Scalable Privacy-Preserving Shortest Path Distance Computation via 2-Hop Labeling in MPC

HUIZHONG WANG, The Chinese University of Hong Kong, Shenzhen, China
YUANYUAN ZENG, The Chinese University of Hong Kong, Shenzhen, China
KUN CHEN*, Ant Group, China
WEI DONG, Nanyang Technological University, Singapore
CHENHAO MA*, The Chinese University of Hong Kong, Shenzhen, China

Shortest distance computation is a fundamental problem in graph data analysis, with critical applications in financial fraud detection, website ranking, and social network analysis. In real-world settings, however, graph data is often distributed across multiple mutually untrusted organizations, making accurate shortest path computation under strict privacy constraints a major challenge. Existing solutions face two key limitations: (1) traditional distributed algorithms lack privacy protection; and (2) secure multi-party computation (MPC)-based methods, though privacy-preserving, suffer from high computational overhead and poor scalability, restricting them to graphs with only tens of thousands of nodes. To address these challenges, we propose `PrivHop`, a novel algorithm that integrates 2-hop labeling with MPC in a two-phase framework. In the offline phase, `PrivHop` constructs an optimized boundary graph index to reduce global queries to small-scale boundary graph queries. In the online phase, it introduces a privacy-aware dynamic pruning strategy based on differential privacy to substantially reduce iteration complexity with privacy guarantees. Extensive experiments on eight real-world datasets show that `PrivHop` preserves privacy while scaling to million-node graphs, achieving up to $10^6\times$ reductions in both runtime and communication compared to state-of-the-art methods.

CCS Concepts: • **Theory of computation → Shortest paths**; • **Security and privacy → Management and querying of encrypted data**.

Additional Key Words and Phrases: Shortest path distance; Multi-praty compution (MPC); 2-hop index

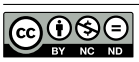
ACM Reference Format:
Huizhong Wang, Yuanyuan Zeng, Kun Chen, Wei Dong, and Chenhao Ma. 2026. Scalable Privacy-Preserving Shortest Path Distance Computation via 2-Hop Labeling in MPC. *Proc. ACM Manag. Data* 4, 1 SIGMOD, Article 81 (February 2026), 27 pages. https://doi.org/10.1145/3786695

1 Introduction

Shortest path distance computation is a fundamental problem in graph data analysis and underpins critical applications such as financial fraud detection [41], website ranking [38], and social network analysis. However, in real-world settings, graph data are often fragmented across multiple mutually distrustful organizations. Privacy regulations and commercial confidentiality prevent direct data sharing, while accurate shortest path computation requires access to the complete global graph

*Corresponding authors.

Authors' Contact Information: Huizhong Wang, huizhongwang@link.cuhk.edu.cn, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China; Yuanyuan Zeng, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, zengyuanyuan@cuhk.edu.cn; Kun Chen, Ant Group, Beijing, China, ck413941@antgroup.com; Wei Dong, Nanyang Technological University, Singapore, wei_dong@ntu.edu.sg; Chenhao Ma, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China, machenhao@cuhk.edu.cn.



ACM 2836-6573/2026/2-ART81
https://doi.org/10.1145/3786695

topology. This creates a fundamental tension between privacy preservation and computational accuracy. As illustrated in Figure 1, consider a financial transaction network where nodes represent users and edges denote transaction relationships. To assess relational proximity and evaluate transaction risks, Bank P needs to compute the shortest path distance between users A and B. Yet, relying solely on its local data, it may erroneously conclude an "infinite distance" because interbank paths through Bank Q remain hidden. Existing distributed and single-machine algorithms (e.g.,

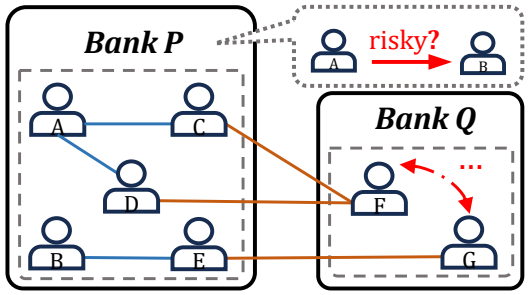


Fig. 1. Shortest path distance computation in a cross-bank transaction network.

DHCA [50], Optimized Dijkstra [27]) inherently depend on global graph information, making them unsuitable for scenarios where data privacy is paramount. This conflict highlights an urgent need for novel computational paradigms that reconcile privacy requirements with accurate graph analysis.

Multi-party computation (MPC) offers a promising solution to this challenge. Built upon cryptographic primitives such as garbled circuits (GC), secret sharing (SS), and homomorphic encryption (HE), MPC enables multiple parties to collaboratively compute functions over their joint data without revealing sensitive information. Each party encrypts its local data and executes a shared computation protocol in the encrypted domain, ensuring that only the final result is revealed while all intermediate data remain confidential [11, 16, 26, 42, 46]. Since its inception in the 1980s, MPC has evolved into a cornerstone of privacy-preserving computation, garnering significant attention from both academia and industry [19, 29].

Motivation. Existing privacy-preserving shortest path distance algorithms typically deploy classical algorithms directly within the MPC framework [45]. Aly et al. [3, 4] pioneered MPC implementations of Bellman-Ford and Dijkstra's algorithms, establishing the foundational framework for this field. Keller et al. [30] and Liu et al. [33] introduced ORAM [37] to address private memory access issues. While this obfuscates memory access patterns and preserves privacy, it inevitably incurs significant performance overhead due to the high cost of oblivious memory operations. Mohammad et al. [6] proposed specialized algorithms for sparse and dense graphs, leveraging parallel oblivious reading subroutines to bypass ORAM. However, their solutions still require hours of computation on graphs with merely tens of thousands of nodes, demonstrating limited scalability. Benjamin et al. [36] introduced d-normalized graph methods and parallel priority queues to enhance per-iteration computational parallelism. Yet, their approaches continue to suffer from ORAM overhead and excessive computation iterations. Although these works represent meaningful advances, they remain constrained to graphs with up to tens of thousands of nodes. This leaves computational efficiency a critical unresolved issue, especially when addressing real-world graphs at the million-node scale.

Challenge. These limitations stem from a fundamental conflict between algorithmic characteristics and MPC's privacy requirements. ❶ *Dynamic data dependencies* (e.g., priority queue accesses

in Dijkstra's algorithm or memory accesses in ORAM-based solutions) *inherently clash with MPC's static computation model,* necessitating expensive oblivious simulation of control flows [40]. ❷ *The sequential nature of shortest path algorithms* (e.g., iterative updates in Bellman-Ford and Dijkstra) *limits opportunities to parallelize computation within MPC protocols*, resulting in computation iterations that grow linearly with graph size [4]. Overcoming these challenges requires rethinking how shortest path distance computation is structured within MPC to break free from these bottlenecks.

Our Solution. To address these challenges, we propose a novel privacy-preserving shortest path distance algorithm for multi-party settings, aiming to support scalable and efficient queries over large graphs with strong privacy guarantees. Our solution is built on two key strategies: (1) reducing the effective graph size during online computation to scale to million-node datasets; and (2) redesigning the algorithmic workflow to minimize shortest-path iterations without compromising privacy.

We present `PrivHop`, a two-phase privacy-preserving shortest path distance query framework based on 2-hop indexing[1]. `PrivHop` consists of an offline index construction phase and an online privacy-preserving query phase, jointly mitigating the efficiency bottlenecks of MPC. In the offline phase, optimized 2-hop indexing transforms graph-based queries into index-based distance queries, substantially reducing the problem size and secure computation cost, addressing Challenge ❶ . In the online phase, we employ a decoupled storage architecture with lightweight oblivious interaction protocols to eliminate ORAM, and introduce a differentially private dynamic pruning strategy to significantly reduce distance refinement iterations while preserving privacy, addressing Challenge ❷ .

To promote reproducibility, we have released our source code and datasets.[2]

Contributions. In summary, our main contributions are as follows:

- We present the first integration of 2-hop labeling with MPC-based shortest path distance queries, introducing a boundary graph transformation strategy during index construction that substantially reduces the graph scale for online query processing and, as a result, minimizes operations requiring collaborative computation.
- We propose a differentially private dynamic pruning strategy that reduces distance refinement iterations during online processing, delivering substantial efficiency gains while ensuring strong privacy guarantees.
- Extensive experiments on eight real-world datasets of varying scales demonstrate that `PrivHop` delivers (1) up to $10^6\times$ runtime acceleration and (2) up to $10^6\times$ communication reduction compared to state-of-the-art solutions.

Outline. The rest of the paper is organized as follows. Section 2 introduces the preliminaries. Section 3 presents an overview of our algorithm. Section 4 describes the technical details of the offline index construction phase, and Section 5 presents the online query processing phase. Experimental results are reported in Section 6. Related work is reviewed in Section 7, and Section 8 concludes the paper.

2 Preliminaries

In this section, we begin by presenting the problem of privacy-preserving shortest path distance queries. Next, we thoroughly examine the state-of-the-art approaches. Table 1 summarizes frequently used notations in this paper.

[1] A technique that precomputes intermediate "hub" distances to efficiently answer shortest path queries [32].
[2] https://github.com/HuiZ-W/PrivHop.git

Table 1. Notations and meanings.

Notation	Meaning
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	conceptual global graph (union of all parties')
$G_i = (V_i, E_i)$	a private subgraph on party i
V_{Bi}	the set of boundary vertices in G_i
E_{cut}	the set of public cutting edges in $\mathcal{G}$
$dist_G(u, v)$	the shortest path distance between u and v in G
$\mathcal{P}(v)$	the party where the vertex v is owned
$p_{\mathcal{G}}(s, t)$	a path from s to t in $\mathcal{G}$
$p^{bd}(s, t)$	a boundary path between s to t
$p^{in}(s, t)$	an interior path between s to t

2.1 Problem Definition

We focus on the two-party setting, where a global undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ and $\mathcal{E}$ denoting the vertex and edge sets, is partitioned between two mutually distrustful parties. The partitioning produces two disjoint subgraphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$, along with a set of cutting edges E_{cut}. Each is privately held by party P_0 and P_1 respectively. To formalize the model, we introduce the following definitions:

Definition 2.1 (Distributed Graph Partition). A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is distributed across two parties as $\{G_i(V_i, E_i)\}_{i=0}^{1} \cup E_{cut}$, where:

- $\mathcal{V} = \bigcup_{i=0}^{1} V_i$, with $V_i \cap V_j = \emptyset$ for $i \neq j$,
- $\mathcal{E} = \bigcup_{i=0}^{1} E_i \cup E_{cut}$, where $E_i = \{e(u, v) \in E \mid \mathcal{P}(u) = \mathcal{P}(v)\}$,
- $E_{cut} = \{(u, v) \in E \mid \mathcal{P}(u) \neq \mathcal{P}(v)\}$ denotes the set of cutting edges that connect vertices across the two subgraphs and is publicly known to both parties,
- Each party P_i privately holds its subgraph G_i for $i \in \{0, 1\}$.

Based on this partitioning, we classify vertices and paths as follows:

Definition 2.2 (Vertex Category [50]). Given a subgraph $G(V, E) \in \mathcal{G}(\mathcal{V}, \mathcal{E})$, let V_B denote the set of boundary vertices of G, defined as:

$$V_B = \{v \mid \exists e(u, v) \in \mathcal{E}, \mathcal{P}(u) \neq \mathcal{P}(v), v \in V\}.$$

Definition 2.3 (Path Classification [50]). Given a graph $\mathcal{G}$ distributed across two parties as $\{G_i(V_i, E_i)\}_{i=0}^{1} \cup E_{cut}$, the paths between any two vertices s and t, denoted as $p(s, t) = \{v_0 = s, v_1, \ldots, v_{k-1}, v_k = t\}$, are categorized as:

- A *boundary path* $p^{bd}(s, t)$ if $\exists i \in [0, k-1]$ such that $\mathcal{P}(v_i) \neq \mathcal{P}(v_{i+1})$,
- An *interior path* $p^{in}(s, t)$ if $\forall i \in [0, k-1], \mathcal{P}(v_i) = \mathcal{P}(v_{i+1})$.

