain (V_1, V_2) (on G or G_r respectively), hence there is no cross-group leakage.*

Proof. Immediate from Theorem 5.5 and Lemma 5.4. □

6 Dynamic Updates

Real query workloads are rarely static: V_s and V_t evolve with roles, e.g., newly flagged accounts become *sources* and monitored entities become *targets*, while G may remain unchanged. This section formalizes *dynamic workload updates* on endpoint domains and primitive operations that preserve PTIL's correctness over $V_s \times V_t$, focusing on changes to V_s and V_t rather than graph-topology updates.

6.1 Problem statement

Let $G = (V, E)$ be a fixed directed graph. At any time, the meaningful query space is restricted to $V_s \times V_t$, where $V_s \subseteq V$ is the current *source domain* and $V_t \subseteq V$ is the current *target domain*.

We consider four update types that modify the membership of V_s or V_t (graph edges are unchanged): *(1) V_s-delete, (2) V_s-insert,*

Algorithm 7 VtInsert(t): insert a target with timestamp shifting and selective label updates

Require: $t \notin V_t$, current V_s, V_t and labels $\{L_s\}$
$\tau = \min\{\text{tout}(x) \mid t \rightsquigarrow x,\ x \in V_t\}$; if none, $\tau = |V_t|$
for all $x \in V_t$ with $\text{tout}(x) \geq \tau$ **do**
  $\text{tout}(x) \leftarrow \text{tout}(x) + 1$
end for
Add t into V_t and set $\text{tout}(t) \leftarrow \tau$
for all $s \in V_s$ **do**
  shift all values $\geq \tau$ in L_s by +1; keep canonical
end for
$A \leftarrow$ get all sources that can reach t by bfs
for all $s \in A$ **do**
  Insert point τ into L_s
  merge with neighbors if adjacent/overlapping
end for

(3) V_t-delete, and *(4) V_t-insert*. Each update must preserve the invariant:

$$\forall s \in V_s,\ \forall t \in V_t: \quad s \rightsquigarrow t \iff \text{tout}(t) \in L_s,$$

where $s \rightsquigarrow t$ denotes reachability in G.

6.2 V_s-delete

Deleting a source only affects its own label; other sources and all targets remain unchanged. Since PTIL answers only queries whose source lies in V_s, removing s requires no relabeling elsewhere. Correctness is immediate.

6.3 V_s-insert

When a new vertex s is added into the source set V_s, our goal is to construct its source-side label $I[s]$ without rebuilding any existing labels. We perform a forward traversal from s over outgoing edges. Whenever the traversal visits a vertex $u \in V_t$, we insert its target timestamp $ts[u]$ into $I[s]$ (as a point) and maintain $I[s]$ as a merged interval set. This produces the exact set of reachable targets restricted to V_t for the newly inserted source.

Pruning via existing sources. When inserting a new source, PTIL performs a forward traversal to collect reachable targets. If the traversal reaches an existing source, its precomputed label is reused and expansion stops, avoiding redundant exploration.

6.4 V_t-delete

Deleting a target removes its timestamp, so PTIL recompacts the target timestamp space and shifts source labels consistently.

Algorithm 6 records $\tau = \text{tout}(t)$, removes t from V_t, and shifts every remaining target timestamp larger than τ left by one. Since source labels share the same timestamp space, DeletePointAndShift(L_s, τ) removes τ, shifts larger interval endpoints, and normalizes each affected label.

6.5 V_t-insert

Inserting a target adds a new timestamp, so PTIL must update the timestamp space and then add the new point only to sources that can reach it.

Algorithm 7 first chooses the insertion position τ as the minimum timestamp among existing targets reachable from t; if none exists, t is appended with $\tau = |V_t|$. All existing target timestamps $\geq \tau$ are shifted right by one, and t is assigned $\text{tout}(t) = \tau$. Since source labels use the same timestamp space, all labels are shifted.

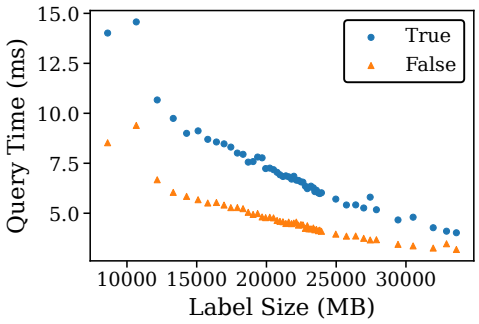


Figure 5: Query Time vs. Label Size on cit-Patents with Different Numbers of Groups.

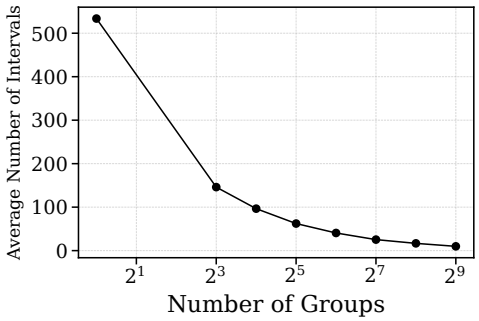


Figure 6: Average Number of Intervals per Query on cit-Patents with Different Numbers of Groups.

Table 1: Statistics of datasets used for experiments

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert E\rvert/\lvert V\rvert$
arXiv	6,000	66,707	11.12
yago	6,642	42,392	6.38
pubmed	9,000	40,028	4.45
citeseer	10,720	44,258	4.13
BerkStan	685,230	3,860,581	5.63
cit-Patents	3,774,768	16,518,947	4.38
citeseerx	6,540,401	15,011,260	2.30
go_uniprot	6,967,956	34,769,339	4.99
rand1m2x	1,000,000	1,999,997	2.00
rand1m4x	1,000,000	3,999,988	4.00
rand1m6x	1,000,000	5,999,959	6.00
rand1m8x	1,000,000	7,999,936	8.00
kron20_2x	1,048,576	2,640,621	2.52
kron20_4x	1,048,576	5,216,670	4.97
kron20_6x	1,048,576	7,745,956	7.39
kron20_8x	1,048,576	10,237,672	9.76
kron24_2x	16,777,216	40,610,242	2.42
kron24_4x	16,777,216	80,777,282	4.81
kron24_6x	16,777,216	120,613,664	7.19
kron24_8x	16,777,216	160,171,001	9.55

Finally, a reverse traversal from t identifies affected sources $A \subseteq V_s$; for each $s \in A$, PTIL inserts τ into L_s and merges adjacent or overlapping intervals. Together, these primitives maintain endpoint-domain changes without full reconstruction.

Table 2: Query Time (ms) on Graphs for Positive Queries.

	label+Graph			label-only				
Data	Fer	BFL	IP	PLL	PPL	TOL-l	TOL-u	PTIL-gg
arXiv	23.0	94.9	204.8	5.1	4.2	5.9	5.3	**4.0**
yago	23.6	8.6	14.8	3.7	4.5	4.8	4.7	**2.8**
pubmed	14.0	28.2	38.0	5.8	7.2	5.5	6.0	**3.1**
citeseer	9.9	21.5	36.2	5.4	7.1	5.1	5.6	**3.1**
BerkStan	120.4	202.0	1757.1	25.4	16.4	15.6	15.0	**3.3**
cit-Patents	4071.3	689.5	1791.2	76.9	65.0	49.5	59.3	**4.3**
citeseerx	34.4	171.8	86.8	19.1	18.8	20.3	22.2	**3.4**
go_uniprot	56.9	50.8	42.1	21.3	21.1	831.6	37.6	**2.9**
rand1m2x	23.1	46.2	73.5	28.8	71.6	14.0	12.4	**3.0**
rand1m4x	693.2	252.9	396.4	57.2	55.1	36.1	29.8	**3.4**
rand1m6x	11245.9	3649.3	6083.1	154.3	230.0	134.3	95.4	**6.8**
rand1m8x	80516.5	15130.4	29615.0	378.7	456.4	274.0	232.4	**10.6**
kron20_2x	706.0	3642.0	4038.8	26.5	10.9	64.1	17.6	**4.2**
kron20_4x	905.9	5243.9	4804.7	31.5	13.3	155.6	17.9	**4.0**
kron20_6x	941.8	5973.5	5691.0	21.0	12.0	75.6	22.7	**3.9**
kron20_8x	819.7	5203.3	5843.3	26.8	12.0	79.0	29.2	**3.9**
kron24_2x	11671.5	45093.2	60070.6	49.9	20.4	—	75.1	**7.1**
kron24_4x	15930.9	53918.8	63981.1	36.0	19.6	—	—	**6.5**
kron24_6x	19046.3	60918.6	74998.8	49.2	24.1	—	—	**6.5**
kron24_8x	14275.4	60283.7	6823.5	43.2	24.3	—	—	**7.3**

Table 3: Query Time (ms) on Graphs for Negative Queries.

	label+Graph			label-only				
Data	Fer	BFL	IP	PLL	PPL	TOL-l	TOL-u	PTIL-gg
arXiv	8.5	7.2	18.0	4.5	5.0	7.2	5.8	**4.5**
yago	8.5	2.3	3.0	3.2	3.2	3.1	3.0	**2.1**
pubmed	4.6	2.7	3.3	4.4	4.6	3.6	3.5	**2.4**
citeseer	5.1	2.5	4.3	4.5	4.9	4.2	4.3	**2.4**
BerkStan	17.0	5.7	29.4	16.4	13.1	12.2	11.8	**3.6**
cit-Patents	619.5	35.6	150.1	45.0	28.2	25.8	33.2	**3.4**
citeseerx	6.8	3.5	5.1	34.0	11.5	9.0	9.4	**2.2**
go_uniprot	18.2	2.4	1.8	18.4	18.1	14.6	11.4	**2.3**
rand1m2x	15.4	2.4	12.2	15.8	19.6	9.3	8.0	**2.1**
rand1m4x	344.9	32.4	113.9	25.5	28.7	16.8	14.4	**3.0**
rand1m6x	4745.8	848.8	1321.6	57.1	86.0	38.1	25.5	**7.7**
rand1m8x	25092.0	2990.6	4247.0	149.2	177.7	69.7	46.6	**12.2**
kron20_2x	6.3	7.6	8.2	10.0	8.1	32.9	7.5	**2.7**
kron20_4x	6.7	9.6	10.0	12.2	8.2	82.9	8.9	**2.8**
kron20_6x	6.7	15.3	12.8	10.9	10.4	45.7	9.9	**2.9**
kron20_8x	7.0	14.7	11.3	12.7	9.9	48.1	10.2	**2.9**
kron24_2x	14.9	15.5	36.4	78.9	34.9	—	25.7	**2.9**
kron24_4x	136.0	15.3	52.1	14.9	18.8	—	—	**2.7**
kron24_6x	19.9	37.0	28.7	21.0	96.2	—	—	**2.8**
kron24_8x	15.1	40.7	21.8	19.3	93.9	—	—	**3.5**

7 Experimental evaluation

This section reports our experimental results and evaluates the efficiency of PTIL on eight real graphs from [19, 43]. These datasets include four small and four large graphs, each with average degree greater than two. We also include four Erdős–Rényi graphs and eight Kronecker graphs, standard reproducible benchmarks that can be generated at arbitrary scales. These two synthetic families represent complementary extremes: a structureless baseline and a structured real-world-like network, enabling controlled evaluation of reachability indexing in correctness, efficiency, and robustness across diverse graph structures. Table 1 reports the real datasets, including the number of vertices $|V|$, the number of edges $|E|$, and the average degree $|E|/|V|$.

We denote the Group reachability variant of PTIL as **PTIL-g**, the Global Group reachability variant as **PTIL-gg**, and the probability-aware variant as **PTIL-p**; "query time" denotes the total time for processing 100,000 queries. For baselines, we prioritize well-established methods from leading database and data-management venues with open or verifiable implementations. Although several recent works outside these venues propose alternative techniques, their source code is unavailable or insufficiently documented, making faithful reimplementation prone to methodological bias. We therefore restrict comparisons to baselines with auditable artifacts and specified experimental settings.