Problem Definition. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ distributed across two parties as $\{G_i(V_i, E_i)\}_{i=0}^{1} \cup E_{cut}$ and a vertex pair (s, t), the shortest path distance query $q(s, t)$ returns the shortest distance $dist_{\mathcal{G}}(s, t)$ between s and t.

Example 1. *Figure 2 illustrates our graph partitioning model through an example involving two parties. The graph comprises 11 vertices, where boundary vertices $\{v_0, \ldots, v_5\}$ maintain cross-partition connections and are jointly visible to both parties. In contrast, interior vertices $\{v_6, \ldots, v_{10}\}$ are confined to their respective partitions, remaining private to the owning party. Four cutting edges (dashed lines) connect the vertices across the two partitions. For instance, the path (v_7, v_8) is an intra-partition path of length 1, while (v_8, v_0, v_4, v_9) exemplifies an inter-partition boundary path of length 3.*

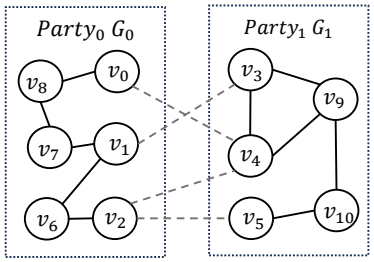


Fig. 2. Global graph $\mathcal{G}$ consisting of two subgraphs G_0 and G_1.

2.2 Differential Privacy (DP) on Graphs

This part formalizes the key concepts of differential privacy (DP) in graph settings. Let $G = (V, E, w)$ be a connected weighted graph where:

- V is the vertex set with $n = |V|$
- E is the edge set with $m = |E|$ $(n \leq m + 1)$
- $w : E \to \mathbb{R}^+$ assigns positive weights to edges

We first give the notion of neighboring graphs.

Definition 2.4. (Neighboring Graphs [23]) Two graphs $G = (V, E, w)$ and $G' = (V, E, w')$ are *neighboring* (denoted $G \sim G'$) if their weight functions satisfy:

$$\|w - w'\|_1 := \sum_{e \in E} |w(e) - w'(e)| \leq 1$$

The Differential Privacy (DP) introduced by Dwork [22] is defined below.

Definition 2.5 ((ϵ, δ)-Differential Privacy [22]). A randomized algorithm $\mathcal{A} : \mathcal{G} \to \mathcal{O}$ satisfies (ϵ, δ)-differential privacy if $\forall G \sim G'$ and all measurable $S \subseteq \mathcal{O}$:

$$\Pr[\mathcal{A}(G) \in S] \leq e^{\epsilon} \Pr[\mathcal{A}(G') \in S] + \delta$$

Special cases:

- Pure ϵ-DP when $\delta = 0$
- Stronger privacy guarantees when $\epsilon, \delta \to 0$

Definition 2.6 (Global Sensitivity [22]). For any function $f : \mathcal{G} \to \mathbb{R}^d$, its global sensitivity is:

$$\Delta f := \sup_{G \sim G'} \|f(G) - f(G')\|_1$$

where the supremum is taken over all neighboring graph pairs.

Lemma 2.7 (Laplace Mechanism [22]). *For any function $f : \mathcal{G} \to \mathbb{R}^d$ with sensitivity Δf, the mechanism:*

$$\mathcal{M}(G) := f(G) + (Y_1, ..., Y_d), \quad Y_i \overset{i.i.d.}{\sim} \text{Lap}(0, \Delta f/\epsilon)$$

satisfies ϵ-differential privacy.

We employ differential privacy (DP) mechanisms into the pruning algorithm to ensure provable privacy guarantees during online query processing.

2.3 Cryptographic Primitives

Secure multi-party computation (MPC) typically relies on three cryptographic primitives: garbled circuits (GC), secret sharing (SS), and homomorphic encryption (HE). Our protocol builds on two of these: secret sharing (SS) and homomorphic encryption (HE).

Secret Sharing (SS). Additive secret sharing splits a value x into shares $\langle x\rangle_0, \langle x\rangle_1$ such that $x = \langle x\rangle_0 + \langle x\rangle_1 \pmod{2^W}$. Each party P_i holds $\langle x\rangle_i$. Linear operations (e.g., addition) are performed locally without interaction, while multiplications require Beaver triples [10] in an interactive protocol. Arbitrary computations combine arithmetic and Boolean shares via conversion and secure circuit evaluation [21].

Homomorphic Encryption (HE). HE schemes [17, 39] enable computations directly on ciphertexts without decryption. Partially homomorphic encryption (PHE) supports a single operation type (e.g., addition), while fully homomorphic encryption (FHE) allows arbitrary functions but with significant computational overhead.

Threat Model and Our Protocol. We focus on the standard semi-honest (honest-but-curious) model, where parties faithfully execute the protocol but may attempt to infer private data from observed execution traces.

To achieve obliviousness, the protocol must ensure that its execution trace—including memory access patterns, control flow, and communication order—depends only on public parameters or input size, and not on specific private inputs. However, in shortest-path distance algorithms, memory access patterns may reveal the order of vertex selection and distance updates, potentially exposing the graph structure. Therefore, prior protocols often employ techniques such as ORAM to hide these patterns.

Our protocol, instead, uses a 2-out-of-2 additive secret sharing-based 2PC protocol [43] supporting arithmetic operations ($+, -, \times, \div$) and logical operations (comparisons, MUX2 [21, 43]). To prevent access-pattern leakage, we integrate a homomorphic-encryption-based oblivious shuffling protocol [35], which eliminates the need for ORAM while providing equivalent security guarantees, ensuring that all memory access patterns remain independent of the actual inputs.

2.4 2-hop Index for Distance Queries

State-of-the-art shortest path query solutions in non-encrypted environments often employ 2-hop indexing techniques for their superior query efficiency. To provide the necessary background, we briefly review the fundamental principles of the 2-hop index [32]. Its key advantage lies in enabling shortest distance queries between any two vertices via at most two hops through intermediate "hub" nodes.

Index Structure. Given a graph $G = (V, E)$, the 2-hop index assigns to each vertex $v \in V$ a label set $L(v)$ containing key-value pairs $(u, dist(u, v))$. The combined index $\bigcup_{v\in V} L(v)$ satisfies the following coverage property:

Definition 2.8 (2-Hop Cover [1, 32]). For any two vertices $s, t \in V$, there exists a vertex $w \in L(s) \cap L(t)$ such that:

$$dist(s, t) = dist(s, w) + dist(w, t).$$

Index Size. Let $\delta = \max_{v\in V} |L(v)|$ denote the maximum label size. The overall index size is $O(n \cdot \delta)$ [32].

Query Process. Given two vertices $s, t \in V$, the shortest path distance can be computed using:

$$Query(s, t, L) = \min_{w\in L(s)\cap L(t)} L(s)[w] + L(t)[w]. \tag{1}$$

This query operation has a time complexity of $O(|L(s)| + |L(t)|)$.

Parallel Shortest Distance Labeling (PSL) [32]. As the state-of-the-art 2-hop indexing technique, PSL fundamentally transforms the index construction paradigm by parallelizing pruned breadth-first searches (BFS), effectively eliminating the inherent sequential bottlenecks of traditional approaches.

However, directly applying existing indices such as PSL to privacy-preserving scenarios is infeasible: the size and content of labels may leak sensitive information such as edge weights and topological structure. To address this, we introduce boundary graph transformations (detailed in Section 4) based on the PSL framework, extending the 2-hop index into a solution that supports privacy-preserving shortest distance queries.

2.5 State-of-the-Art Approaches

Privacy-Preserving Dijkstra [36]. This approach proposes a privacy-preserving variant of Dijkstra's algorithm based on a d-normalized replicated adjacency list. It transforms the graph into a secret-shared form for MPC, achieving favorable theoretical complexity. Specifically, it employs: (1) a secure renaming and sorting algorithm mapping vertex IDs to $[1, |V|]$ in $O(\log |V|)$ rounds[3]; and (2) an oblivious d-normalization algorithm that constructs the replicated adjacency list within $O(1)$ rounds. The resulting data structure supports parallel single-round computations for shortest path queries. Theoretical analysis reports $O(|V|)$ Dijkstra iterations, a secure operation complexity of $O((|V| + |E|) \log |V|)$, and a round complexity of $O(|V| \log |V| \log \log \log |V|)$.

However, this algorithm faces several practical limitations:

(1) It incurs high overhead due to frequent secure comparisons and secure shuffles, resulting in runtimes far exceeding theoretical predictions.
(2) The d-normalized structure amplifies storage requirements (theoretically 4×), which becomes a severe bottleneck in distributed settings for large graphs.
(3) Reliance on Distributed Oblivious RAM (DORAM [34]) introduces additional logarithmic communication overhead and implementation complexity.

Privacy-Preserving Bellman-Ford [6]. Designed for sparse graphs, this algorithm improves performance via two techniques: (1) parallel read operations during preprocessing to concentrate costly computations upfront, and (2) Ladner-Fisher parallel prefix computation for distance updates, reducing the round complexity per Bellman-Ford iteration to $O(\log |V|)$. While this design avoids ORAM through innovative data access patterns, it suffers from critical limitations:

(1) It requires $O(|V|)$ Bellman-Ford iterations to converge, leading to prohibitive latency on million-node graphs.
(2) The preprocessing data structure of size $O(|E|)$ significantly increases memory usage and setup time on large-scale graphs.

3 Overview

Building on insights from state-of-the-art privacy-preserving shortest path distance algorithms (Section 2.5), `PrivHop` introduces a two-phase framework that addresses their limitations in iteration complexity, scalability, and ORAM dependency. As shown in Figure 3 and Algorithm 1, `PrivHop` operates in a two-party computation setting where each party, P_0 and P_1, holds private graph data (G_0 and G_1) and collaboratively executes two key phases:

During the **offline index construction phase**, each party independently builds a Hop-Boundary index HB based on its local topology. This preprocessing achieves two objectives: (1) it construct an internal subgraph 2-hop index HB_S, which enables efficient retrieval of intra-subgraph shortest path distances; and (2) it identifies boundary vertices to construct a compact boundary graph

[3]The number of sequential communication rounds among parties required to complete the protocol. High round complexity increases latency.

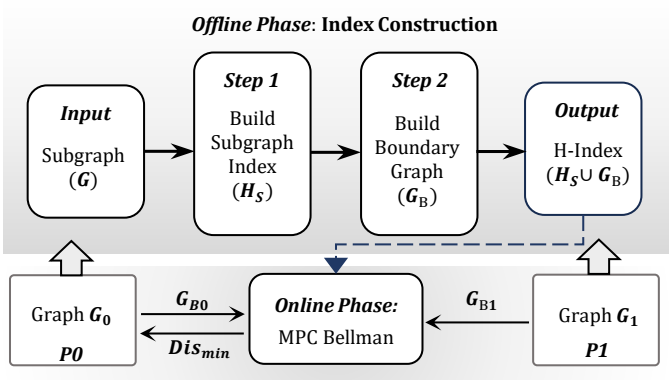


Fig. 3. The overview of PrivHop.