PTIL-g is compared with state-of-the-art reachability algorithms, including Ferrari [28], IP [36], BFL [30], PLL [39], PPL [39], TOL-lower [45], and TOL-upper [45]. These baselines cover the main reachability-indexing frameworks identified in prior surveys and overviews: transitive-closure-based methods, 2-hop labeling, and tree-cover-based methods [41, 44]. PPL and TOL represent 2-hop methods, Ferrari represents tree-cover-based methods, and BFL and IP represent transitive-closure-based methods, covering both label-only and label+graph categories.

The source codes of these algorithms were kindly provided by the authors. All algorithms, including PTIL, are implemented in C++ and compiled using g++ with `-std=c++17` and `-O2`. Most Label+Graph approaches have parameters controlling index size. We set $k = 5$ for Ferrari and compute $s = 32$ seeds as an additional seed-pruning index, following the original paper. For IP, we set $k = 5$ and $h = 5$. For BFL, we set $s = 32k$ and $d = 10s$ for fairness. All experiments are conducted on a server with 2× Intel® Xeon® Gold 6342 CPUs at 2.80 GHz, 96 logical cores, 48 physical cores, and 503 GB RAM, running Ubuntu 22.04.1 LTS.

Table 4: Query Time (ms) on cit-Patents for Positive Queries with different number of reachable nodes.

	label+Graph			label-only				
Reachable	Fer	BFL	IP	PLL	PPL	TOL-l	TOL-u	PTIL-gg
1–10	12.4	9.1	14.8	39.4	42.2	8.2	8.4	**3.3**
10–100	60.2	17.7	27.4	43.5	44.9	11.8	14.6	**3.3**
100–1K	72.2	81.3	101.8	51.8	57.6	24.2	31.7	**4.0**
1K–10K	699.6	1866.8	1420.8	75.2	88.3	46.5	55.5	**6.9**
10K–100K	15910.7	26494.5	15211.2	106.9	140.0	97.6	83.8	**11.7**
100K+	119175.6	125548.9	55381.3	75.0	102.8	206.0	125.3	**15.3**
Growth	9642	13850	3742	1.9	2.4	25.2	15.0	**4.7**

For each dataset, the same query files are reused across all methods. Full-graph queries are sampled from $V \times V$, restricted-domain queries from $V_s \times V_t$, and each query-time result is measured over 100,000 queries.

Trading Space for Time: Faster PTIL-g via Larger Labels. This experiment evaluates PTIL-g's space–time trade-off by jointly considering grouping granularity, label size, interval count, and query latency. Using Cit-Patents as a representative large graph, we vary the number of groups from 1 to 512. Fig. 5 shows that larger label size reduces query time for both true and false queries. Fig. 6 further explains this trend: increasing the number of groups reduces the average number of intervals checked per query. Thus, finer grouping creates shorter source-side interval lists, enabling faster membership tests at the cost of larger overall labels. In summary, **PTIL-g** provides a clear space–time trade-off by using larger labels to obtain lower query latency.

Large-Margin Superiority in Query Efficiency. As summarized in Fig. 5 and Fig. 6, we report query times on whole graphs in two tables: the positive (reachable) query in Table 2 and the negative (unreachable) query in Table 3. In both tables, **PTIL-gg** is reported using a single representative grouping size k per graph, selected from the coarse grid $\{1, 8, 16, 32, 64, 128, 256, 512\}$

Table 5: Query Time (ms) on cit-Patents for Negative Queries with different number of reachable nodes.

	label+Graph			label-only				
Reachable Nodes	Fer	BFL	IP	PLL	PPL	TOL-l	TOL-u	PTIL-gg
1~10	8.0	1.5	4.2	12.2	12.6	6.9	5.9	**2.6**
10~100	10.4	3.9	8.8	18.8	18.4	10.1	15.2	**2.7**
100~1K	27.8	25.8	28.8	30.5	31.9	22.9	20.8	**3.8**
1K~10K	270.3	615.7	210.7	45.7	53.3	46.1	29.8	**7.7**
10K~100K	5095.6	9989.0	1389.4	82.6	92.1	104.2	49.7	**12.6**
100K+	48377.8	64517.9	6382.5	129.7	148.4	254.1	181.1	**16.0**
Growth Factor	6020	41700	1526	10.6	11.7	36.6	30.6	**6.0**

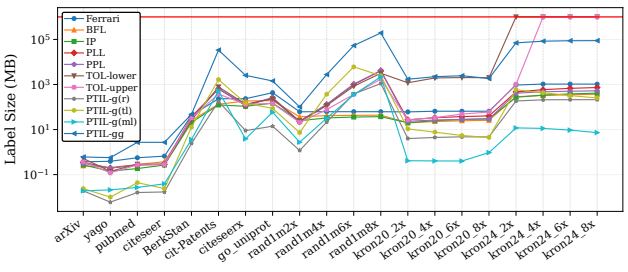


Figure 7: Label size (MB) on graphs.

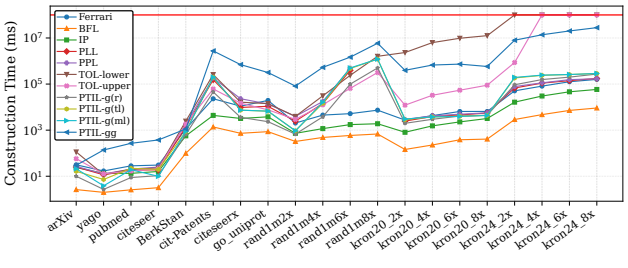


Figure 8: Construction time (ms) on graphs.