Algorithm 1: Workflow of PrivHop

Offline Phase // Each party runs locally
Input : $G(V, E)$ and E_{cut}
Output: Index HB_S, Boundary Graph G_B
$HB_S \leftarrow$ IndexConstruction(G)
$V_B \leftarrow$ FindBoundary(V, E_{cut})
$G_B \leftarrow$ BoundaryGraphConstruction(V_B, L, G)

Online Phase
Input : P_0: source s, target t, V_B, G_{B0}, index HB_S;
P_1: G_{B1}, E_{cut}
Output: P_0 learns the shortest distance D_{min}
// Executed locally on P_0
$D_{S2B} \leftarrow$ Query(s, V_B, L);
$D_{T2B} \leftarrow$ Query(t, V_B, L);
// Executed collaboratively by P_0 and P_1
$D_{min} \leftarrow$ ObliviousBellman$(D_{S2B}, D_{T2B}, G_{B0}, G_{B1}, E_{cut})$;

G_B. This concise representation, derived from internal shortest path distances, preserves critical inter-subgraph connectivity while reducing the problem scale to computations on the smaller boundary graph.

In the **online query processing phase**, when a query is initiated from a source node s to a target node t, the parties collaboratively compute the shortest path using only their boundary graph information. This two-phase design ensures strong data privacy guarantees while improving computational efficiency through graph compression. Detailed implementation specifics are provided in Section 4 (offline phase) and Section 5 (online phase).

Note. Without loss of generality, all our pseudocodes assume that the query nodes s and t belong to the same party, with the other party acting as a collaborator. The case where s and t belong to different parties can be easily modified.

4 Offline Phase

This section systematically elaborates on the two core aspects of our offline phase: index structure and index construction.

4.1 Hop-Boundary Index Structure

Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be a graph partitioned between two parties as $G_i(V_i, E_i)_{i=0}^{1} \cup E_{cut}$. The most straightforward indexing strategy would have each party maintain its own internal index while collaboratively building an inter-partition index. However, privacy constraints prohibit direct sharing of local graph topology across parties. To address this, our approach requires each party to construct local internal indices and independently refine its respective boundary graph. The following section details this index structure, illustrated from the perspective of a single participating party. It should be noted that the index structure for the other party is symmetric.

A Category-Based Vertex Order. During index construction, we propose a category-based vertex prioritization strategy to process vertices according to their query-critical roles. Since boundary graph construction requires accurate distances between boundary vertices, prioritizing boundary vertices in the ranking function ensures they are more likely to appear in the labels of other boundary vertices. This property is essential for efficiently deriving the boundary graph from local 2-hop labels. Let $r(v)$ denote the ranking function for vertex v. The order $r(u) > r(v)$ is established if:

- $u \in V_B$ and $v \in V \setminus V_B$, or
- $\deg(u) > \deg(v)$, or
- $\deg(u) = \deg(v)$ and $ID(u) < ID(v)$.

HB-Index structure. The Hop-Boundary Index (*HB-Index*) consists of two components: a subgraph index $HB_S = \bigcup_{v \in V} L(v)$ and an optimized boundary graph G_B. We define HB_S and G_B as follows:

*Definition 4.1 (**Subgraph-Index**).* For a subgraph $G(V, E)$, HB_S contains a 2-hop index for each vertex. For any vertex $v \in V$ and its label $(u, d_{vu}) \in L(v)$, the following conditions hold:

- $r(u) \geq r(v)$;
- $d_{vu} = dist_G(u, v)$;
- $d_{vu} < d_{vw} + d_{uw}$ with $\forall r(w) > \max\{r(v), r(u)\}$.

This implies that each vertex v ultimately stores the shortest distances to all higher-ranked reachable bridging vertices. Based on HB_S, the local shortest path distance of any internal vertex pair (s, t) on the subgraph G can be computed.

*Definition 4.2 (**Boundary Graph**).* Let $G_B(V_B, E_B, W)$ denote the *boundary graph* of subgraph $G(V, E)$, where V_B is a set of boundary vertices, E_B is a set of egdes among V_B and $W : E_B \to \mathbb{R}^+$ assigns weights to edges. The boundary graph satisfies:

- $\forall u, v \in V_B, dist_{G_B}(u, v) = dist_G(u, v)$,
- $\forall e(s, t) \in E_B, W(s, t) = dist_G(s, t)$.

This ensures G_B preserves all pairwise shortest path distances between boundary vertices.

In summary, the HB_S enables efficient retrieval of internal shortest path distance, while the boundary graph G_B is dedicated to recording the shortest path distances of pairs of boundary vertices in the original subgraph. Section 5 will demonstrate how to compute shortest paths between arbitrary vertex pairs by jointly utilizing both parties' indexes. Then the HB-Index is formally defined as follows.

*Definition 4.3 (**HB-Index**).* The HB-Index of subgraph $G(V, E)$ is defined as the combination of the subgraph-index HB_S and the boundary graph G_B.

Party	ID	Label Entries
P_0	v_0	$\{v_0, 0\}, \{v_1, 3\}$
	v_1	$\{v_1, 0\}$
	v_2	$\{v_2, 0\}, \{v_1, 2\}$
	v_6	$\{v_6, 0\}, \{v_1, 1\}, \{v_2, 1\}$
	v_7	$\{v_7, 0\}, \{v_1, 1\}, \{v_0, 2\}$
	v_8	$\{v_8, 0\}, \{v_0, 1\}, \{v_7, 1\}, \{v_1, 2\}$
P_1	v_3	$\{v_3, 0\}$
	v_4	$\{v_4, 0\}, \{v_3, 1\}$
	v_5	$\{v_5, 0\}, \{v_3, 3\}, \{v_4, 3\}$
	v_9	$\{v_9, 0\}, \{v_3, 1\}, \{v_4, 1\}, \{v_5, 2\}$
	v_{10}	$\{v_{10}, 0\}, \{v_5, 1\}, \{v_9, 1\}, \{v_3, 2\}, \{v_4, 2\}$

Fig. 4. The label entries of subgraph indexes of subgraphs in Figure 2.

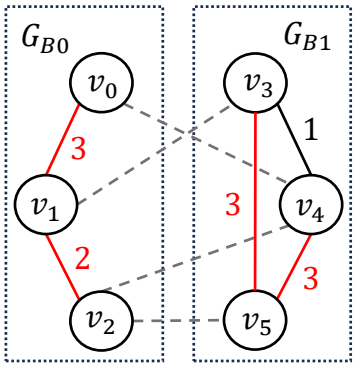


Fig. 5. The boundary graphs of subgraphs in Figure 2.

Example 2. *Figure 4 and Figure 5 illustrate the HB-Index structures for the subgraphs in Figure 2:*

(1) Figure 4 shows the subgraph indices HB_S, where each vertex label records shortest path distances to other vertices in the same subgraph. White rows correspond to G_0, and shaded rows to G_1.

(2) Figure 5 depicts a feasible implementation of the boundary graphs G_{B0} and G_{B1}, where weighted edges connect boundary vertex pairs. Each edge weight equals the shortest path distance between the corresponding vertices in the original subgraph.

4.2 Index Construction

Our method requires the construction of two core index structures: the subgraph index HB_S and the boundary graph G_B. The HB_S can be directly built on subgraphs using an improved PSL method, while the boundary graph G_B requires more sophisticated design.

According to Definition 4.2, the boundary graph needs to accurately record the shortest distance information between all boundary vertex pairs. Although the most straightforward approach would be to compute and store the shortest distances for all boundary vertex pairs using the H_S index, this would result in an edge set of size $O({V_B}^2)$, which is completely infeasible for large-scale graph data.

Through in-depth research, we discovered that the path implication principle can be used to optimize the boundary graph. Specifically:

Lemma 4.4 (Boundary Path Implication). *Given a boundary graph $G_B(V_B, E_B, W)$, for any boundary vertex pair $(s, t) \in V_B \times V_B$, if its shortest path $p(s, t)$ in subgraph G passes through at least another boundary vertex $w \in V_B \setminus \{s, t\}$, then:*

- *Even if $e(s, t) \notin E_B$, $dist_{G_B}(s, t) = dist_G(s, t)$.*

Proof. Let $p(s, t)$ be the shortest path from s to t. If $p(s, t)$ does not pass through any other boundary vertices, then by Definition 4.2, boundary graph must contain the direct edge $e(s, t)$, in which case $dist_{G_B}(s, t) = W(s, t) = dist_G(s, t)$.

If $p(s, t)$ passes through $w \in V_B \setminus \{s, t\}$, according to the optimal substructure property of shortest paths, the path can be decomposed as:

$$p(s, t) = p(s, w) \bowtie p(w, t)$$

where $p(s, w)$ and $p(w, t)$ are the shortest paths in their respective subpaths. By recursively applying this argument, we know that:

- When the subpath does not pass through other boundary vertices, its direct edge must exist in E_B
- When the subpath passes through other boundary vertices, it can be further decomposed until basic paths are obtained

Therefore, there must exist a sequence of subpaths $s \rightarrow w \rightarrow t$ in the boundary graph such that:

$$dist_{G_B}(s,t) = dist_{G_B}(s,w) + dist_{G_B}(w,t) = dist_G(s,t).$$

□

Based on Lemma 4.4, we propose the following construction principles to optimize size and efficiency:

- **Minimal Edge Set:** Include an edge (u,v) in the boundary graph only if the shortest path between u and v in G does not traverse other boundary vertices. This avoids redundant connections when intermediate boundary nodes already provide shortest paths.
- **Accurate Distance Preservation:** Assign weights $W(u,v) = dist_G(u,v)$ to all included edges, ensuring that $dist_{G_B}(u,v)$ matches $dist_G(u,v)$.

Construction Algorithm. Based on the previous analyses, we give the construction algorithms for the HB-Index.

Algorithm 2: `IndexConstruction`

Input: Graph $G(V,E)$, boundary node set $V_B \subseteq V$
Output: Subgraph index HB_S, boundary flags F

1 Initialize $L^0(u) \leftarrow \{(u,0)\}$ and $F^0(u) \leftarrow \{\texttt{false}\}, \forall u \in V$;
2 **foreach** *edge* $(u,v) \in E$ **do**
3 **if** $r(u) > r(v)$ **then**
4 $L^1(v) \leftarrow L^1(v) \cup \{(u,1)\}$, append `false` to $F^1(v)$;
5 **else**
6 $L^1(u) \leftarrow L^1(u) \cup \{(v,1)\}$, append `false` to $F^1(u)$;
7 **for** $d \leftarrow 2; L^{d-1} \neq \emptyset; d \leftarrow d+1$ **do**
8 **for** *each* $u \in V$ *in parallel* **do**
9 **for** *each* $v \in N(u)$ *and* $(w,\_) \in L^{d-1}(v)$ **do**
10 **if** $r(w) \geq r(u)$ ***and*** $\texttt{Query}(w,u,L^{<d}) > d$ **then**
11 $L^d(u) \leftarrow L^d(u) \cup \{(w,d)\}$;
12 `flag`$\leftarrow w \in V_B$, append `flag` to $F^d(u)$;
13 $HB_S = \cup_{v \in V} L(v)$;
14 **return** (HB_S, F);

Algorithm 2 demonstrates the complete process of subgraph index (HB_S) construction. Its inputs are the graph $G(V,E)$ and the boundary node set V_B. The outputs include a 2-hop index HB_S and a flag storage structure F whose size matches the index. Each flag in this structure indicates whether the corresponding 2-hop index entry is generated through scaling via a boundary nodes.

It initializes the labels and flags for all nodes (line 1), then propagates labels from higher-ranked to lower-ranked nodes based on directly connected edges (line 2-6). In the expansion phase, the algorithm iteratively collects candidate nodes from neighbors, verifying their rank and distance conditions, and adds valid candidates to the index, updating their boundary flags as needed (line

7-12). The process continues until no more labels can be propagated, and the final subgraph index HB_S and the boundary flags F are returned(line 13-14).

Algorithm 3: `BoundaryGraphConstruction`

Input: Boundary node set V_B, distance index L, boundary flags F
Output: Boundary graph G_B

```
1 Initialize G_B ← ∅;
2 for each u ∈ V_B do
3     for (w, d) ∈ L(u) with corresponding flag ∈ F(u) do
4         if flag = false and w ∈ V_B then
5             G_B ← G_B ∪ {(u, w, d)};
6 return G_B;
```

Algorithm 3 demonstrates the construction process of the boundary graph (G_B). The algorithm takes the boundary node set V_B, the subgraph index HB_S, and the corresponding flag F as inputs, and outputs the boundary graph G_B that preserves complete shortest path distance information.

The process begins by initializing an empty graph G_B (line 1). For each boundary node $u \in V_B$, the algorithm iterates through the index pairs $(w, d) \in L(u)$ and the corresponding flag values in $F(u)$ (lines 2-3). It then checks two conditions: (1) the node w must be a boundary node, and (2) the connection between u and w must represent a direct boundary path, i.e., the flag is `false`, (line 4). If both conditions are satisfied, the edge (u, w, d) is added to G_B (line 5). This ensures that G_B contains only the relevant boundary connections without redundancy.

Example 3. *Figure 5 illustrates the boundary graph construction process described in Algorithm 3 for the two subgraphs in Figure 2. In G_{B0}, the edge $e(v_0, v_2)$ is excluded because its shortest path passes through another boundary vertex (v_1): $v_0 \to v_1 \to v_2$ with a total distance of 5. By contrast, G_{B1} retains all edges between boundary vertices, as no intermediate boundary nodes appear on their shortest paths.*

Time complexity. Assuming that m is the maximal number of edges of subgraph G and δ is the maximal number of label entries of interior vertices. The time complexity of index construction (Algorithm 2) is $O(\delta^2 \cdot m)$. The time complexity of boundary graph construction (Algorithm 3) is $O(\delta \cdot V_B)$.

Index Update. Building on prior work [49], our method can be extended to support edge insertions, deletions, and node additions. When a new node is inserted, it is assigned the lowest rank, and each incident edge is handled as a regular edge insertion. Updating the index requires re-propagating labels for affected vertices, with complexity $O(n \cdot \delta^2 - n' \cdot \delta' \cdot \delta)$, where δ is the maximum label size across all vertices and n' counts unaffected vertices. Afterward, boundary vertices are updated by re-running Algorithm 3 with complexity $O(|V_B| \cdot \delta)$.

5 Online Phase

In this section, we present the online phase of `PrivHop`, which securely processes shortest path distance queries using the precomputed indices. We describe how to evaluate queries with the HB-Index, implement the privacy-preserving shortest path distance algorithm, and apply optimizations for efficiency.

5.1 Index-Based Query Decomposition

Building on the two-party indices HB_0 and HB_1, we now describe how to process shortest path distance queries over the global graph $\mathcal{G}$. For clarity, we first introduce the *Combined Boundary Graph* as a conceptual tool to facilitate algorithm design.

*Definition 5.1 (**Combined Boundary Graph**).* Given a graph $\mathcal{G}(V, E)$ partitioned into two parties as $\{G_i(V_i, E_i)\}_{i=0}^{1} \cup E_{cut}$, and their respective HB-Indices HB_0 and HB_1, the combined boundary graph $G_{CB}(V_{CB}, E_{CB})$ is defined as:

- $V_{CB} = V_{B0} \cup V_{B1}$
- $E_{CB} = E_{B0} \cup E_{B1} \cup E_{cut}$

Though G_{CB} provides a convenient abstraction for reasoning, it is never materialized in plaintext. Instead, the boundary graphs G_{B0}, G_{B1}, and E_{cut} are maintained in secret-shared form to preserve privacy.

Lemma 5.2 (Distance Preservation in G_{CB}). *For any $(u, v) \in V_{CB} \times V_{CB}$, the shortest path distance satisfies:*

$$dist_{G_{CB}}(u, v) = dist_{\mathcal{G}}(u, v).$$

Proof. We consider all types of shortest paths in G_{CB}:

(1) **Intra-boundary paths**: If $u, v \in V_{Bi}$ and $p(u, v)$ lies entirely within G_{Bi}:

$$p(u, v) = u \overset{G_{Bi}}{\rightsquigarrow} v$$

By Definition 4.2, $W_{G_{CB}}(u, v) = dist_{G_i}(u, v)$. Thus:

$$dist_{G_{CB}}(u, v) = dist_{G_{Bi}}(u, v) = dist_{\mathcal{G}}(u, v).$$

(2) **Single cut-edge paths**: If $u \in V_{B0}, v \in V_{B1}$ and $(u, v) \in E_{cut}$:

$$p(u, v) = u \overset{E_{cut}}{\rightarrow} v$$

Then $W_{G_{CB}}(u, v) = dist_{\mathcal{G}}(u, v)$ by construction.

(3) **Multi-hop inter-partition paths**: For general paths crossing partitions multiple times,

$$p(u, v) = u \overset{G_{B0}}{\rightsquigarrow} w_1 \overset{E_{cut}}{\rightarrow} w_2 \overset{G_{B1}}{\rightsquigarrow} w_3 \overset{E_{cut}}{\rightarrow} \dots \overset{G_{Bi}}{\rightsquigarrow} v$$

By the optimal substructure property of shortest paths:

$$dist_{G_{CB}}(u, v) = \sum_{k} dist_{G_{Bi}}(w_{k-1}, w_k) + dist_{E_{cut}}(w_k, w_{k+1})$$

Since each $dist_{G_{Bi}}(\cdot, \cdot)$ and $dist_{E_{cut}}(\cdot, \cdot)$ preserves global distances by Definition 4.2 and E_{cut}, it follows recursively:

$$dist_{G_{CB}}(u, v) = dist_{\mathcal{G}}(u, v).$$

□

Query Processing Procedure. Given a query $q(s, t)$, we distinguish two cases:

• **Case 1: Cross-Partition Query**. Without loss of generality, assume $\mathcal{P}(s) = P_0$ and $\mathcal{P}(t) = P_1$. Here, any shortest path must traverse at least two boundary vertices. Define $V_B^s = \{v \in V_{B0} \mid L(v) \cap L(s) \neq \emptyset\}$, $V_B^t = \{v \in V_{B1} \mid L(v) \cap L(t) \neq \emptyset\}$, the shortest path distance can be computed by

$$dist_{\mathcal{G}}(s, t) = \min_{u \in V_B^s, v \in V_B^t} L(s)[u] + dist_{G_{CB}}(u, v) + L(t)[v]. \tag{2}$$

This formulation reduces global queries to boundary graph computations plus local label lookups.

Example 4. *For $q(v_8, v_9)$, we find $V_B^s = \{v_0, v_1, v_2\}$, $V_B^t = \{v_0, v_3, v_6\}$. Using labels and G_{CB}, we compute:*

$$dist_{\mathcal{G}}(v_8, v_9) = \min_{u \in V_B^s, v \in V_B^t} L(v_8)[u] + dist_{G_{CB}}(u, v) + L(v_9)[v] = 3.$$

• **Case 2: Intra-Partition Query**. Without loss of generality, assume $\mathcal{P}(s) = P_0$ and $\mathcal{P}(t) = P_0$. Here, two candidate paths exist: (1) purely internal paths; and (2) paths crossing into the other partition and returning. Define:

$$\begin{aligned} d_1 &= \min_{w \in L(s) \cap L(t)} L(s)[w] + L(t)[w], \\ d_2 &= \min_{u,v \in V_{B0}} dist_{G_0}(s, u) + dist_{G_{CB}}(u, v) + dist_{G_0}(v, t), \\ dist_{\mathcal{G}}(s, t) &= \min(d_1, d_2). \end{aligned} \tag{3}$$

Example 5. *Consider query $q(v_8, v_6)$, The common vertices in $L(v_8) \cap L(v_6)$ are only v_1. Thus:*

$$d_1 = L(v_8)[v_1] + L(v_6)[v_1] = 2 + 1 = 3.$$

The boundary vertices in V_{B0} are $\{v_0, v_1, v_2\}$. We compute:

- *Distances from v_8 to boundary nodes: $dist_{G_0}(v_8, v_0) = 1$, $dist_{G_0}(v_8, v_1) = 2$, $dist_{G_0}(v_8, v_2) = 4$*
- *Distances from v_6 to boundary nodes: $dist_{G_0}(v_6, v_0) = 4$, $dist_{G_0}(v_6, v_1) = 1$, $dist_{G_0}(v_6, v_2) = 1$*
- *Cross-boundary distances $dist_{G_{CB}u,v \in V_{B0}}(u, v)$ between boundary nodes*

$$\begin{aligned} d_2 &= \min_{u,v \in V_{B0}} dist_{G_0}(v_8, u) + dist_{G_{CB}}(u, v) + dist_{G_0}(v, v_6) \\ &= dist_{G_0}(v_8, v_1) + dist_{G_{CB}}(v_1, v_1) + dist_{G_0}(v_1, v_6) = 3 \end{aligned}$$

Finally:

$$dist_{\mathcal{G}}(v_8, v_6) = \min\{d_1, d_2\} = 3.$$

Privacy-Preserving Computation. Since G_{CB} is a conceptual construct and not stored explicitly, computing $dist_{G_{CB}}(u, v)$ requires a secure protocol. Global distance evaluation also depends on the distances from s and t to their respective boundary sets, denoted D_{S2B} and D_{T2B}, which are precomputed locally using HB_S. This preprocessing step has a time complexity of $O(V_B \cdot \delta)$, where V_B is the number of boundary vertices and δ is the maximum label size.

To perform secure global shortest path computation (Equations 2 and 3), we design an *Oblivious Bellman-Ford* algorithm (Section 5.2) that operates on secret-shared inputs: G_{B0}, G_{B1}, D_{S2B}, D_{T2B}, and E_{cut}.

In cross-partition queries, the target vertex identifier t needs to be revealed to the counterpart party to compute D_{T2B}. This disclosure is limited to the query endpoint and does not expose any information about graph topology or intermediate computations, thereby preserving overall privacy.

5.2 Oblivious Bellman

This section introduces our Oblivious Bellman algorithm for secure shortest path distance computation over secret-shared boundary graphs.

We begin with two cryptographic primitives essential to our protocol: the *Oblivious Extended Permutation* (OEP) [35] and *Oblivious Group Aggregation* (OGA) [9], to protect the boundary graph topology and its vertex degree information.

OEP. The primitive $\mathcal{F}_{OEP}$ applies an extended permutation π to a secret-shared vector $\langle T_{in} \rangle$:

$$\langle T_{out} \rangle \leftarrow \mathcal{F}_{OEP}(\langle T_{in} \rangle, Id_{in}, Id_{out}) \tag{4}$$

Here, (Id_{in}, Id_{out}) defines π, mapping identifiers from the input domain to the output domain. Each element in T_{out} is a random reshare of a permuted element from T_{in}.

OGA. The primitive $\mathcal{F}_{OGA}$ performs secure aggregation on a secret-shared input vector using a binary operator $\boxplus$:

$$\langle T_{out} \rangle \leftarrow \mathcal{F}_{OGA}(\langle T_{in} \rangle, \Gamma, \boxplus) \tag{5}$$

Here, Γ is a group identifier vector where contiguous identical values define aggregation groups. For each group, $\mathcal{F}_{OGA}$ applies $\boxplus$ (e.g., min, $\sum$) across all group elements and writes the result to the first position of the corresponding segment in T_{out}.

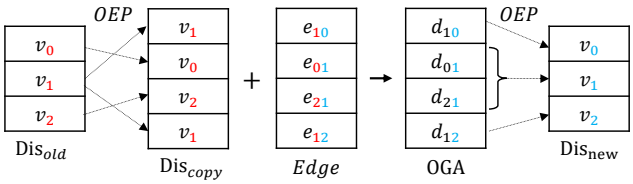


Fig. 6. Oblivious relaxation in the Bellman iteration.