to provide an appropriate space–time trade-off under the prescribed resource and time budgets. Methods are grouped into two categories, *label+Graph* and *label-only*, to contrast query time versus relying on labels. "—" indicates that construction exceeded 24 hours (construct failure), so no query results are reported. Overall, the positive table shows that as graph size grows, label-only methods become increasingly advantageous, and **PTIL-gg** is the fastest on all datasets, achieving order-of-magnitude gains over the second-best method on all graphs. *Label+Graph* methods (e.g., **Ferrari**, **BFL**, **IP**) suffer from high latency because they must fall back to graph traversal (e.g., DFS) when labels alone cannot determine reachability. This fallback is costly on large or dense graphs. In contrast, *label-only* methods like **PTIL-gg** always resolve queries via fast, index-only membership tests. For example, on rand1m8x, **PTIL-g** completes in 10.568 ms, while Ferrari, BFL, IP, PLL, PPL, TOL-lower, and TOL-upper are 7619×, 1432×, 2803×, 35.8×, 43.2×, 25.9×, and 22.0× slower, respectively. The negative table shows that the gap between label+Graph and label-only narrows for unreachable queries; nonetheless, **PTIL-gg** remains the fastest on all graphs and still achieves multi-fold to order-of-magnitude improvements on the vast majority of graphs. On rand1m8x, **PTIL-g** completes in 12.159 ms, whereas Ferrari, BFL, IP, PLL, PPL, TOL-lower, and TOL-upper are approximately 2064×, 246×, 349×, 12.3×, 14.6×, 5.74×, and 3.84× slower, respectively. **PTIL-g** achieves greater stability and lower query latency by sacrificing a larger label size, delivering significant improvements for both positive and negative queries. In most scenarios, it offers an order-of-magnitude performance boost, demonstrating a highly pronounced advantage.

Using `cit-Patents`, we evaluate how query time varies with the number of reachable targets. As shown in Tables 4 and 5, *label+Graph* methods degrade rapidly with reachability, while *label-only* methods grow more moderately. **PTIL-gg** maintains sub-16 ms latency across all bins and outperforms all baselines by at least one order of magnitude in high-reachability scenarios.

To assess stability, we report the growth factor from the 1 ~ 10 bin to the 100,000+ bin. For positive queries, **PTIL-gg** grows by ≈ 4.69× (vs. 1.9× for **PLL** and 13,850× for **BFL**); for negative queries, **PTIL-gg** is the most stable at ≈ 6.05×. These results confirm that **PTIL-gg** offers both low latency and strong robustness across varying reachability complexities.

Restricted-Domain Reachability Queries. This experiment evaluates query performance under restricted-domain queries, where sources and targets are drawn from subsets V_s and V_t. Such queries arise in applications where endpoints are constrained by semantic roles, temporal order, or operational semantics.

We consider three configurations. The random setting is used as a neutral baseline to isolate the effect of restricting the endpoint domain, while TL and ML introduce topology-induced endpoint roles that approximate temporal, causal, and dependency-oriented query patterns. R randomly samples 10% of vertices as V_s and another 10% as V_t, allowing overlap. TL selects V_s from earlier topological levels and V_t from later levels, and ML selects V_s from middle levels and V_t from later levels, covering common topology-induced query patterns. These two settings model scenarios such as citation networks, in which earlier papers may be cited by later ones. For each configuration, we generate 100,000 queries from $V_s \times V_t$, including both positive and negative queries. Tables 6 and 7 report the query times for positive and negative queries, respectively, with each entry shown as R | TL | ML. Across the three settings, **PTIL-g** achieves the fastest query time in the vast majority of cases for both positive and negative queries. Its advantage is especially clear on large and dense graphs, where it often provides multi-fold or even order-of-magnitude speedups over both label-only and label+graph baselines.

These results show that **PTIL-g** effectively exploits the restricted $V_s \times V_t$ query space and reduces query latency across random and topology-induced restricted domains. Although our restricted-domain experiments use fixed-size endpoint sets, the full-domain setting $V_1 = V_2 = V$ is also evaluated as the largest case. The results show that PTIL remains effective, and larger costs can be controlled by adjusting the grouping parameter k.

Index Size and Construction Time. This experiment further evaluates the offline cost of PTIL by comparing index size and construction time with existing methods. Figures 7–8 compare label size and construction time across graphs, where "Fail" denotes unsuccessful index construction within the time budget. In addition to the full-graph indexing results of existing methods, these figures also include the restricted-domain results of **PTIL-g**, showing how its index size and construction cost behave under the source–target domain setting considered in this work.

The results show that **PTIL-gg** completes construction on all datasets, whereas several baselines fail on large or challenging graphs. Although **PTIL-gg** is not always the smallest or fastest to build, its offline cost is offset by its query-time advantage and can be further controlled by adjusting the number of groups k. By contrast, **PTIL-g** is evaluated under the restricted-domain setting,

Table 6: Query Time (ms) on Graphs for Positive Restricted-Domain Reachability Queries.