Oblivious Bellman Overview. Algorithm 4 details the protocol. It performs iterative relaxation over secret-shared boundary graphs G_{B0}, G_{B1}, and cross-partition edges E_{cut}. Each iteration consists of two phases: 1. Internal boundary relaxation: updates local distances within G_{B0} and G_{B1} (lines 4–5). 2. Cross-partition relaxation: updates distances across partitions using E_{cut} (lines 6–7).

The `Relax` function performs the distance relaxation through four sequential operations (Fig. 6): (1) Oblivious Extend Protocol (OEP) propagates vertex distances to outgoing edges (line 12); (2) Edge-weighted distance increments are computed (line 13); (3) Oblivious Aggregation (OGA) aggregates tentative updates at destination vertices using min (line 14); (4) Finally, OEP writes back refined distances (V^{dst}) while preserving privacy through minimum-value retention (lines 15-17). This process repeats for a fixed number of iterations ($N = |V_{B0}| + |V_{B1}|$) or until convergence via the `PruneLoopNum` optimization (see Section 5.3).

The final distances D_{S2B} (from s to all boundary vertices) are added to the precomputed D_{T2B} (distances from boundary vertices to t) to form a combined distance vector (line 8). The global shortest path distance D_{min} is then determined as the minimum value of this vector (line 9).

5.3 Privacy-Aware Iteration Pruning

While OEP and OGA eliminate ORAM overhead, the iteration count of Oblivious Bellman remains linear in graph size. To address this, we propose a round pruning mechanism that uses precomputed target-to-boundary distances (D_{T2B}) and dynamically updated source-to-boundary distances (D_{S2B}) to compute an upper bound on iterations. This dual-distance constraint ensures early termination without compromising correctness.

Lemma 5.3 (Iteration Bound). *Given a graph $\mathcal{G}(V, E)$, let $D_{S2B}^{(l)}$ be the source-to-boundary distances after l iterations and D_{T2B} be the fixed target-to-boundary distances. The upper bound on the number of remaining relaxation rounds required for the Oblivious Bellman algorithm to converge after l iterations $LoopNum^{(l)}$ satisfies:*

$$LoopNum^{(l)} \leq \min\{D_{S2B}^{(l)} + D_{T2B}\} - \min\{D_{S2B}^{(l)}\} - \min\{D_{T2B}\} \tag{6}$$

Proof. Every boundary path $p(s, t)$ traverses some boundary vertex $u \in V_B$, so:

$$dist(s, t) \leq \min_{u \in V_B}\{D_{S2B}^{(l)}(u) + D_{T2B}(u)\}. \tag{7}$$

Algorithm 4: ObliviousBellman

Input : P_0 provides $G_{B0}(V_0, E_0), D_{S2B}, D_{T2B}$;
P_1 provides $G_{B1}(V_1, E_1), E_{cut}$;
N is the total number of boundary vertices.
Output: P_0 obtains the global shortest distance D_{min}.

Initialize: $V_0.dis \leftarrow \langle D_{S2B} \rangle$, $V_1.dis \leftarrow \langle \infty \rangle$, $n \leftarrow N$;
for $l = 1$ *to* n **do**
  $\langle V_0.dis \rangle \leftarrow \texttt{RELAX}(V_0, E_0)$;
  $\langle V_1.dis \rangle \leftarrow \texttt{RELAX}(V_1, E_1)$;
  $\langle V_1.dis \rangle \leftarrow \texttt{RELAX}(V_0, E_{cut})$;
  $\langle V_0.dis \rangle \leftarrow \texttt{RELAX}(V_1, E_{cut})$;
  $n \leftarrow \min\{n, l + \texttt{PruneLoopNum}(l, V_0, D_{T2B})\}$;
$\langle D_{final} \rangle \leftarrow \langle V_0.dis \rangle + \langle D_{T2B} \rangle$;
$\langle D_{min} \rangle \leftarrow \min(\langle D_{final} \rangle)$;
return D_{min};

Function $\texttt{RELAX}(V, E)$**:**
  $\langle V^{src} \rangle \leftarrow \texttt{OEP}(\langle V.dis \rangle, V.id, E.src)$;
  $\langle D \rangle \leftarrow \langle V^{src} \rangle + \langle E.weight \rangle$;
  $\langle D' \rangle \leftarrow \texttt{OGA}(\langle D \rangle, E.dst, \min)$;
  $\langle V^{dst} \rangle \leftarrow \texttt{OEP}(\langle D' \rangle, E.dst, V.id)$;
  $\langle V.dis \rangle \leftarrow \min\{\langle V^{dst} \rangle, \langle V.dis \rangle\}$;
  return $\langle V.dis \rangle$;

The minimal source-to-boundary plus target-to-boundary distances yield a lower bound:.

$$dist(s,t) \geq \min\{D_{S2B}^{(l)}\} + \min\{D_{T2B}\}. \tag{8}$$

Equality holds only if the same boundary vertex minimizes both terms.

Thus, the number of remaining iterations is at most the difference between these bounds:

$$LoopNum^{(l)} \leq \min\{D_{S2B}^{(l)} + D_{T2B}\} - \min\{D_{S2B}^{(l)}\} - \min\{D_{T2B}\}. \tag{9}$$

This ensures the algorithm terminates once all potential improvements are exhausted, yielding correct shortest distances between all boundary vertices. □

In MPC, both parties must agree on remaining rounds. Directly revealing it risks privacy leakage, so we apply differential privacy to protect privacy. We begin by the sensitivity analysis.

- Changing a single edge weight affects $\min(D_{S2B}^{(l)})$, $\min(D_{T2B})$ by at most ±1, and $\min(D_{S2B}^{(l)} + D_{T2B})$ by ±2.
- These changes are coherent: if one increases (or decreases), others change likewise or remain unchanged.

Thus, the global sensitivity of $LoopNum^{(l)}$[4] is $\Delta = 2$, according to Definition 2.6

[4]A detailed proof is provided in https://github.com/HuiZ-W/PrivHop/Supplementary_Materials_for_Paper_856.pdf

Differential Privacy Mechanism. We use the Laplace mechanism with truncation to ensure convergence:

$$LoopNum_{DP}^{(l)} = \max\left(LoopNum^{(l)} + \mathrm{Lap}(2/\epsilon) + \frac{2}{\epsilon}\log(\frac{1}{\delta}),\quad LoopNum^{(l)}\right) \tag{10}$$

where N is the number of vertices, $\frac{2}{\epsilon}\log(\frac{1}{\delta})$ is a correction term.

Theorem 5.4 (Privacy Guarantee). *The above mechanism satisfies (ϵ, δ)-differential privacy.*

Proof. Let $f(\mathcal{G}) = LoopNum^{(l)}$ with global sensitivity $\Delta = 2$. The Laplace mechanism gives $M_1(\mathcal{G}) = f(\mathcal{G}) + \mathrm{Lap}(2/\epsilon)$ which satisfies ϵ-DP.

Now define the truncated mechanism:

$$M(\mathcal{G}) = \max\left(M_1(\mathcal{G}) + \frac{2}{\epsilon}\log(\frac{1}{\delta}),\ f(\mathcal{G})\right)$$

Let $X \sim \mathrm{Lap}(2/\epsilon)$. The failure event is when:

$$M_1(\mathcal{G}) + \frac{2}{\epsilon}\log(\frac{1}{\delta}) < f(\mathcal{G})$$

which occurs when:$X < -\frac{2}{\epsilon}\log(\frac{1}{\delta})$

The probability of this event is:

$$\mathbb{P}\left[X < -\frac{2}{\epsilon}\log(\frac{1}{\delta})\right] = \frac{1}{2}\exp\left(-\frac{\epsilon}{2}\cdot\frac{2}{\epsilon}\log(\frac{1}{\delta})\right) = \frac{\delta}{2}$$

When this event occurs, the output is exactly $f(\mathcal{G})$, which may reveal sensitive information. By the truncation lemma for the Laplace mechanism, adding appropriate truncation ensures that the mechanism satisfies (ϵ, δ)-differential privacy. □

Theorem 5.5 (Correctness Guarantee). *The mechanism guarantees $LoopNum_{DP}^{(l)} \geq LoopNum^{(l)}$ always, ensuring Bellman-Ford algorithm convergence.*

Proof. By construction:

$$LoopNum_{DP}^{(l)} = \max\left(LoopNum^{(l)} + \mathrm{Lap}(2/\epsilon) + \frac{2}{\epsilon}\log(\frac{1}{\delta}),\quad LoopNum^{(l)}\right) \geq LoopNum^{(l)}$$

The actual number of execution rounds is always sufficient for convergence, regardless of the noise realization. □

By employing this differentially private truncation strategy, we bound the execution iterations while rigorously maintaining (ϵ, δ)-differential privacy and guaranteeing absolute algorithm convergence.

The implementation details of our dynamic round optimization strategy are presented in Algorithm 5. Since each pruning operation requires allocating a privacy budget, excessive pruning would lead to reduced budget per operation and consequently introduce excessive noise that undermines the pruning effectiveness. Therefore, we perform pruning only once when the iteration count l reaches $\log(N)$ rounds.The algorithm recalculates the theoretical convergence rounds using Equation 5.3 and injects Laplace noise along with a deterministic compensation term according to Equation 10. This approach achieves dynamic optimization of iteration rounds while ensuring (ϵ, δ)-differential privacy and maintaining algorithmic correctness, thereby striking an optimal balance between privacy protection and computational efficiency.

Time Complexity. The Oblivious Bellman algorithm consists of two components: `Relax` and `PruneLoopNum`.

Algorithm 5: PruneLoopNum

Input : Current Loop ℓ, $\langle D_{S2B} \rangle$ and $\langle D_{T2B} \rangle$
Output : Pruned LoopNum ℓ_{DP}

if $\ell == \lceil \log(N) \rceil$ **then**
  $\langle D_{cur} \rangle \leftarrow \min\{\langle D_{S2B} \rangle + \langle D_{T2B} \rangle\}$;
  $\langle D_{lb} \rangle \leftarrow \min\{\langle D_{S2B} \rangle\} + \min\{\langle D_{T2B} \rangle\}$;
  $M \leftarrow \langle D_{cur} \rangle - \langle D_{lb} \rangle$;
  $R \leftarrow \texttt{Reveal}(M)$;
  $\ell_{DP} \leftarrow \max\{R + Lap(2/\varepsilon) + (2/\varepsilon)\log(\frac{1}{\delta}), R\}$;
  return $\lceil \ell_{DP} \rceil$;
return ∞

- Relax involves two OEPs and one OGA per call, with secure operation complexity $O(|V|+|E|)$ and round complexity $O(\log |E|)$, with V and E denoting vertices and edges in boundary graphs.
- PruneLoopNum has a secure operation complexity of $O(|V|)$ and constant round complexity.

Each iteration executes 4 Relax, yielding per-iteration complexity $O(|V| + |E|)$ operations and $O(\log |E|)$ rounds. PruneLoopNum is executed only once after $\log(V)$rounds. This pruning step reduces the total number of iterations from $|V|$ to k ($k \ll |V|$), resulting in:

$$\textbf{Total secure ops: } O((\log V + k)(|V| + |E|))$$
$$\textbf{Total rounds: } O((\log V + k)\log |E|)$$

This improves efficiency over the classic Bellman-Ford, which requires $|V|$ iterations.

6 Experiments

This section evaluates the performance of our method via extensive experiments: Section 6.1 describes the experimental setup (datasets, evaluation protocols, and baselines), while Sections 6.2 and 6.3 present the empirical results, to validate the efficiency and effectiveness of our PrivHop.

6.1 Experiment Setup

Dataset. In our experimental evaluation, we employed 8 real-world datasets with varying scales. All datasets are available at Network Repository[5], and their statistical characteristics are summarized in Table 2. To support the two-party computation scenario in our experiments, each dataset was partitioned into two approximately equal-sized subsets using the KaHIP graph partitioner [44]. The resulting partition statistics for each dataset are summarized in Table 3.

Algorithms. We compare the following algorithms:

(1) **Privacy-Preserving Dijkstra (PPD)** [36]: a privacy-preserving Dijkstra algorithm that combines d-normalization with parallel priority queues for theoretically optimal performance in MPC settings.
(2) **Privacy-Preserving Bellman (PPB)** [6]: a Sharemind-based Bellman-Ford variant with parallel oblivious read operations, demonstrated on graphs up to 10,000 nodes.
(3) **PP-Dijkstra On Boundary Graph (B-PPD)**: Applies PP-Dijkstra to the boundary graph preprocessed by our method to evaluate its performance on the simplified graph structure.

[5]http://networkrepository.com/index.php

Table 2. Real-world graphs statistics.

Alias	Dataset	$\|V\|$	$\|E\|$	d_{avg}	Type
SK	SocKarate	34	78	4	Social Network
SD	SocDolphins	62	159	5	Social Network
SW	SocWikiVote	889	3K	6	Social Network
SH	SocHamster	2K	17K	13	Social Network
SE	SocEpinions	27K	100K	7	Social Network
WS	WebStandford	282K	2M	16	Web Graph
IT	WebIT2004	509K	7M	28	Web Graph
TS	TechSkitter	1.7M	11.1M	13	Tech. Networks

Table 3. Partition statistics of real-world graphs.

Alias	Subgraph A (P_0)			Subgraph B (P_1)			Cutting Edges
	$\|V\|$	$\|E\|$	$\|V_B\|$	$\|V\|$	$\|E\|$	$\|V_B\|$	
SK	17	34	6	17	32	7	12
SD	29	61	12	33	70	14	28
SW	417	1021	113	472	1483	135	410
SH	1291	10.7K	323	1135	4562	390	1293
SE	15.9K	81.1K	3983	10.6K	10.5K	4661	8454
WS	132.1K	1.04M	15.9K	149.8K	909.7K	18.1K	40.8K
IT	250.5K	3.54M	4888	255.8K	3.63M	7139	11.5K
TS	846.9K	5.23M	62.5K	847.6K	5.51M	95.4K	346.7K

(4) **PP-Bellman On Boundary Graph (**`B-PPB`**)**: Applies `PP-Bellman` to the boundary graph preprocessed by our method to evaluate its performance on the simplified graph structure.
(5) `PrivHop`: Our two-phase solution constructs a 2-hop index and reduces the graph size in the offline phase, while enhancing query efficiency in the online phase by applying pruning techniques to minimize iterations.

Since neither algorithm's original codebase is publicly available, we faithfully reimplemented them based on the algorithmic descriptions in their respective papers.

Environment. Experiments were conducted on a Linux server with 128 Intel Xeon Platinum 8358 CPUs (2.60 GHz). Participants were isolated using Linux network namespaces and interconnected via virtual switches. A simulated LAN was configured with `tc`, providing 4 Gbps bandwidth and 1 ms latency. The differential privacy parameters were set to $\varepsilon = 1$ and $\delta = 1/N$. All algorithms were implemented in C++17 with `-O3` optimization and executed with 20 threads for consistency.

6.2 Experiment Results

Exp-1: Index Evaluation.

We evaluate the index construction performance in a single-threaded setting, with results reported in Table 4. Three metrics are considered: index construction time, index storage size, and boundary graph scale, along with overall boundary graph statistics (WBG Info). The results show that our method scales well to million-node datasets, with index construction time on the order of 10^3s and storage overhead around 1 GB, demonstrating strong practical feasibility.

Notably, our implication rules reduce the boundary graph edge count by up to 20× compared to the original graph, substantially lowering the encrypted computation workload and significantly improving both runtime and communication efficiency for subsequent queries.

Exp-2: Query Efficiency. This part presents a systematic comparison of query efficiency among different algorithms. The experimental setup is as follows: For each test dataset, we randomly selected 1,000 query pairs and recorded their average query time. For query tasks that were difficult to complete on large-scale datasets, we employed a time extrapolation method based on partial query results (specific approach: scaling proportionally according to query size using the average time of completed query rounds). The experimental results are shown in Figure 7. Algorithms unable to complete within a reasonable time (10^9 seconds) were considered to have timed out and are reported at the upper bound. Our algorithm demonstrates substantial advantages in query efficiency, achieving 2–6 orders of magnitude speedup over baseline methods on most datasets. This performance gain is attributed to two key design features:

(1) **Precomputed index architecture**: The HB-Index, constructed offline, significantly reduces the graph size for online query processing.

Table 4. Statistics of our HB-Index.

Alias	Subgraph A(P_0)			Subgraph B(P_1)			WBG Info	
	Index Size	Index Time	#E	Index Size	Index Time	#E	$\|V\|$	$\|E\|$
SK	0.21KB	0.86ms	9	0.21KB	0.57ms	9	13	30
SD	0.64KB	2.02ms	22	0.97KB	2.18ms	29	26	159
SW	25.79KB	42.76ms	294	30.32KB	35.54ms	466	248	1170
SH	0.16MB	0.14s	2564	0.06MB	0.03s	1377	713	5234
SE	6.36MB	3.43s	92.2K	0.34MB	0.54s	5.4K	8644	39.2K
WS	8.27MB	11.27s	47.6K	29.97MB	24.89s	110.8K	34.0K	199.2K
IT	40.93MB	38.21s	22.1K	33.56MB	28.99s	323K	12.1K	356.6K
TS	1.09GB	1321.6s	1.9M	0.76GB	837.3s	1.3M	157.9K	3.5M

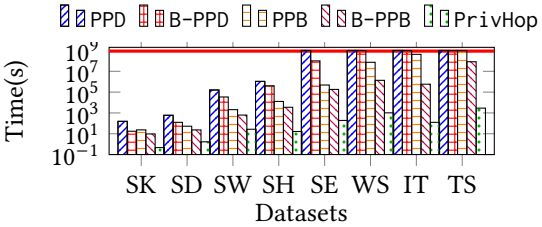


Fig. 7. Query efficiency comparison of the algorithms.

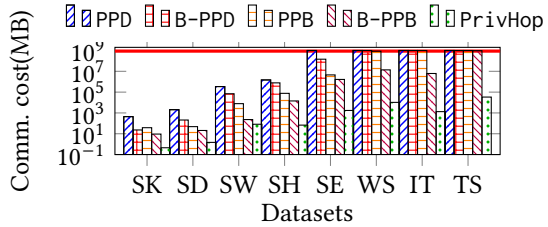


Fig. 8. Communication cost comparison of the algorithms.

(2) **Security pruning strategy**: The dynamic round control mechanism based on differential privacy effectively minimizes the number of relaxation operations in secure environments.

Table 5. Running time (in sec) of all MPC shortest path distance methods.

Method	Phase	SK	SD	SW	SH	SE	WS	IT	TS
PPD	Pre.	9.13×10^{0}	1.65×10^{1}	9.24×10^{2}	1.04×10^{4}	-	-	-	-
	Iterations	1.55×10^{2}	5.93×10^{2}	1.60×10^{5}	1.07×10^{6}	-	-	-	-
	Total	2.46×10^{2}	6.10×10^{2}	1.60×10^{5}	1.08×10^{6}	-	-	-	-
B-PPD	Pre.	4.29×10^{0}	8.21×10^{0}	1.34×10^{2}	1.12×10^{3}	8.63×10^{4}	-	-	-
	Iterations	1.75×10^{1}	1.25×10^{2}	3.42×10^{4}	4.07×10^{5}	1.00×10^{8}	-	-	-
	Total	2.18×10^{1}	1.33×10^{2}	3.43×10^{4}	4.08×10^{5}	1.00×10^{8}	-	-	-
PPB	Pre.	6.97×10^{-1}	9.51×10^{-1}	8.47×10^{0}	4.37×10^{1}	2.57×10^{2}	4.64×10^{3}	1.61×10^{4}	2.70×10^{4}
	Iterations	2.36×10^{1}	5.29×10^{1}	2.06×10^{3}	1.27×10^{4}	4.81×10^{5}	7.32×10^{7}	4.26×10^{8}	2.52×10^{9}
	Total	2.42×10^{1}	5.39×10^{1}	2.07×10^{3}	1.27×10^{4}	4.82×10^{5}	7.32×10^{7}	4.26×10^{8}	2.52×10^{9}
B-PPB	Pre.	5.02×10^{-1}	8.04×10^{-1}	3.69×10^{0}	1.47×10^{1}	2.72×10^{2}	5.34×10^{2}	7.31×10^{2}	1.05×10^{4}
	Iterations	9.64×10^{0}	2.34×10^{1}	6.24×10^{2}	3.49×10^{3}	1.82×10^{5}	1.35×10^{6}	5.77×10^{5}	8.31×10^{7}
	Total	1.01×10^{1}	2.42×10^{1}	6.28×10^{2}	3.50×10^{3}	1.82×10^{5}	1.35×10^{6}	5.78×10^{7}	8.31×10^{7}
PrivHop	Init.	1.70×10^{-5}	3.60×10^{-5}	2.85×10^{-4}	1.59×10^{-3}	6.82×10^{-2}	1.26×10^{-2}	6.67×10^{-3}	7.32×10^{-1}
	Pre.	4.67×10^{-1}	7.76×10^{-1}	3.45×10^{0}	1.42×10^{1}	2.52×10^{2}	5.10×10^{2}	7.07×10^{2}	1.00×10^{4}
	Iterations	4.69×10^{-1}	1.68×10^{0}	2.70×10^{1}	1.65×10^{1}	1.90×10^{2}	1.02×10^{3}	1.24×10^{2}	2.90×10^{3}
	Total	9.36×10^{-1}	2.45×10^{0}	3.05×10^{1}	3.08×10^{1}	4.43×10^{2}	1.53×10^{3}	8.32×10^{2}	1.29×10^{4}

A detailed breakdown of query times is shown in Table 5, with total runtimes divided into:

- **Preprocessing**: D-normalization for PPD, private index setup for PPB, and OEP-based precomputations for PrivHop.
- **Online phase**: Iterative shortest path distance computations. For PrivHop, we further separate *query initialization* (i.e., computing distances from query points to boundary nodes via the HB-Index) for fine-grained analysis.

PPD suffers heavy preprocessing overhead from priority queue initialization, with runtimes exceeding 24 hours even on medium graphs, preventing main loop execution (denoted "–" in Table 5).

In contrast, PrivHop's lightweight initialization and fewer iterations, thanks to the HB-Index and dynamic pruning, demonstrate up to **6 orders of magnitude faster runtimes** than PPD and PPB on large datasets, but also outperform B-PPD and B-PPB by up to **3 orders of magnitude** under comparable graph scales.

Exp-3: Query Communication Cost. This experiment systematically evaluates the communication overhead during query processing, following the same setting as **Exp-2**. The results in Figure 8 show communication costs (capped at $1e9$ MB for readability), where our algorithm consistently reduces volumes by 1–6 orders of magnitude compared to baseline methods. This improvement stems from two key optimizations: (1) reduction in computational graph scale through precomputed indexing and (2) decreased algorithm iterations via secure pruning strategies.

Table 6 provides a detailed breakdown of communication volumes, separating preprocessing and main loop phases. Notably, in PrivHop, the query initialization phase requires transmitting only target point information, resulting in negligible communication overhead, which is therefore omitted from the statistics.

Table 6. Communication costs(MB) of all MPC shortest path distance methods.