	label+Graph									label-only														
Data	Fer			BFL			IP			PLL			PPL			TOL-l			TOL-u			PTIL-g		
arXiv	14	10	85	78	39	34	248	200	176	7.5	5.5	17	5.0	3.8	3.0	5.1	4.9	5.2	4.6	16	15	**2.3**	**2.1**	**1.9**
yago	16	15	23	28	31	36	52	10	93	3.8	4.8	6.5	4.0	5.5	7.0	**1.5**	2.1	3.0	1.9	2.4	9.9	**1.5**	**1.2**	**1.3**
pubmed	98	82	17	18	14	50	39	30	93	9.8	10	9.2	9.2	12	8.5	4.4	4.4	6.7	5.0	4.5	19	**2.3**	**2.7**	**3.5**
citeseer	48	36	67	96	58	17	24	16	54	7.0	7.5	9.2	8.0	8.2	10	3.2	2.9	5.1	2.6	10	7.0	**1.7**	**1.8**	**2.5**
BerkStan	63	204	182	103	158	168	1511	3434	2766	12	7.5	9.5	11	8.2	8.5	7.3	5.3	7.0	5.5	15	16	**2.3**	**3.8**	**3.0**
cit-Patents	3142	6477	2788	440	1179	563	1756	4928	2238	87	160	117	90	165	150	38	57	36	27	65	50	**4.6**	**7.4**	**7.2**
citeseerx	183	123	129	88	167	334	105	262	366	13	25	7.5	16	30	8.5	22	36	16	12	37	31	**3.5**	**4.7**	**5.8**
go_uniprot	74	10	99	95	13	10	87	26	12	6.0	11	7.8	6.5	12	10	1863	94	725	7.0	29	40	**2.8**	**3.1**	**2.6**
rand1m2x	59	18	77	10	30	15	21	67	31	12	30	18	13	30	19	5.4	9.4	6.0	6.6	9.5	5.4	**2.6**	**3.1**	**2.9**
rand1m4x	404	1540	200	91	378	69	387	1634	319	65	140	82	70	155	80	37	43	27	29	49	20	**3.9**	**6.1**	**4.4**
rand1m6x	+++	+++	1887	2528	+++	616	8912	+++	2601	295	455	300	312	515	312	89	147	108	84	152	59	**6.5**	**11**	**7.1**
rand1m8x	+++	+++	+++	+++	+++	3787	+++	+++	+++	755	370	715	690	340	722	167	160	280	148	180	170	**7.9**	**9.9**	**10**
kron20_2x	297	273	27	1769	1756	10	5136	4627	18	12	12	13	8.2	9.0	10	74	78	9.3	7.1	6.5	5.0	**3.5**	**3.2**	**2.0**
kron20_4x	305	374	25	2491	2315	68	5313	4333	56	14	12	11	9.5	9.0	7.2	81	107	6.6	19	8.3	4.8	**3.5**	**2.6**	**2.0**
kron20_6x	481	425	12	2275	2074	18	5722	3614	53	14	14	11	9.5	9.8	7.8	93	87	4.3	17	12	6.3	**3.3**	**2.2**	**1.5**
kron20_8x	302	367	224	1943	1589	998	5786	3508	3260	15	13	7.0	10	6.8	4.0	75	87	90	16	12	7.4	**3.0**	**2.1**	**1.9**
kron24_2x	6265	6973	1028	+++	+++	5502	+++	+++	5283	27	25	22	25	19	16	—	—	—	27	27	33	**5.1**	**4.5**	**6.2**
kron24_4x	6798	6862	1006	+++	+++	5924	+++	+++	4226	30	23	22	18	20	15	—	—	—	—	—	—	**5.6**	**3.6**	**6.1**
kron24_6x	8045	7633	2122	+++	+++	7990	+++	+++	9430	62	30	18	24	47	9.8	—	—	—	—	—	—	**6.6**	**3.5**	**6.5**
kron24_8x	6608	5582	2905	+++	+++	+++	+++	+++	+++	30	25	12	20	19	15.8	—	—	—	—	—	—	**5.6**	**3.3**	**6.2**

Table 7: Query Time (ms) on Graphs for Negative Restricted-Domain Reachability Queries.

	label+Graph									label-only														
Data	Fer			BFL			IP			PLL			PPL			TOL-l			TOL-u			PTIL-g		
arXiv	5.4	22	22	5.4	12	11	18	63	63	4.0	7.3	6.9	3.8	7.0	6.5	7.1	10	7.8	5.8	25	22	**2.3**	**2.2**	**2.2**
yago	2.8	2.5	2.5	1.5	1.8	**1.8**	**1.4**	2.0	4.1	2.1	**1.0**	4.8	2.2	1.1	5.2	2.2	1.7	4.3	2.1	1.5	13	**1.4**	1.1	**1.8**
pubmed	2.5	4.4	2.5	2.2	2.8	3.0	3.1	8.6	6.7	3.2	5.5	4.3	3.3	7.1	4.2	3.0	6.0	4.1	3.2	5.7	12	**1.8**	**2.2**	**1.3**
citeseer	2.7	2.8	4.0	**1.8**	**1.8**	2.7	3.9	8.4	24	3.4	4.9	6.5	3.6	5.2	6.5	3.3	4.9	7.7	2.7	15	6.1	**1.8**	2.0	**2.1**
BerkStan	8.1	66	23	4.8	25	14	22	488	90	7.6	9.8	13	9.0	8.4	11	6.7	5.5	7.7	6.7	20	20	**2.5**	**3.0**	**2.0**
cit-Patents	295	2381	479	251	336	760	120	1843	401	19	75	50	18	76	59	17	46	27	28	65	46	**2.4**	**4.3**	**3.2**
citeseerx	517	301	49	19	246	16	35	645	18	7.0	22	29	8.7	22	16	7.5	15	10	8.6	28	20	**1.8**	**2.7**	**1.5**
go_uniprot	201	89	201	10	12	10	9.0	32	8.0	11	9.8	**1.0**	10	10	1.8	11	8.6	7.2	9.0	14	10	**1.9**	**2.6**	2.3
rand1m2x	74	139	92	18	33	17	74	541	174	8.2	23	17	8.4	21	17	7.8	13	10	8.1	13	8.9	**1.9**	**2.2**	**2.0**
rand1m4x	117	1486	108	143	301	244	858	1675	171	18	83	45	19	86	43	26	57	32	22	53	25	**2.4**	**5.5**	**3.0**
rand1m6x	2356	+++	1084	398	+++	342	1259	+++	1380	60	324	139	64	378	152	50	208	114	39	182	70	**3.4**	**10**	**5.8**
rand1m8x	+++	+++	+++	1782	+++	2375	4189	+++	+++	115	598	375	114	639	398	59	339	258	57	275	157	**5.0**	**13**	**8.9**
kron20_2x	36	94	7.0	37	261	12	52	441	6.0	6.9	21	20	5.0	9.3	7.8	26	31	57	5.5	3.0	7.3	**1.4**	**1.9**	**1.2**
kron20_4x	41	112	6.0	61	463	8.0	83	555	7.0	8.2	20	22	5.3	9.2	7.4	42	28	60	14	9.4	13	**1.8**	**1.8**	**1.4**
kron20_6x	68	69	7.0	57	325	11	155	347	6.0	8.5	22	20	5.3	8.3	6.9	37	15	32	12	4.9	23	**2.2**	**1.7**	**1.4**
kron20_8x	39	42	11	75	219	38	81	169	34	9.2	20	23	5.2	4.8	7.8	35	13	33	10	4.4	20	**1.7**	**1.7**	**1.5**
kron24_2x	105	105	13	163	226	15	272	373	19	10	38	35	12	20	19	—	—	—	10	9.1	37	**2.3**	**2.3**	**1.4**
kron24_4x	88	104	12	239	233	19	221	294	13	12	35	36	9.2	19	18	—	—	—	—	—	—	**2.7**	**2.2**	**1.5**
kron24_6x	120	591	11	325	288	14	312	245	16	13	37	40	11	18	17	—	—	—	—	—	—	**2.8**	**2.2**	**1.5**
kron24_8x	90	577	11	263	239	12	282	184	21	13	36	37	9.4	17	16	—	—	—	—	—	—	**2.8**	**2.1**	**1.4**

Note for Tables 6–7: Each entry is R | TL | ML, denoting Random, Topo-Top-to-late, and Topo-Middle-to-Late, respectively. +++ denotes values larger than 9999 ms; "—" denotes unavailable results. Bold values indicate the fastest query time for each setting on each dataset.