Method	Phase	SK	SD	SW	SH	SE	WS	IT	TS
PPD	Pre.	1.39×10^{2}	2.71×10^{2}	1.69×10^{4}	1.12×10^{5}	-	-	-	-
	Iterations	4.41×10^{2}	2.05×10^{3}	3.41×10^{5}	1.44×10^{6}	-	-	-	-
	Total	5.80×10^{2}	2.32×10^{3}	3.58×10^{5}	1.59×10^{6}	-	-	-	-
B-PPD	Pre.	5.73×10^{1}	1.25×10^{2}	2.58×10^{3}	1.42×10^{4}	1.42×10^{6}	-	-	-
	Iterations	2.31×10^{1}	2.08×10^{2}	6.99×10^{4}	7.76×10^{5}	1.45×10^{8}	-	-	-
	Total	8.06×10^{1}	3.34×10^{2}	7.25×10^{4}	7.90×10^{5}	1.47×10^{8}	-	-	-
PPB	Pre.	1.01×10^{1}	1.69×10^{1}	2.04×10^{2}	1.02×10^{3}	6.62×10^{3}	1.20×10^{5}	4.14×10^{5}	7.15×10^{5}
	Iterations	3.81×10^{1}	4.83×10^{1}	7.80×10^{3}	7.46×10^{4}	4.50×10^{6}	8.52×10^{8}	5.09×10^{9}	2.75×10^{10}
	Total	4.83×10^{1}	6.52×10^{1}	8.00×10^{3}	7.56×10^{4}	4.51×10^{6}	8.52×10^{8}	5.09×10^{9}	2.75×10^{10}
B-PPB	Pre.	7.53×10^{0}	1.17×10^{1}	9.31×10^{1}	3.72×10^{2}	7.06×10^{3}	1.39×10^{4}	1.95×10^{4}	2.27×10^{5}
	Iterations	9.31×10^{0}	2.06×10^{1}	2.30×10^{2}	1.42×10^{4}	1.65×10^{6}	1.37×10^{7}	6.16×10^{6}	9.59×10^{8}
	Total	1.68×10^{1}	3.24×10^{1}	3.23×10^{2}	1.45×10^{4}	1.66×10^{6}	1.36×10^{7}	6.18×10^{6}	9.59×10^{8}
PrivHop	Pre.	7.26×10^{0}	1.10×10^{1}	8.63×10^{1}	3.48×10^{2}	6.45×10^{3}	1.31×10^{4}	1.83×10^{4}	2.13×10^{5}
	Iterations	4.52×10^{-1}	1.48×10^{0}	8.19×10^{1}	6.75×10^{1}	1.73×10^{3}	1.03×10^{4}	1.33×10^{3}	3.34×10^{4}
	Total	7.71×10^{0}	1.25×10^{1}	1.68×10^{2}	4.15×10^{2}	8.18×10^{3}	2.34×10^{4}	1.97×10^{4}	2.47×10^{5}

Exp-4: Average Running Iterations. We analyze the number of main loop iterations required during encrypted query processing. Let N be the number of vertices. In the baselines, PP-Dijkstra

requires $2N$ iterations due to normalization, while PP-Bellman requires N iterations without pruning. Their boundary-graph variants, B-PPD and B-PPB, require $2N_B$ and N_B iterations, respectively, where N_B denotes the boundary graph size.

In contrast, our algorithm employs privacy-aware iteration pruning. For cross-party queries, the iteration count is dynamically reduced by the pruning strategy. For single-party queries, if the gap between the internal shortest distance and the boundary distance is ≤ 2, no iteration is required; otherwise, the same pruning applies. Table 7 reports the average iterations over 1,000 random queries. The results show that our pruning reduces iteration counts from dataset-scale complexity to a much smaller level, which is a key contributor to the superior runtime and communication efficiency of PrivHop.

Table 7. Average iteration counts of different methods.

Alias	Method				
	PPD	B-PPD	PPB	B-PPB	PrivHop
SK	68	26	34	13	0.631
SD	124	52	62	26	1.865
SW	1778	496	889	248	10.752
SH	4K	1426	2K	713	3.385
SE	54K	17.2K	27K	8644	9.040
WS	564K	68.0K	282K	34.0K	25.590
IT	1.0M	24.2K	509K	12.1K	2.616
TS	3.4M	315.8K	1.7M	157.9K	5.510

Table 8. Information Leakage Comparison(✓ denotes leaked, ✗ denotes protected, † Protected via differential privacy).

Leaked Info.	PPD	PPB	PrivHop
#Nodes (total)	✓	✓	✗
#Edges (total)	✓	✓	✗
Query nodes	✓	✓	✓
Boundary edges	✗	✗	✓
Boundary nodes	✓	✓	✓
Bellman iters	✗	✗	✓†

Exp-5: Information Leakage Table 8 compares information leakage among the three algorithms. Our approach achieves a better privacy-utility balance: it avoids disclosing global graph structure (e.g., total nodes and edges)—leaked by both PPD and PPB only exposing the boundary edge count as a necessary cost of indexing. Although Bellman-Ford iterations remain visible, we protect them with differentially private noise. This limited and controlled leakage leads to considerable efficiency improvements.

6.3 Ablation Study

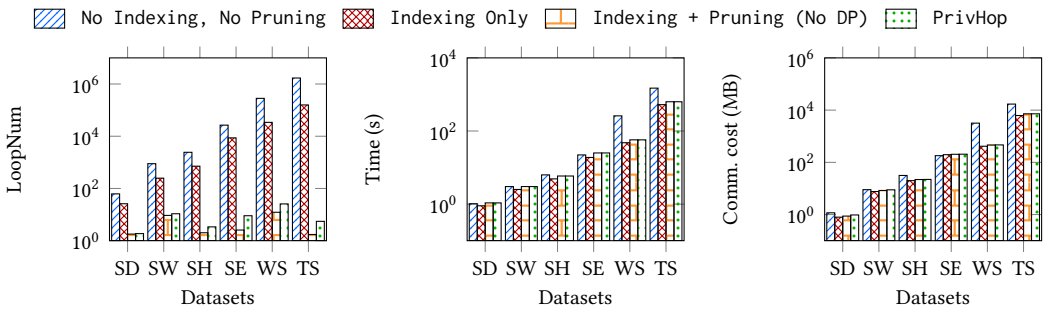
Ab-1: Component-wise Contributions. This part presents a systematic evaluation of the contributions of individual optimization components by comparing four configurations of our framework:

- **No Indexing, No Pruning**: The baseline version without any optimizations.
- **Indexing Only**: Employs precomputed HB-Index but no iteration pruning.
- **Indexing + Pruning (No DP)**: Incorporates both HB-Index and iteration pruning, but without differential privacy noise.
- **PrivHop**: The full proposed framework with all optimizations enabled.

We evaluate these configurations across three metrics: total iteration counts (Figure 9a), running time per iteration (Figure 9b), and communication cost per iteration (Figure 9c).

The results in Figure 9 reveal the following observations:

- **Indexing reduces per-iteration costs.** By transforming queries onto the smaller boundary graph, indexing dramatically lowers the computation and communication required per iteration.
- **Indexing lowers iteration counts.** Beyond per-round savings, indexing compresses the graph scale and reduces total iteration counts significantly.



(a) Iteration counts. (b) Running time per iteration. (c) Comm. cost per iteration.

Fig. 9. Component-wise ablation study on query processing performance.

- **Pruning accelerates convergence.** Dynamic iteration pruning further decreases the number of iterations by orders of magnitude.
- **Pruning with DP preserves efficiency.** Incorporating differential privacy for pruning control introduces minimal overhead while providing strong privacy guarantees.

Ab-2: Privacy-Utility Trade-off. This section quantifies the impact of the privacy budget ϵ on algorithmic efficiency. Since ϵ primarily influences the number of post-pruning execution rounds without affecting per-round performance, we report the average Bellman-Ford iteration counts under various privacy budgets in Table 9. The results clearly demonstrate the intrinsic privacy-utility trade-off: a smaller ϵ provides stronger privacy guarantees but necessitates more iterations to maintain correctness due to the increased noise. Nevertheless, the additional overhead remains within an acceptable range across all tested datasets, underscoring the practicality of our approach.

Ab-3: Handling High Privacy Protection Scenarios.

Table 9. Average iteration counts w.r.t. privacy budgets.

Alias	Privacy budget ε				
	0.1	0.5	1	5	10
SK	1.831	0.877	0.631	0.425	0.337
SD	6.527	3.911	1.865	0.798	0.516
SW	47.432	15.277	10.752	9.014	8.339
SH	33.457	6.464	3.385	0.801	0.521
SE	77.050	17.717	9.040	2.635	1.921
WS	166.110	41.436	25.590	13.559	10.454
IT	17.570	3.890	2.616	0.780	0.635
TS	11.860	7.140	5.510	1.145	0.770

Table 10. Running Time (in seconds) and Average LoopNum of PrivHop with and without Intermediate Pruning.

Alias	PrivHop		PrivHop-S	
	Running Time	Average LoopNum	Running Time	Average LoopNum
SK	4.692×10^{-1}	0.631	9.645×10^{1}	13
SD	1.680×10^{0}	1.865	2.342×10^{1}	26
SW	2.708×10^{1}	10.752	6.246×10^{2}	248
SH	1.657×10^{2}	3.385	3.490×10^{3}	713
SE	1.905×10^{2}	9.040	1.821×10^{5}	8644
WS	1.023×10^{3}	25.590	1.359×10^{6}	34.0K
IT	1.249×10^{2}	2.616	5.777×10^{5}	12.1K
TS	2.900×10^{3}	5.501	8.311×10^{7}	157.9K

Our privacy-preserving pruning scheme is based on weight-level differential privacy. Although edge-level differential privacy offers stronger guarantees, it is unsuitable for distance-based computation, as edge insertion or removal may cause drastic distance changes (e.g., from finite to infinite), resulting in sensitivity of at least $O(N)$. Ensuring correctness would require bias correction with noise on the order of $O(N \log N)$, which can negate the benefits of pruning.

Consequently, under edge-level differential privacy, pruning is infeasible and all N rounds must be executed (denoted as **PrivHop-S**). Table 10 compares the online runtime and average iteration

count of PrivHop-S and PrivHop. The results show that disabling pruning reduces our method to performance comparable to B-PPB on the boundary graph.

7 Related Work

This section reviews prior studies on shortest path distance queries and secure multi-party computation (MPC) in graph analysis, providing a comprehensive view of the field.

Shortest Path Distance Queries. Traditional shortest path query methods fall into two categories: *online search* and *index-based search*. Online search algorithms such as Dijkstra's algorithm [27] and bidirectional search [20] avoid preprocessing but must traverse the entire graph during each query. This leads to high time complexity and poor scalability on large graphs. In contrast, index-based approaches [2, 24, 25, 28, 50–52] precompute distance information to accelerate queries. For example, IS-Label [25] uses independent sets to construct vertex hierarchies and node labels, while Akanori et al. [28] leverage shortest path trees of high-degree nodes as guidance structures. PLL [2] introduces pruned BFS for efficient 2-hop index construction, further parallelized in PSL [32] to support large graphs. However, these methods require access to the full graph topology, making them unsuitable for multi-party settings. Recent work DHCA [50] extends PSL to distributed environments but still lacks robust privacy preservation mechanisms.

MPC in Graph Analysis. Secure multi-party graph analysis [5, 15, 18, 31] primarily builds on general MPC frameworks [47, 48], often representing graphs as adjacency matrices to enable data-oblivious algorithms for tasks like spanning tree and network flow [12]. However, adjacency matrices suffer from severe efficiency bottlenecks on large sparse graphs [13, 14]. To address this, GraphSC [7] proposed an edge-list-based framework that integrates parallel garbled circuits, offering a two-party computation paradigm well-suited for sparse networks. Araki et al. [8] further extended these techniques to the three-party setting using secret sharing and shuffle protocols, improving performance under the honest majority assumption. Despite these advances, general-purpose MPC frameworks remain suboptimal for specific algorithms like shortest path computation, leaving opportunities to design tailored protocols that exploit algorithmic properties for better efficiency.