Table 8: PTIL-gg vs. PTIL-p: Query Time (ms) and Label Size (MB).

	PTIL-gg		PTIL-p								
Dataset	**Q**	**S**	**Q(hh)**	**Q(nh)**	**Q(hn)**	**Q(nn)**	**Q(Total)**	**S(T)**	**S(S)**	**S(N)**	**S(Total)**
cit-Patents	64.240	17 443.121	9.936	9.863	10.383	19.747	49.929	4700.607	4466.527	10 790.378	20 043.909
go_uniprot	47.245	1252.238	7.471	8.103	7.438	12.367	35.381	1895.786	1787.401	1047.509	4890.180
rand1m2x	35.345	101.921	4.012	4.061	4.275	8.999	21.347	261.676	261.731	87.861	634.156
rand1m4x	78.889	2103.801	9.299	10.253	9.915	25.302	54.769	515.295	517.589	1125.440	2181.212
rand1m6x	1542.715	38 121.780	25.153	25.555	26.672	63.548	140.928	5940.876	6155.124	22 628.910	34 747.798
rand1m8x	1734.868	109 220.848	37.852	38.049	38.267	87.852	202.021	23 892.382	24 719.013	88 723.110	137 357.394

and its construction time and label size are generally comparable to those of label-only methods, while in some cases being even smaller or faster. This indicates that **PTIL-g** can preserve the query-time benefit of interval-based labeling without necessarily incurring higher preprocessing or storage costs than existing label-only approaches.

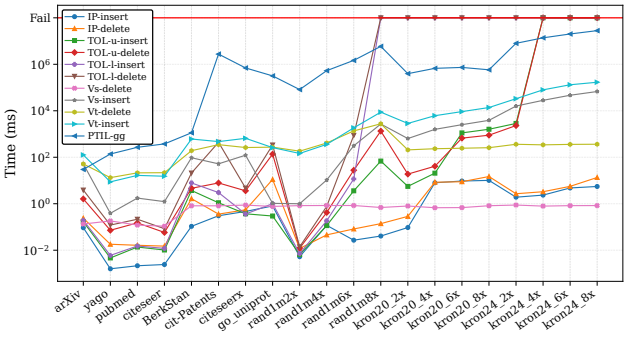
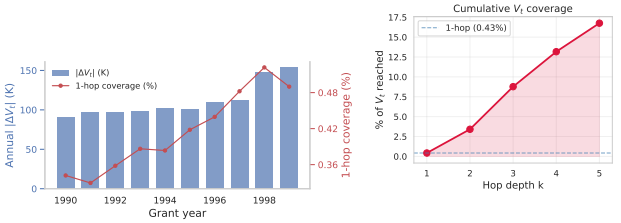


Figure 9: Update time (ms) on graphs.



(a) Annual $|\Delta V_t|$ volume and 1-hop coverage from V_s, 1990–1999.

(b) Cumulative V_t coverage from V_s as a function of hop depth k.

Figure 10: Enterprise patent landscape workload on dataset cit-Patents. V_s: 50 top-cited 1975–1985 C&C patents; V_t: all 1990–1999 USPTO grants($|V_t| \approx 1.1$M).

Global-Group vs Probability-Aware. This experiment is designed as a controlled workload-skew study rather than a replacement for query-log-based evaluation. In deployment, hot sources and targets can be identified from query logs, access statistics, or application-level monitoring rules. To evaluate **PTIL-p**, we sample 10% of vertices as high-probability sources V_{hs} and 10% as high-probability targets V_{ht}, with the remaining vertices forming V_n. We generate 100,000 queries from hot/normal source–target combinations. **PTIL-gg** uses $k = 10$, while **PTIL-p** uses priority routing with $k_1 = k_2 = 10$, $k_3 = 1$. Table 8 shows that, under comparable label size S, **PTIL-p** consistently achieves lower total query time Q(Total) than **PTIL-gg**. For example, on `rand1m6x`, **PTIL-p** uses less space (34,747.8 vs. 38,121.8 MB) and reduces query time from 1,542.72 ms to 140.93 ms, giving a 10.95× speedup. This improvement comes from routing frequent queries to finer-grained indexes over hot sources and targets.

Overall, **PTIL-p** provides a stronger space–time trade-off than **PTIL-gg**, with parameters (k_1, k_2, k_3) enabling flexible adjustment under workload skew.

Dynamic workload updates. This experiment evaluates the maintenance overhead of PTIL under endpoint-domain updates without full index reconstruction. We apply each update operation 1,000 times and report the total update time in Fig. 9. **PTIL-g** updates are much cheaper than rebuilding the index and are comparable to or faster than TOL on most datasets. Unlike TOL, which fails on several large graphs, **PTIL-g** covers more datasets, showing better robustness for large-scale dynamic workloads. Although IP has the lowest update time, its much slower query performance limits its overall competitiveness.

8 CASE STUDY

Commercial patent intelligence platforms (e.g., PatSnap) must repeatedly determine which newly granted patents are influenced by a firm's core technology portfolio. This query captures how core technologies diffuse into new filings, supporting competitor monitoring, licensing discovery, freedom-to-operate analysis, and R&D planning. Structurally, this is a restricted-domain reachability problem in which V_s is a small, stable set of high-impact portfolio patents and V_t is a rolling window of recently issued patents. These endpoint sets are determined by application roles: V_s is the firm's monitored core portfolio, while V_t is the rolling set of new filings relevant to that portfolio.

We instantiate this workload on the NBER cit-Patents dataset. V_s comprises the 50 most-cited Computers & Communications patents from 1975–1985. V_t contains all 1.1M patents granted in 1990–1999. Figure 10(a) reveals two operational facts: roughly 100K patents enter V_t each year, and fewer than 0.5% of them directly cite any portfolio patent. The first implies that the target domain grows continuously, so rebuilding a global reachability index after each update is impractical. The second shows that direct-citation analysis captures only a very small visible portion of portfolio influence. Figure 10(b) shows why multi-hop reachability is necessary. Extending the search to citation chains up to depth $k = 5$ reaches 16.8% of V_t, a 39× increase over the 1-hop view. This growth indicates that portfolio influence is largely indirect: later patents often build on intermediate inventions.

The alternative method—precomputing all-pairs reachability ($\approx 1.4 \times 10^{13}$ entries)—is incompatible with this update pattern. This combination of a stable V_s, a growing V_t, and deep multi-hop influence chains is exactly the operating regime targeted by PTIL's partitioned construction (Sections 4 and 5). PTIL indexes only the $|V_s| \times |V_t|$ sub-domain and supports incremental V_t insertion. As new patents are granted, the target set can be updated locally rather than reconstructing a global reachability index, making the method better aligned with the rolling-window nature of patent monitoring. The same asymmetric structure recurs in academic citation graphs and software dependency graphs. The cit-Patents instantiation thus shows that PTIL's restricted-domain abstraction is not an algorithmic convenience, but a workload pattern induced by real application semantics.

9 CONCLUSIONS

We presented **PTIL**, a partitioned reachability labeling framework aligned with structured query domains. PTIL timestamps only targets and stores source-side disjoint intervals, reducing each query to a single membership test while avoiding redundant global labels. It offers explicit space–time trade-offs and two modes: **PTIL-g** for domain-level control and **PTIL-p** for further prioritization when query distributions are known. Together, PTIL shows that reachability indexing can be domain-aligned, storage-efficient, and scalable. Future work includes parallel and distributed PTIL construction, enabled by independent group construction after shared preprocessing, as well as more efficient batched endpoint-domain maintenance and extensions to temporal and heterogeneous graphs.

Acknowledgments

Dong Wen is supported by ARC DP260100709 and ARC DE240100668. Tianming Zhang is supported by NSFC 62302451.

Artifacts

The implementation and experimental artifacts for PTIL are available at https://github.com/hh-2575/ptil. The repository contains the source code and instructions for building and running the implementation. The experiments use publicly available graph datasets described in Section 7; due to their size, these datasets are not redistributed with the artifact.

References

[1] Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. [n. d.]. Efficient management of transitive relationships in large data and knowledge bases. *Proc. ACM SIGMOD Int. Conf. Manage. Data, 1989, pp. 253–262.* ([n. d.]).

[2] Ramadhana Bramandia, Byron Choi, and Wee Keong Ng. 2010. Incremental Maintenance of 2-Hop Labeling of Large Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2010).

[3] 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*. 119–128.

[4] L. Chen, A. Gupta, and M. E. Kurul. 2005. Stack-based algorithms for pattern matching on dags. In *Proceedings of the 31st International Conference on Very Large Data Bases*.

[5] Liuyi Chen, Yuchen Hu, Zhengyi Yang, Xu Zhou, Wenjie Zhang, and Kenli Li. 2025. Efficient Partition-Based Approaches for Diversified Top-k Subgraph Matching. *Proceedings of the VLDB Endowment* 19, 4 (Dec. 2025), 698–712. doi:10.14778/3785297.3785310

[6] 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. *Proceedings of the ACM SIGMOD International Conference on Management of Data. Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.), ACM, 193–204* (2013).

[7] EDITH COHEN, ERAN HALPERIN, HAIM KAPLAN, and URI ZWICK. 2003. REACHABILITY AND DISTANCE QUERIES VIA 2-HOP LABELS. *SIAM J. Comput. 32(5), 1338–1355 (2003)* (2003).

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

[9] Kathrin Hanauer, Christian Schulz, and Jonathan Trummer. 2022. O'Reach: Even Faster Reachability in Large Graphs. *J. Exp. Algorithmics 27, 4, Article 4.2 (October 2022)* (2022).

[10] Daoliang He and Pingpeng Yuan. 2025. TDS: fast answering reachability queries with hierarchical traversal trees. *Clust. Comput.* 28, 3 (2025), 178. doi:10.1007/S10586-024-04814-8

[11] Huangleshuai He, Zhengyi Yang, Dong Wen, Wenke Yang, and John Shepherd. 2026. Graph Reachability Queries: Empirical Evaluation and Practical Guidelines. In *Web Information Systems Engineering - WISE 2025 PhD Symposium, Demos and Workshops*, Irfan Awan, Muhammad Younas, Yanchun Zhang, Mahmoud Barhamgi, and Hua Wang (Eds.). Springer Nature Singapore, Singapore, 361–373.

[12] Huangleshuai He, Zhengyi Yang, Dong Wen, Wenqian Zhang, Michael Yu, Wenke Yang, and Wenjie Zhang. 2025. A Survey on Efficient Graph Reachability Queries. In *Data Science: Foundations and Applications*, Xintao Wu, Myra Spiliopoulou, Can Wang, Vipin Kumar, Longbing Cao, Xiangmin Zhou, Guansong Pang, and Joao Gama (Eds.). Springer Nature Singapore, Singapore, 58–77.

[13] Chengying Huan, Zhengyi Yang, Haoshen Yang, Shaonan Ma, Rong Gu, Fang Xi, Yongchao Liu, Guihai Chen, and Chen Tian. 2025. Gem: Scalable Monotonic Graph Processing Beyond Billion-Scale on a Single Machine. *Proceedings of the ACM on Management of Data* 3, 6 (Dec. 2025), 1–30. doi:10.1145/3769795

[14] Ruoming Jin, Zhen Peng, Wendell Wu, Feodor Dragan, Gagan Agrawal, and Bin Ren. 2020. Parallelizing Pruned Landmark Labeling: Dealing with Dependencies in Graph Algorithms. In *Proceedings of the 2020 International Conference on Supercomputing (ICS '20)*. ACM, New York, NY, USA, 1–13. doi:10.1145/3392717.3392745

[15] Ruoming Jin and Guan Wang. 2013. Simple, Fast, and Scalable Reachability Oracle. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1978–1989.