8 Conclusion

We presented `PrivHop`, a privacy-preserving shortest path algorithm for secure two-party computation. `PrivHop` combines offline 2-hop index construction (which reduces graph size and secure computation overhead) with online encrypted query processing and introduces privacy-aware iteration pruning (to minimize the number of iterations), achieving substantial efficiency gains. Our method addresses two key limitations of prior work: the reliance on global graph access in traditional shortest path algorithms and the poor scalability of general MPC frameworks. Experiments on eight real-world datasets demonstrate up to six orders of magnitude improvements in both runtime and communication overhead compared to state-of-the-art methods. In future work, we plan to extend `PrivHop` to support dynamic graph updates and generalize its framework for multi-party scenarios and other privacy-preserving path-related query tasks.

Acknowledgments

This work was partially supported by NSFC under Grant 62302421, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2025A1515010439, Ant Group through CCF-Ant Research Fund, and the Guangdong Provincial Key Laboratory of Big Data Computing, The Chinese University of Hong Kong, Shenzhen.

References

[1] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast Exact Shortest-Path Distance Queries on Large Networks by Pruned Landmark Labeling. *CoRR* abs/1304.4661 (2013). arXiv:1304.4661 http://arxiv.org/abs/1304.4661

[2] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, New York, USA) *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 349–360. doi:10.1145/2463676.2465315

[3] Abdelrahaman Aly and Sara Cleemput. 2017. An Improved Protocol for Securely Solving the Shortest Path Problem and its Application to Combinatorial Auctions. *IACR Cryptol. ePrint Arch.* 2017 (2017), 971. https://api.semanticscholar.org/CorpusID:3990412

[4] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. 2013. Securely Solving Simple Combinatorial Graph Problems. In *Financial Cryptography and Data Security*, Ahmad-Reza Sadeghi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 239–257.

[5] Abdelrahaman Aly and Mathieu Van Vyve. 2022. *Securely Solving Classical Network Flow Problems.* LIDAM Reprints CORE 3192. Université catholique de Louvain, Center for Operations Research and Econometrics (CORE). https://ideas.repec.org/p/cor/louvrp/3192.html

[6] Mohammad Anagreh, Peeter Laud, and Eero Vainikko. 2021. Parallel Privacy-Preserving Shortest Path Algorithms. *Cryptogr.* 5 (2021), 27. https://api.semanticscholar.org/CorpusID:244611986

[7] Toshinori Araki, Jun Furukawa, Kazuma Ohara, Benny Pinkas, Hanan Rosemarin, and Hikaru Tsuchida. 2021. Secure Graph Analysis at Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 610–629. doi:10.1145/3460120.3484560

[8] Toshinori Araki, Jun Furukawa, Kazuma Ohara, Benny Pinkas, Hanan Rosemarin, and Hikaru Tsuchida. 2021. Secure Graph Analysis at Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 610–629. doi:10.1145/3460120.3484560

[9] Nuttapong Attrapadung, Hiraku Morita, Kazuma Ohara, Jacob C. N. Schuldt, Tadanori Teruya, and Kazunari Tozawa. 2022. Secure Parallel Computation on Privately Partitioned Data and Applications. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 151–164. doi:10.1145/3548606.3560695

[10] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings (Lecture Notes in Computer Science, Vol. 576)*. Springer, 420–432. doi:10.1007/3-540-46766-1_34

[11] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). 1–10. doi:10.1145/62212.62213

[12] Marina Blanton, Aaron Steele, and Mehrdad Alisagari. 2013. Data-oblivious graph algorithms for secure computation and outsourcing. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security* (Hangzhou, China) *(ASIA CCS '13)*. Association for Computing Machinery, New York, NY, USA, 207–218. doi:10.1145/2484313.2484341

[13] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. arXiv:1011.5425 [cs.DS] https://arxiv.org/abs/1011.5425

[14] P. Boldi and S. Vigna. 2004. The webgraph framework I: compression techniques. In *Proceedings of the 13th International Conference on World Wide Web* (New York, NY, USA) *(WWW '04)*. Association for Computing Machinery, New York, NY, USA, 595–602. doi:10.1145/988672.988752

[15] Justin Brickell and Vitaly Shmatikov. 2005. Privacy-Preserving Graph Algorithms in the Semi-honest Model. In *Advances in Cryptology - ASIACRYPT 2005*, Bimal Roy (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 236–252.

[16] David Chaum, Claude Crépeau, and Ivan Damgård. 1987. Multiparty Unconditionally Secure Protocols (Abstract). *20th STOC* 293, 462. doi:10.1145/62212.62214

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2018. TFHE: Fast Fully Homomorphic Encryption over the Torus. Cryptology ePrint Archive, Paper 2018/421. https://eprint.iacr.org/2018/421

[18] Daniele Cozzo, Nigel P. Smart, and Younes Talibi Alaoui. 2021. Secure Fast Evaluation of Iterative Methods: With an Application to Secure PageRank. Cryptology ePrint Archive, Paper 2021/207. https://eprint.iacr.org/2021/207

[19] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2011. Multiparty Computation from Somewhat Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2011 (01 2011), 535. doi:10.1007/978-3-642-32009-5_38

[20] Dennis de Champeaux. 1983. Bidirectional Heuristic Search Again. *J. ACM* 30, 1 (Jan. 1983), 22–32. doi:10.1145/322358.322360

[21] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation

[22] Cynthia Dwork. 2006. Differential Privacy. In *Automata, Languages and Programming*, Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

[23] Chenglin Fan, Ping Li, and Xiaoyun Li. 2022. Private Graph All-Pairwise-Shortest-Path Distance Release with Improved Error Rate. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 17844–17856. https://proceedings.neurips.cc/paper_files/paper/2022/file/71b17f00017da0d73823ccf7fbce2d4f-Paper-Conference.pdf

[24] Ada Wai-Chee Fu, Huanhuan Wu, James Cheng, Shumo Chu, and Raymond Chi-Wing Wong. 2012. IS-LABEL: an Independent-Set based Labeling Scheme for Point-to-Point Distance Querying on Large Graphs. arXiv:1211.2367 [cs.DB] https://arxiv.org/abs/1211.2367

[25] Ada Wai-Chee Fu, Huanhuan Wu, James Cheng, and Raymond Chi-Wing Wong. 2013. IS-Label: an independent-set based labeling scheme for point-to-point distance querying. 6, 6 (April 2013), 457–468. doi:10.14778/2536336.2536346

[26] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to play ANY mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '87)*. Association for Computing Machinery, New York, NY, USA, 218–229. doi:10.1145/28395.28420

[27] Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. 2025. Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps. arXiv:2311.11793 [cs.DS] https://arxiv.org/abs/2311.11793

[28] Takanori Hayashi, Takuya Akiba, and Ken-ichi Kawarabayashi. 2016. Fully Dynamic Shortest-Path Distance Query Acceleration on Massive Networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (Indianapolis, Indiana, USA) *(CIKM '16)*. Association for Computing Machinery, New York, NY, USA, 1533–1542. doi:10.1145/2983323.2983731

[29] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2024. Communication-Efficient Triangle Counting under Local Differential Privacy. arXiv:2110.06485 [cs.CR] https://arxiv.org/abs/2110.06485

[30] Marcel Keller and Peter Scholl. 2014. Efficient, Oblivious Data Structures for MPC. *IACR Cryptol. ePrint Arch.* 2014 (2014), 137. https://api.semanticscholar.org/CorpusID:17251251

[31] Varsha Bhat Kukkala, Jaspal Singh Saini, and S. R. S. Iyengar. 2016. Privacy Preserving Network Analysis of Distributed Social Networks. Cryptology ePrint Archive, Paper 2016/427. https://eprint.iacr.org/2016/427

[32] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Scaling Distance Labeling on Small-World Networks. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1060–1077. doi:10.1145/3299869.3319877

[33] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael W. Hicks. 2014. Automating Efficient RAM-Model Secure Computation. *2014 IEEE Symposium on Security and Privacy* (2014), 623–638. https://api.semanticscholar.org/CorpusID:875114

[34] Steve Lu and Rafail Ostrovsky. 2013. Distributed Oblivious RAM for Secure Two-Party Computation. In *Theory of Cryptography*, Amit Sahai (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 377–396.

[35] Payman Mohassel and Saeed Sadeghian. 2013. How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation. In *Advances in Cryptology – EUROCRYPT 2013*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 557–574.

[36] Benjamin Ostrovsky. 2024. Privacy-Preserving Dijkstra. Cryptology ePrint Archive, Paper 2024/988. https://eprint.iacr.org/2024/988

[37] Rafail M. Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *J. ACM* 43 (1996), 431–473. https://api.semanticscholar.org/CorpusID:7502114

[38] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Stanford InfoLab. http://ilpubs.stanford.edu:8090/422/ Previous number = SIDL-WP-1999-0120.

[39] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques* (Prague, Czech Republic) *(EUROCRYPT'99)*. Springer-Verlag, Berlin, Heidelberg, 223–238.

[40] Nicholas Pippenger and Michael J. Fischer. 1979. Relations Among Complexity Measures. *J. ACM* 26, 2 (April 1979), 361–381. doi:10.1145/322123.322138

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

[42] T. Rabin and M. Ben-Or. 1989. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing* (Seattle, Washington, USA) *(STOC '89)*. Association

for Computing Machinery, New York, NY, USA, 73–85. doi:10.1145/73007.73014

[43] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 325–342. doi:10.1145/3372297.3417274

[44] Peter Sanders and Christian Schulz. 2013. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13) (LNCS, Vol. 7933)*. Springer, 164–175.

[45] R. K. Shyamasundar. 1996. Introduction to algorithms. *Resonance* 1 (1996), 14–24. https://api.semanticscholar.org/CorpusID:123556377

[46] Andrew C. Yao. 1982. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 160–164. doi:10.1109/SFCS.1982.38

[47] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.

[48] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.

[49] Yuanyuan Zeng, Yixiang Fang, Kun Chen, Yangfan Li, and Chenhao Ma. 2025. Efficient Maintenance of 2-Hop Labeling Index on Dynamic Small-World Graphs. *Proc. VLDB Endow.* 18, 7 (Aug. 2025), 2005–2017. doi:10.14778/3734839.3734840

[50] Yuanyuan Zeng, Chenhao Ma, and Yixiang Fang. 2024. Distributed Shortest Distance Labeling on Large-Scale Graphs. 17, 10 (June 2024), 2641–2653. doi:10.14778/3675034.3675053

[51] Mengxuan Zhang, Lei Li, Wen Hua, and Xiaofang Zhou. 2021. Efficient 2-Hop Labeling Maintenance in Dynamic Small-World Networks. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 133–144. doi:10.1109/ICDE51399.2021.00019

[52] Bolong Zheng, Yong Ma, Jingyi Wan, Yongyong Gao, Kai Huang, Xiaofang Zhou, and Christian S. Jensen. 2023. Reinforcement Learning based Tree Decomposition for Distance Querying in Road Networks. In *2023 IEEE 39th International Conference on Data Engineering (ICDE) (Proceedings of the International Conference on Data Engineering)*. IEEE (Institute of Electrical and Electronics Engineers), United States, 1678–1690. doi:10.1109/ICDE55515.2023.00132 Publisher Copyright: © 2023 IEEE.; 39th IEEE International Conference on Data Engineering, ICDE 2023 ; Conference date: 03-04-2023 Through 07-04-2023.

Received July 2025; revised October 2025; accepted November 2025