[16] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 2009. 3-HOP: a high-compression indexing scheme for reachability query *(SIGMOD '09)*. 813–826.

[17] Ruoming Jin, Yang Xiang, Ning Ruan, and Haixun Wang. [n. d.]. "Efficiently answering reachability queries on very large directed graphs. ([n. d.]).

[18] A. B. Kahn. 1962. Topological sorting of large networks. *Commun. ACM* (1962).

[19] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. https://snap.stanford.edu/data/.

[20] Qiuyi Lyu, Yuchen Li, Bingsheng He, and Bin Gong. 2021. DBL: Efficient Reachability Queries on Dynamic Graphs. (2021), 761–777.

[21] Florian Merz and Peter Sanders. 2014. PReaCH: A Fast Lightweight Reachability Index Using Pruning and Contraction Hierarchies. *Proceedings of the European Symposium on Algorithms* (2014), 701–712.

[22] Yue Pang, Lei Zou, and Yu Liu. 2023. IFCA: Index-Free Community-Aware Reachability Processing Over Large Dynamic Graphs. *2023 IEEE 39th International Conference on Data Engineering (ICDE)* (2023).

[23] Xiao Qiu, Wei Cen, Zhihong Qian, Yongqing Peng, Yiqing Zhang, Xuemin Lin, and Jiaheng Zhou. 2018. Real-time constrained cycle detection in large dynamic graphs. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1876–1888.

[24] Thomas Reps. 1998. Program analysis via graph reachability. *Information and Software Technology* 40, 11–12 (1998), 701–726.

[25] Ralf Schenkel, Alexander Theobald, and Gerhard Weikum. 2005. Efficient creation and incremental maintenance of the HOPI index for complex XML document collections. In *Proceedings of the 21st International Conference on Data Engineering*. 360–371.

[26] Bernhard Scholz, Chengzheng Zhang, and Cristina Cifuentes. 2008. User-input dependence analysis via graph reachability. In *Proceedings of the 2008 8th IEEE International Working Conference on Source Code Analysis and Manipulation*. 25–34.

[27] Neha Sengupta, Amitabha Bagchi, Maya Ramanath, and Srikanta Bedathur. 2019. ARROW: Approximating Reachability Using Random Walks Over Web-Scale Graphs. In *35th IEEE International Conference on Data Engineering, ICDE 2019*. IEEE, 470–481.

[28] Stephan Seufert, Avishek Anand, Srikanta J. Bedathur, and Gerhard Weikum. 2013. Ferrari: Flexible and efficient reachability range assignment for graph indexing. *Proc. IEEE Int. Conf. Data Eng., 2013, pp. 1009–1020.* (2013).

[29] K. Simon. 1988. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Comput. Sci.* 58, 1–3 (1988), 325–346.

[30] 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* (2017).

[31] Zhixiang Su, Di Wang, Xiaofeng Zhang, Lizhen Cui, and Chunyan Miao. 2022. Efficient Reachability Query with Extreme Labeling Filter. In *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*, K. Selcuk Candan, Huan Liu, Leman Akoglu, Xin Luna Dong, and Jiliang Tang (Eds.). ACM, 966–975. doi:10.1145/3488560.3498446

[32] S. Trißl and U. 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 2007)*. ACM.

[33] S. J. van Schaik and O. de Moor. [n. d.]. A memory efficient reachability data structure through bit vector compression. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*.

[34] R. R. Veloso, L. Cerf, W. M. Jr., and M. J. Zaki. 2014. Reachability queries in very large graphs: A fast refined online search approach. In *Proc. 17th Int. Conf. Extending Database Technol.* 511–522.

[35] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu. [n. d.]. Dual labeling: Answering graph reachability queries in constant time. In *Proc. IEEE 29th Int. Conf. Data Eng.*

[36] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. 2018. Reachability querying: An independent permutation labeling approach. *The VLDB Journal 27, 1 (2018), 1–26* (2018).

[37] Zhuoqing Xu, Xin Cao, and Zhengyi Yang. 2026. ReaCH-TGN: Contrastive Hop and Time-Aware Temporal Graph Network for Reachability Prediction. In *Databases Theory and Applications*, Renata Borovica-Gajic, Arijit Khan, Bolong Zheng, Xiaoyang Wang, and Junhao Gan (Eds.). Springer Nature Singapore, Singapore, 208–222.

[38] Zhuoqing Xu, Zhengyi Yang, Xiaoshuang Chen, Huangleshuai He, and Xin Cao. 2025. Efficient Answering of k-Reachability on Temporal Bipartite Graphs. In *Databases Theory and Applications*, Tong Chen, Yang Cao, Quoc Viet Hung Nguyen, and Thanh Tam Nguyen (Eds.). Springer Nature Singapore, Singapore, 224–238.

[39] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast and Scalable Reachability Queries on Graphs by Pruned Labeling with Landmarks and Paths. *22nd CIKM 2013: San Francisco, CA, USA* (2013).

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

[41] Jeffrey Xu Yu and Jiefeng Cheng. 2010. Graph Reachability Queries: A Survey. In *Managing and Mining Graph Data*. Advances in Database Systems, Vol. 40. Springer, 181–215.

[42] Pingpeng Yuan, Yujie You, Shuang Zhou, Hai Jin, and Ling Liu. 2022. Providing Fast Reachability Query Services With MGTag: A Multi-Dimensional Graph Labeling Method. *IEEE Trans. Serv. Comput.* 15, 2 (2022), 1000–1011. doi:10.1109/TSC.2020.2969898

[43] Hilmi Yıldırım, Vineet Chaoji, and Mohammed J. Zaki. 2012. GRAIL: a scalable index for reachability queries in very large graphs. *VLDB J. 21(4): 509-534 (2012)* (2012).

[44] Chao Zhang, Angela Bonifati, and M. Tamer Özsu. 2023. An Overview of Reachability Indexes on Graphs. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023*. ACM, 61–68.

[45] Andy Diwen Zhu, Wenqing Lin, Sibo Wang, and Xiaokui Xiao. 2014. Reachability Queries on Large Dynamic Graphs: A Total Order Approach. (2014), 1323–1334.